# Scalable Augmented Reality on Mobile Devices:

## Applications, Challenges, Methods and Software

Xin Yang and K.T.Tim Cheng

Dept. of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA, USA
xinyang@umail.ucsb.edu, timcheng@ece.ucsb.edu

### 1. Applications

In recent years, mobile devices such as smartphones and tablets have experienced phenomenal growth. Their computing power has grown enormously and the integration of a wide range of sensors, e.g. compass, accelerometer, gyroscope, GPS, etc., have significantly enriched these devices' functionalities. The connectivity of smartphones has also gone through rapid evolution. A variety 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. These mobile handhelds equipped with cameras, sensors, low-latency data networks, and powerful multi-core application processors can now run very sophisticated augmented reality (AR) applications. There are already a very rich collection of mobile AR (MAR) apps for visual guidance in assembly, maintenance and training, interactive books, multimedia-augmented advertising, contextual information-augmented personal navigation, etc., that can be used anytime and anyplace.

A scalable MAR system, that has the ability to identify objects and/or locations from a large database and track the pose of a mobile device with respective to the physical objects, can support a wide range of MAR apps. Such a system can enable applications such as nation-wide

multimedia enhanced advertisements printed on papers, augmenting millions of book pages in a library, and recognizing millions of world-wide places of interests and providing navigation or contextual information, etc. Broadly, scalable MAR apps can be categorized into three classes according to the underlying techniques they rely on: computer vision-based, sensor-based and hybrid.

### 1.1 Computer Vision-based MAR apps

Vision-based MAR apps rely on images captured by mobile cameras, and use vision-based recognition and tracking algorithms to identify physical objects and further link them to appropriate virtual objects. Specially, a conventional vision-based MAR pipeline consists of three main steps: 1) deriving a set of features from a captured image frame and matching it to a database to recognize the object (i.e. recognition); 2) tracking the recognized object from frame to frame by matching features of consecutive frames (i.e. tracking); and 3) building precise coordinates transforms (e.g. RANSAC [Fischler et al. 1981] or PROSAC [Chum et al. 2005]) between the current image frame and the recognized object in the database (i.e. pose estimation). Some details of vision-based MAR approaches will be presented in Sec. 3.1.

With rapid advances in real-time object recognition and tracking capability, vision-based MAR apps are increasingly popular. To date, we have seen vision-based apps permeating into marketing (e.g. multimedia-augmented advertising), education (e.g. interactive books), social network, search engine and mobile health. In the following, we highlight some exemplar vision-based MAR apps, illustrating the rich and exciting opportunities in these areas.

In marketing and education, vision-based MAR apps can significantly improve the user experiences by overlaying related multimedia information or digital operations (e.g. mouse click) on paper-based advertisements (such as newspapers, handbills, and movie posters) and books.

Picture the following app for reading a book or newspaper: through the phone camera and touchscreen, you could highlight any text or figure, which is automatically recognized and used for search on the web. Such search results are then displayed on the viewfinder to facilitate the reading. Vision-based MAR technology can overlay extra 3D pictures, videos, and sounds related to the recognized object on the viewfinder in real-time to provide the user an augmented sense which mixes reality and virtuality. For example, the user can see flowers that are not actually blooming when pointing the camera to a picture of bare branches or can link handouts with the augmented digital information. Regarding the latter example, systems which facilitate the development of scalable mobile augmented papers can be found in *EMM* [Yang et al. 2011a; Yang et al. 2011b], *FACT* [Liao et al. 2010] and *Mixpad* [Yang et al. 2012] from FXPAL, and *Mobile Retriever* [Liu et al. 2008].

A wide range of MAR apps for social mining heavily rely on face recognition technologies. For instance, *SocialCamera* from Viewdle, allows smartphone users to take photos with built-in, instant tagging. It uses face recognition technology to identify people among the friends in your face database, and tag them automatically. Therefore, it will only take a few clicks to share tagged mobile photos with friends through Facebook, Line, MMS or email.

Google Goggles is a mobile AR app that conducts searches based on pictures taken by the phone. Its visual search relies on real-time image recognition to identify objects in the picture as the starting point for search, often referred to as ‘query-by-image’. For example, a user can take a picture of a famous landmark or a painting to search for information about it, 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.

There already exist several vision-based MAR apps for mobile health. These apps could recognize food objects on the camera viewfinder, analyze their calories and nutrition data, and then display the information using overlaid text and charts on the viewfinder. While object recognition for most food items are quite feasible, the key challenge for this app is the ability to estimate object volume (the serving size for nutrition analysis). Accurate estimation would require knowing the distance between the phone camera and the object.

### 1.2 Sensor-based MAR Apps

Sensor-based MAR apps rely on the input from sensors (e.g. compass, GPS, NFC, accelerometer, and gyroscope) to identify and track the geographical position of a mobile device and its orientation. Using this information, specific contextual information, such as the route to a destination, nearby shops, points of interests, etc. can be brought out from a database and overlaid on top of the real-world scene displayed on the viewfinder. With more types and more accurate sensors integrated into each new generation of smartphones, sensor-based MAR apps have become increasingly important and are gaining popularity, especially in the application area of personal navigation.

An exemplar navigation app for outdoor activity is *Theodolite* which utilizes a phone display as a viewfinder to overlay GPS data (coordinates and elevation), compass heading, attitude, and time data. It serves as a compass, GPS, map, zoom camera, rangefinder, and inclinometer. The obvious uses for such an app include backcountry pursuits, skiing, fishing or boating navigation, surveying, landscaping, tracking the trajectory of objects, as well as search and rescue. *Spyglass*

is a similar app for outdoor activities. It overlays positional information using available sensors of the phone and could be used as a waypoints tool, sextant, compass, range finder, speedometer and inclinometer.

Other sensor-based MAR apps which serve for navigation focus on the scenario of car driving. For instance, *Wikitude Drive* is a MAR navigation system for which computer-generated driving instructions are drawn on top of the reality - the real road the user is driving on. Navigation thus takes place in real-time in the smartphone's live camera image. This app solves a key problem that existing navigation systems have - the driver no longer needs to take his/her eyes off the road when looking at the navigations system. Other similar apps include *Route 66 Maps+Navigation* which provides an amalgamation of comprehensive 3D maps and AR navigation to bring fun and informative experience to drivers and *Follow Me* which can trace the user's exact route on road, backed with real-time graphics and a virtual car that leads the user all the way to the destination.

In addition to navigation, sensor-based MAR is also applicable to educational applications. A famous example is *Star Walk*, an interactive astronomy guide offering augmented reality of the sky with the actual sky outside. Using this app, users can align the physical sky, to the sky shown on the display of the phone/tablet. This allows for pin-point precision for tracking satellites, finding stars, or finding constellations, offering an attractive educational tool for students, part-time star gazers or astronomers. As the user adjusts the orientation of phone/tablet towards the sky, with the help of the accelerometer and gyroscope sensors, which offer highly accurate positioning, the sky shown on the display will move along to match the physical sky. Finding celestial objects becomes easy - simply moving the viewfinder over the object in the real world view and clicking on it. The user can also search for an object - Jupiter, a satellite, or a specific

constellation, to have it instantly come up on the viewfinder. Then the app will guide the user to adjust the orientation of the device to match it up with the object in the real sky.

### 1.3 MAR apps based on Hybrid Approaches

Sensor-based and vision-based methods have distinct advantages and disadvantages. Sensor-based method can obtain geographical locations and relative pose of a mobile device from built-in inertial sensors, requiring few complex calculations. Therefore, it is attractive for any mobile platform with limited computing and memory resources. However, its accuracy is usually low due to the low-cost inertial sensors used in mobile handhelds. On the other hand, vision-based method can achieve much better accuracy at a cost of high computational and memory complexity.

Many scalable MAR apps which demand both high accuracy and high efficiency employ hybrid methods which combine the complementary advantages of both approaches. *InterSense* [Naimark et al. 2002] is a good example which uses a sensor-based inertial tracker to predict the potential positions of markers and then leverages vision-based image analysis for the candidate regions to refine the results. There are several other examples using magnetic and gyro sensors to stabilize the tracking systems [Ribo et al. 2002; Uchiyama et al. 2002; Jiang et al. 2004; Reitmayr et al. 2007].

## 2. Challenges

Despite advances in computer vision and signal processing algorithms as well as mobile hardware, scalable MAR remains very challenging due to the following reasons:

a) 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 perform computationally intensive vision tasks such as sophisticated feature extraction and image recognition algorithms. Graphics processing units (GPUs), which have been built into most application processors, can help speed up processing via parallel computing [Cheng et al. 2013; Terriberry et al. 2008], but most feature extraction and recognition algorithms are designed to be executed sequentially and cannot fully utilize GPU capabilities. In addition, 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 object recognition using a large database. In order to realize efficient object recognition, 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 MAR systems 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 object recognition and pose estimation on the server side, and then sending the estimated pose and associated digital data 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.

b) For most algorithms, it is very challenging to achieve both good accuracy and high efficiency. As mention in previous section, sensor-based methods can achieve good efficiency but its performance is often limited by the low precision of sensors used in mobile devices, limiting their applicability for apps demanding high recognition rate and tracking accuracy. In addition, sensor-based approaches do not provide information about the objects in the camera picture. As a result, the presented information could only be related to the direction and position of the device, not to a specific object. On the other hand, vision-based methods usually require significant computation and memory space to process the image which consists of a large number of pixels, to match against a large database and to estimate geometric transformation between the object within a captured image and the recognized database object. A hybrid approach which integrates vision-based and sensor-based methods can potentially combine their complementary advantages; however, designing a fusion solution which optimizes accuracy, efficiency and robustness is not a trivial task at all.

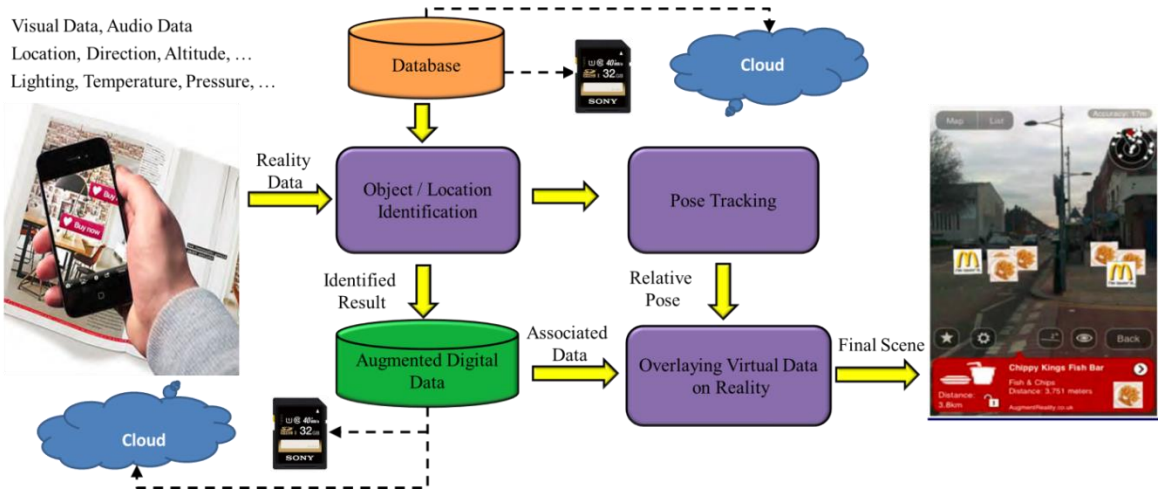


**Fig. 1**. A General pipeline for scalable augmented reality on mobile devices

## 3. Pipelines and Methods

Augmented reality links proper context with a real-world object/location and adjusts the pose of the virtual data so as to render it at a correct position and with a correct orientation on a real-world image. In order to do this, the system needs to know where the user is and what the user is looking at. More specifically, a MAR system needs to determine the location and orientation of the mobile device, calculate the relative pose (location and orientation) of the device in real time, and then render virtual objects in the correct place. Fig. 1 illustrates a general pipeline of a scalable MAR system. Given the physical data (e.g. an image, the geographical location, the direction, etc.) captured from a mobile handheld, the system identifies objects/locations of interests captured by the mobile device. The identification process is usually conducted by processing the captured physical data to generate a description delineating the real-world scene, and then matching the description to a large database. The database object which best matches the feature of the captured object is considered as the recognized object and an initial pose is generated. After recognition, the movement of the recognized object is tracked. The recognizer

and the tracker of AR are executed alternatively to complement each other: the recognizer is activated whenever the tracker fails or new objects/locations occur and the tracker bridges the recognized results and speeds up the process by avoiding unnecessary recognition task which is more computational expensive than tracking. Databases which contain objects for recognition and associated digital data can be stored either in local storage space or in the cloud, depending on the size of databases, available local storage resources and performance requirement.

### 3.1 Visual Object Recognition and Tracking

As a camera is a built-in component in most MAR systems, methods based on visual object recognition and tracking are of special interests in MAR. Researchers in image processing, computer vision, pattern recognition and machine learning have developed a considerable number of object recognition and tracking methods. Some of successful ones for AR rely on predefined markers that consist of easily detectable patterns. In a marker-based MAR system, each object of interest is associated with a particular marker. By detecting, identifying and tracking the marker using image analysis techniques, an AR system can recognize the associated object and obtain the correct scale and pose of the camera relative to the physical object.

Another category of approaches for visual object recognition and tracking is based on visual feature extraction and matching. These methods use a pre-built database containing pre-computed features for all objects of interests. For each image frame captured by a camera, the system extracts the same type of features from the image and then matches the feature to database features. The corresponding object of the best match in the database is then reported as the recognized object. For tracking the recognized object, visual features extracted from consecutive frames are derived and matched to track the movement of the object from frame to frame. Marker-based methods and feature-based methods, having distinct advantages and

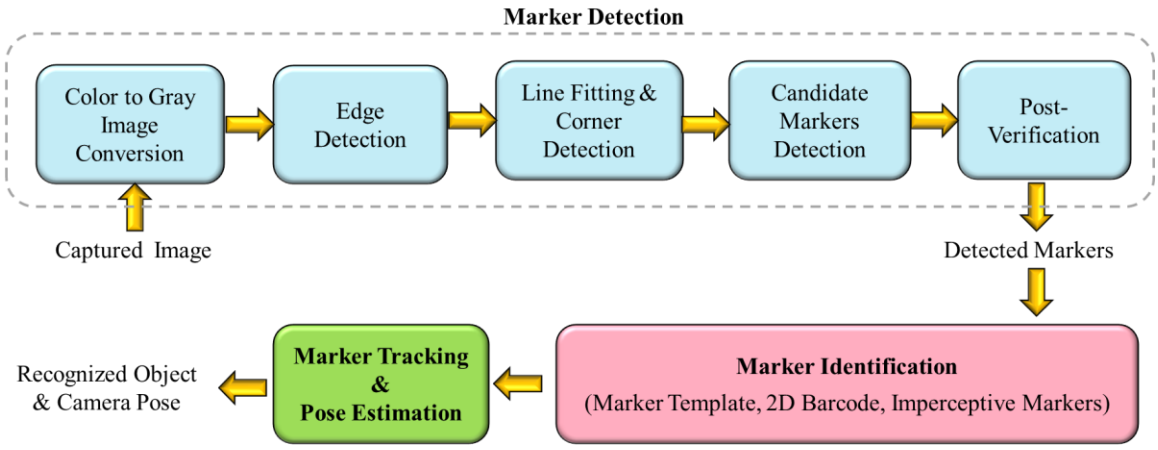


**Fig. 2**. Illustration of marker-based MAR pipeline.

limitations, are not mutually exclusive and thus can be combined. In the following, we describe the procedure pipeline for each category and give an overview of state-of-the-art methods for each step in the pipeline.

### 3.1.1 Marker-based Methods

A good marker design should facilitate a quick and reliable detection, identification and tracking of the marker under various imaging conditions. It has been shown that black and white markers are much more robust to various photometric changes and background clutters than chromatic markers. Previous studies have also concluded that a square is the simplest and most suitable shape for a maker. This is because a system needs at least four pairs of corresponding points between two detected markers from two image frames to estimate the camera pose. Four corner points of a square are sufficient for homography estimation and can be reliably detected as intersections of edge lines. Therefore, most existing MAR systems leverage black-and-white and squared-shaped markers for recognition and tracking tasks, and in this section we focus our discussion for this type of markers.

Typically, a marker-based MAR system consists of three key components, as shown in Fig. 2: 1) marker detection (blue blocks) in which regions that are likely to be markers are localized, 2) marker identification (pink block) in which candidate marker regions are verified and recognized, and 3) marker tracking and pose estimation (green block) in which four pairs of corner points are used to estimate the homography transformation.

**a. Marker Detection**

The goal of marker detection is to find positions of markers, to delineate the boundary of each marker and to localize corner points of markers in an image. A basic marker detection procedure is illustrated by blue blocks in Fig. 2. First, an RGB image is converted into an intensity image. Second, edge detection is performed on the grayscale image to obtain a list of edge segments. Popular edge detection operators include canny operator [Canny 1986], sobel operator [Gonzalez et al. 1992], and Laplacian operator [Haralick 1984]. Line fitting is then conducted based on the detected edges, which detects intersections of two lines as potential corners. Then, regions which are defined by four straight connecting lines and consist of four corners are considered as potential marker candidates. Finally, candidate markers are verified by some effective and fast-to-compute criteria. Candidates which do not pass the verification procedure are removed as false positives to avoid unnecessary processing for non-marker regions in the following identification and tracking steps. A simple yet effective verification scheme is based on the size of a candidate marker region, i.e. rejecting small regions with a limited number of pixels. Since small regions are either false positives or true markers but are too far away from a camera to achieve reliable pose estimation. Another fast verification criterion is based on the histogram of a region. As a maker consists of black and white colors, its histogram should be bipolar. Based on this criterion, we could easily remove false positives which have a relatively


**Fig. 3**. Illustration of 2D markers. (a) Template marker. (b)-(d) Barcode markers: QR code, DadaMatrx and PDF417.

uniform histogram. In addition, depending on the particular appearance of a marker, a system could quickly reject obvious non-markers with high confidence. For instance, 2D barcode markers have a number of sharp edges inside a marker (i.e. edges between white and black cells). A heuristic yet efficient criterion is to examine the frequency of intensity changes in two perpendicular directions: if the frequency is below a predefined threshold, this region is rejected as a false positive.

**b. Marker Identification**

A scalable MAR system utilizes a large database, in which each element is linked to corresponding virtual data, digital interactions, etc. The goal of marker identification is to find a matched element from the database for a captured marker so that the system knows which virtual data should be overlaid on top of the current real-world scene. Techniques used for marker identification depend on the marker type. Two popular types of markers widely used for AR are template-based markers and 2D barcode markers.

***Template markers*** are black and white markers which have a simple image inside a black border (as shown in Fig. 3 (a)). Identification of a template marker is typically based on template matching, i.e. comparing the marker image captured from the detection process to a database consists of all marker templates. Specifically, a marker region is first cropped from the captured

image and then it is rectified to be a square and scaled to the same size as marker templates in the database. After that, the rectified and scaled marker is rotated into four different orientations ($0^o$, $90^o$, $180^o$, $270^o$). For each orientation, an overall best match in the database with the highest similarity value to the rotated marker is identified. If the highest similarity value is greater than a threshold, the region is considered as a recognized marker; otherwise its unrecognized and the system rejects it. Several similarity metrics have been proposed for matching, such as the sum of squared differences (SSD), mutual information, etc. [Ashby et al. 1988].

One major limitation of template matching is that a system needs to match a marker candidate region against all marker templates in the database. Matching each pair of marker images involves processing of all corresponding pixels of the two images. As a result, the total runtime for template matching is non-trivial and grows linearly with the number of templates in the database, prohibiting its usage for a large database. To speed up the matching process, markers are often down-sampled to a small size, i.e. 16×16 or 32×32, to reduce the amount of pixels needs to be compared. However, the scalability and accuracy of a MAR system could be greatly restricted by the size of quantized markers. For a quantized image size of 16x16 inside a maker, the maximum number of distinct patterns can be generated is $(16\times16)^2$ =65536 (each pixel can be either 1 or 0). In other words, the upper bound on the number of distinct markers is limited to 65536. In practice, due to the photometric changes, inaccuracy in the detection process and other noise sources, the number of markers that can be correctly and reliably detected could be even smaller than this number. Due to these limitations, template markers are not suitable for scalable MAR apps which use a large database.

***2D barcode markers*** are markers consisting of frequently-changed black and white data cells and possibly a border or other landmarks (as shown in Figs. 3 (b) – (d)). A system identifies a 2D

barcode marker by decoding the encoded information in it. Typically, the decoding process is performed by sampling the pixel values from the calculated center of each cell, and then resolves the cell values using them. The resolved cell value can be either interpreted as a binary number (i.e. 0 or 1) or can link to more information (e.g. ASCII characters) via a database.

Popular 2D barcode standards include QR Code [Information Technology 2006a], DataMatrix [Information Technology 2006b], and PDF417 [Information Technology 2006c] which were originally developed for logistics and tagging purposes but are also used for AR apps. In addition to these three standards which will be briefly described in the following, there are many other standards (e.g. MaxiCode, Aztec Code, SPARQCode, etc.) which might be used for tracking in some applications too.

QR Code (Fig. 3 (b)) is a two-dimensional barcode created by the Japanese corporation Denso-Wave in 1994. QR is the abbreviation for Quick Response, as the code is intended for high-speed decoding. QR Code became popular for mobile tagging applications and is the de-facto standard in Japan. QR Code is flexible and has large storage capacity. A single QR Code symbol can contain up to 7089 numeric characters, 4296 alphanumeric characters, 2953 bytes of binary data or 1817 Kanji characters. Therefore, QR codes are very suitable for large-scale MAR apps.

DataMatrix (Fig. 3 (c)) is another popular barcode marker which is famous for marking small items such as electronic components. The DataMatrix can encode up to 3116 characters from the entire ASCII character set with extensions. The DataMatrix barcode is also used in mobile marketing under the name *SemaCode*.

PDF417 (Fig. 3 (d)) was developed in 1991 by Symbol (recently acquired by Motorola). A single PDF417 symbol can be considered multiple linear barcode rows stacked above each other.

A single PDF417 symbol can theoretically hold up to 1850 alphanumeric characters, 2710 digits or 1108 bytes. The exact data capacity depends on the structure of the data to be encoded; this is due to the internal data compression algorithms used during coding. The ratio of the widths of the bars (or spaces) to each other encode the information in a PDF417 symbol. For that reason, the printing accuracy and a suitable printer resolution are important for high-quality PDF417 symbols. This also makes PDF417 the least suitable for AR applications where the marker is often under perspective transformation.

**c. Marker Tracking**

The main idea of AR is to present virtual objects in a real environment as if they were part of it. Virtual objects should move and change its pose accordingly with the movement of a mobile camera. Tracking the camera pose (i.e. camera location and orientation) in realtime is required in order to render the virtual object in the right scale and perspective. The pose of a camera relative to a marker in the real scene can be uniquely determined from a minimum of four corresponding points between the marker in the real scene and the marker on the camera image plane. Note that the four points used for determining the camera pose need to be *coplanar* but *non-collinear*. Marker-based object tracking uses the four corners of a square marker, which can be reliably detected, for this purpose. We define a transformation $T$ between a camera and a marker as

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

where $[X\ Y\ Z\ 1]^T$ is a homography representation for a marker corner's coordinates in the earth coordinate system and $[x\ y\ 1]^T$ is its projected coordinates on the image plane. Once an initial camera pose is obtained, the system can keep tracking the marker on the image plane by

constructing corner correspondences between consecutive frames and computing the transformation matrix between two frames based on the correspondences.

**d. Discussions**

2D barcode identification directly decodes the information in a marker without demanding enormous amount of computations for image matching. In addition, marker-based tracking only needs to detect four corners of a marker and estimate the camera pose according to Eq. 1. Therefore, the barcode marker-based method is time efficient so as to provide real-time performance for many MAR apps. Furthermore, 2D barcode markers have a large storage capacity and thus can support applications which require high scalability. However, barcode markers need to be printed on or attached to objects beforehand for association with specific contents. Thus they are visually obtrusive and for some outdoor scenarios (e.g. landmarks) attaching markers to objects is not feasible. Moreover, marker-based methods are sensitive to occlusion. These limitations may lead to a poor user experience. Feature-based methods can overcome these limitations, while the slower speed and greater memory usage could be two major issues.

### 3.1.2 Feature-based Method

Local features (an example shown in Fig. 4(a)) have been used for various computer vision apps, including object recognition, tracking, image registration, etc. Different from the conventional 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, it has been demonstrated that local features are more robust to various geometric and photometric transformations, occlusion, and background clutters. As a result, to ensure a more satisfactory user experience, many existing MAR systems choose a local feature representation for object recognition and tracking.

A typical flow for a local feature-based MAR is as follows. In the offline phase, local feature extraction is performed for every database image. An indexing structure which encodes feature descriptors of all database images is constructed. In the recognition phase, local features of a captured image is first extracted, each of which is then used to query the database using the indexing structure for finding a matching local feature in the database. The database image which has the most matching features with the capture image is considered as the recognized target. An initial camera pose is then estimated based on the corresponding matches using RANSAC or PROSAC algorithms. In the tracking phase, local features between consecutive frames are compared. Corresponding matches are used to track the movement of cameras between frames. In the following, we review state-of-the-art methods for each step.

**1) Local Feature Extraction**

The efficiency, robustness and distinctiveness of local feature representation significantly affect the user experience and scalability of a MAR system. In this part, we review relevant work about interest point detection and description, and present their latest advances for scalable MAR. However, there is an enormous breadth and amount for results in this field. With limited space, we can only afford reviewing a small subset of representative results that are most relevant to the application of scalable MAR.

**a. 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 can be visible in both images, 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 time-critical applications, and 4) *quantity*, i.e. a typical image should contain a sufficient number of points to cover the target object, so that it can be recognized even under partial occlusion.

A wide variety of interest point detectors exist in the literature. Some light-weight detectors [Rosten et al. 2006] aim at high efficiency to target applications which demand real-time performance and/or mobile hardware platforms which have limited computing resources. However, the performance of these detectors is relatively poor. As a result, it requires pose-verification to exclude false matches in the matching phase which often incurs a non-trivial runtime. On the other hand, several high-quality feature detectors [Bay et al. 2006; Bay et al. 2008, Lowe 2004] 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 object recognition. However, the computational complexity of these detectors is usually very high, making them inefficient on a mobile device. Some recent efforts, e.g. [Yang et al. 2012a], have been made to adapt these feature detection algorithms onto mobile devices and optimize their performance and efficiency for MAR. Due to space limitation, we review only the most representative work for the light-weight detector, the high-quality detector

and algorithm adaptation. A thorough survey on local feature based detectors that can be found in [Tuytelaars et al. 2008].

**Light-Weight Detector: FAST**

The FAST (Features from Accelerated Segmented Test) detector, proposed by Rosten and Drummond [Rosten et al. 2006], has become popular recently 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) of around a central point, each pixel is labeled from integer number 1 to 16 clockwise: if the intensities of $n$ ($n \geq$ threshold) consecutive pixels are all higher or all lower than the central pixel, 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 *non-maximum suppression* step (i.e. if the response value of a point is the local maximum within a small region, this point is considered as a feature point). Since the FAST detector only involves a set of intensity comparisons with little arithmetic operations, it is highly efficient.

The FAST detector is not invariant to scale changes. To achieve scale invariance, Rublee et al. [Rublee et al. 2011] proposed to employ a scale pyramid of an image and detect FAST feature points at each level in the pyramid. FAST could incur large responses along edges, leading to a lower repeatability and distinctiveness compared to high-quality detectors such as SIFT [Lowe 2004] and SURF [Bay et al. 2006; Bay et al. 2008]. To address this limitation, Rublee et al. employed a Harris corner measure to order the FAST feature points and discard those with small responses to the Harris measure.

**High-Quality Detector: SURF**

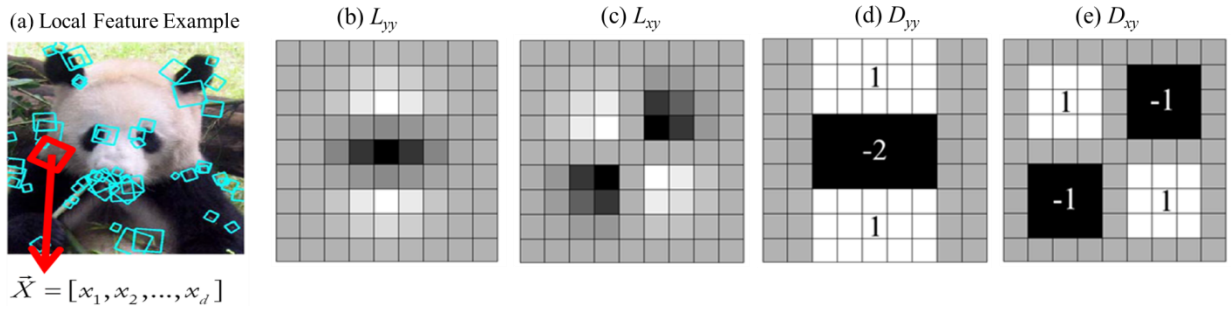


Fig. 4. (a) An exemplar image overlaid with detected local features. (b) and (c) are the discretized and cropped Gaussian second-order partial derivative in the *y*-direction and the *xy*-direction respectively; (d) and (e) are SURF box-filter approximation for $L_{yy}$ and $L_{xy}$ respectively

The SURF (Speed-Up Robust Feature) detector, proposed by Bay et al. [Bay et al. 2006; Bay et al. 2008], 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 $H(X, \sigma)$:

$$H(X,\sigma) = \begin{bmatrix} L_{xx}(X,\sigma) & L_{xy}(X,\sigma) \\ L_{xy}(X,\sigma) & L_{yy}(X,\sigma) \end{bmatrix} \tag{2}$$

where $X=(x, y)$ is a pixel location in an Image $I$, $\sigma$ is a scale factor, $L_{xx}(X,\sigma)$ is the convolution of the Gaussian second-order derivative in $x$ direction, similarly for $L_{yy}$ and $L_{xy}$ (see Figs. 4 (b) and (c)).

To speed up the process, a SURF detector approximates the Gaussian second-order partial derivatives with a combination of box filter responses (see Figs. 4 (d) and (e)), computed using the integral image technique [Simard et al. 1998]. The approximated derivatives are denoted as $D_{xx}$, $D_{xy}$ and $D_{yy}$ and accordingly the approximate Hessian determinant is:

$$\det\left(H_{approx}\right) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{3}$$

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 3×3×3 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 1 compares the runtime performance of a FAST detector and a SURF detector running on a mobile device (Motorola Xoom1) and a laptop (Thinkpad T420) respectively. Running a FAST detector takes 170ms on a Motorola Xoom1 and 40ms on an i5-based laptop, yielding a 4x speed gap. However, running a SURF detector on them takes 2156ms and 143ms respectively, indicating a 15x speed gap.

**Table 1. Comparison of FAST and SURF Detector on Mobile Device and PC**

| Time / Detector | Mobile Device (ms) | PC (ms) | Speedup |
|---|---|---|---|
| FAST Detector | 170 | 40 | 4x |
| SURF Detector | 2156 | 143 | 15x |

Although FAST detector is more efficient than SURF, it cannot match SURF's robustness and distinctiveness. As a result, it usually fails to achieve satisfactory performance for MAR apps which demand high recognition accuracy from a large databases and/or handling content with large photometric/geometric changes.

**Algorithm Adaptation: Accelerating SURF on Mobile Devices**

There are several techniques for improving SURF's efficiency by exploiting coherency between consecutive frames [Ta et al. 2009], employing graphics processing units (GPUs) for parallel computing or optimizing various aspects of the implementation [Terriberry et al. 2008]. An interesting solution proposed recently [Yang et al. 2012a] is to analyze 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 existing SURF algorithm and common mobile hardware platforms are identified as the sources of significant performance degradation:

- *Mismatch between data access pattern and small cache size of a mobile platform*. 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. 5 (a)). The data cache size of a mobile application processor (AP), typically 32KB for today's devices, is too small to cache all memory rows 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 algorithm and high pipeline hazard penalty of the mobile platform*. To identify a dominant orientation, a SURF

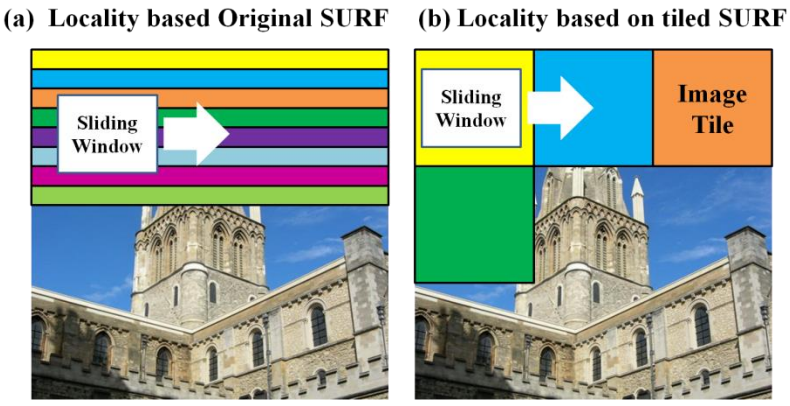


Fig. 5. Illustration of data locality and access pattern in (a) the original SURF detector, and (b) the tiled SURF. Each color represents data stored in a unique DRAM row. In the original SURF a sliding window needs to access multiple DRAM rows, leading to frequent cache misses, while in tiled SURF, all required data within a sliding window can be cached.

detector analyzes 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. 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 usually not as advanced as that of a laptop or desktop processor. Consequently, it incurs high pipeline hazard penalties, yielding significant performance degradation.

To address the problem caused by the mismatch between data access pattern of SURF and the small cache size of a mobile CPU, a tiled SURF was proposed in [Yang et al. 2013a] which divides an image into tiles (illustrated in Fig. 5 (b)) 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 [Yang et al. 2013a] to remove data-dependent branch operations. The first solution is to use an alternative implementation, i.e. 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 solution 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) [Rosin 1999]. The gradient-moment-based method may slightly degrade the robustness of a SURF detector, but can greatly improve the speed on mobile platforms.

Tables 2 and 3 compare the runtime cost and the Phone-to-PC runtime ratio between the original SURF and adapted SURF respectively [Yang et al. 2012a]. The Phone-to-PC ratio, defined in Eq. (4), 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.

$$Phone\text{-}to\text{-}PC\ ratio = \frac{\text{runtime on mobile platform}}{\text{runtime on x86-based PC}} \tag{4}$$

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 2 and 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 the three devices. The reduction in Phone-to-PC ratio indicates that the mismatch between 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 3$^{rd}$ to 5$^{th}$ rows of Tables 2 and 3 compare the results of oriented SURF (O-SURF) with branch operations, O-SURF using a lookup table and using *GMoment* [Rosin 1999], respectively. Results 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 2 and 3 show the results with the application of both two adaptations to O-SURF: comparing to original SURF, the two adaptations can reduce the runtime on mobile platforms by 6X ~ 8X.

**Table 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 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 |

**b. 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 have been the SIFT descriptor and the SURF descriptor. SIFT and SURF have successfully demonstrated their good robustness and distinctiveness in a variety of computer vision applications. However, the computational complexity of SIFT is too high for real-time application with tight time constraints. Despite that 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. Recently, the booming development of real-time mobile apps stimulates a rapid development of binary descriptors that are more compact, and faster to compute than SURF-like features while maintaining a satisfactory feature quality. Notable work includes BRIEF [Terriberry et al. 2008] and its variants rBRIEF [Rublee et al. 2011], BRISK [Leutenegger et al. 2011], FREAK [Alahi et al. 2012], and LDB [Yang et al. 2014a; Yang et al. 2014b; Yang et al. 2012b]. In the following, we review three representative descriptors: SURF, BIREF and LDB.

**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 has been proposed in [Terriberry et al. 2008] that takes advantage of the computational power available in current CUDA enabled graphics cards. This GPUSURF implementation has been reported to perform feature extraction for a 60×480 image at a frame rate upto 20 Hz, thus making feature extraction

a truly affordable processing step. However, to date, most mobile GPU cores do not support CUDA and thus porting an implementation from desktop-based GPUs to mobile GPUs remains a tedious task.

**BRIEF: Binary Robust Independent Elementary Features**

The BRIEF descriptor, proposed by Calonder et al. [Calonder et al. 2010], 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 $\tau$ is defined and performed on a patch p of size $S \times S$ as

$$\tau(p; x, y) = \begin{cases} 1 & if \quad I(p, x) < I(p, y) \\ 0 & otherwise \end{cases} \tag{5}$$

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

$$\sum_{1 \le i \le n_d} 2^{i-1} \tau(p; x_i, y_i) \tag{6}$$

By construction, the tests of Eq. 6 consider only the information at single pixels; therefore the resulting BRIEF descriptors are very sensitive to noises. To increase 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 [Calonder et al. 2010], the authors experimented with five sampling geometries for determining the spatial arrangement. 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 - give the highest recognition rate.

**LDB: Local Difference Binary**

Binary descriptors such as BRIEF and a list of enhanced versions of BRIEF [Rublee et al. 2011; Leutenegger et al. 2011; Alahi et al. 2012] 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 them more suitable for real-time applications and handheld devices. However, these binary descriptors 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 incurs an enormous number of false matches when matching against a large database. Expensive post-verification methods (e.g. RANdom SAmple Consensus (RANSAC) [Fischler et al. 1981]) are usually required to discover and validate matching consensus, increasing the runtime of the entire process.

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 high 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 Figs. 6 (a) and (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. 6 (c)). Coarse-level grids can cancel out high-frequency noise while fine-level grids can capture

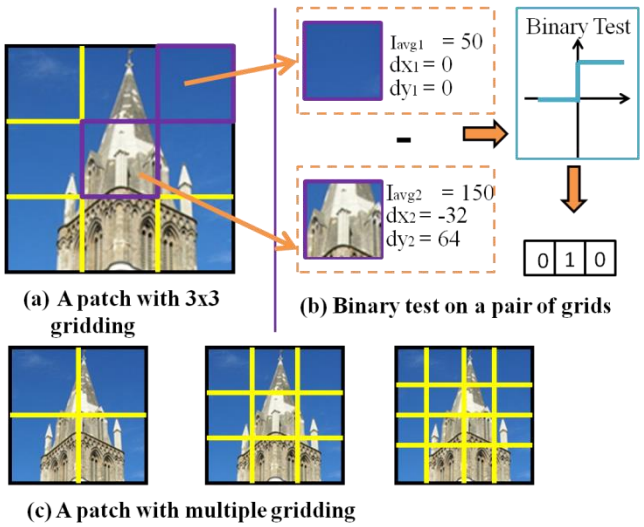


Fig. 6: Illustration of LDB extraction. (a) An image patch is divided into 3x3 equal-sized grids. (b) Compute the intensity summation (I), gradient in $x$ and $y$ directions ($d_x$ and $d_y$) of each patch, and compare $I$, $d_x$ and $d_y$ between every unique pair of grids. (c) 3-level gridding (with 2×2, 3×3 and 4×4 grids) is applied to capture information at different granularities.

detailed local patterns, thus enhancing distinctiveness. Third, LDB leverages a modified AdaBoost method [Yang et al. 2014b] 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 very fast. 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.

**2) Local Feature based Object Recognition**

To recognize objects in a captured image, a system matches each feature descriptors on a captured image to database features in order to find its nearest neighbor (NN). If a pair of NNs pass the verification criteria (i.e. the similarity between a feature and its NN being above a predetermined threshold, complying with a geometric model, etc.), this feature pair is considered

a matched pair; otherwise it is discarded as a false positive. 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 object recognition, ensuring a satisfactory user experience and scalability for MAR apps. Two popular techniques that have been commonly used for large-scale NN matching are Locality Sensitive Hashing (LSH) and bag-of-words (BOW) matching.

**LSH: Locality Sensitive Hashing**

LSH [Gionis et al. 1999] is a widely used technique 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 different descriptors in 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 *multi-table* and *multi-probe*, are usually used. The multi-table 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. Multi-table 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. Multi-probe examines both the bucket in which the query descriptor falls and its neighboring buckets. While

multi-probe would result in more matching checks of database descriptors, it actually requires fewer hash tables 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 [Sivic et al. 2003] 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; hence it may greatly degrade the accuracy. In order to enhance the accuracy for BoW matching, several approaches have been proposed to compensate the loss of spatial information. For example, geometric verification [Philbin et al. 2007], which is designed for general image-matching applications, is a popular scheme which verifies local correspondences by checking their homography consistency. Wu et al. presented a bundling feature matching scheme [Wu et al. 2009] for partial-duplicate image detection. In their approach, sets of local features are bundled into groups by MSER [Matas et al. 2002] region detected regions, and robust geometric constraints are then enforced within each group. Spatial pyramid matching [Lazebnik et al. 2006], which considers approximate global geometric correspondences, is another scheme to enforce geometric constraints for more accurate BoW matching. The scheme partitions the image into increasingly finer sub-regions and computes histograms of local features found within each sub-region. To compute the similarity between two images the distance between histogram at each spatial level is weighted and summed together. All these schemes yield more reliable local-region matches by enforcing various geometric constraints. However, these schemes are very computationally expensive, thus when applying them to MAR,

the recognition procedure is conducted on the server side or in the cloud where abundant computing and memory resources are available.

**3) Local Feature based Object Tracking**

A typical flow of local feature-based object tracking is to find corresponding local features on consecutive frames and then estimate the homography transformation between image frames based on local feature matches according to Eq. 1. But different from marker-based tracking and pose estimation, which utilize only four reliable corner matches, local feature-based method often generates a large amount of correspondences which inevitably could include some outliers. Selecting reliable matches from a large correspondence set is challenging and existing solutions often rely on the RANSAC or PROSAC algorithms to solve this problem. The key idea of RANSAC and PROSAC is to iteratively estimate parameters of a transformation model from a set of noisy feature correspondences so that a sufficient number of consensuses can be obtained. We refer readers to [Fischler et al. 1981] and [Chum et al. 2005] for details of the RANSAC and PROSAC algorithms. The quality of local features is essential for the accuracy of local feature matches. A large number of false positive matches resulting from low-quality features could lead to an enormous amount of iterations in the RANSAC and PROSAC procedures, yielding an excessively long runtime.

### 3.2 Sensor-based Recognition and Tracking

Sensor-based method typically leverages the Global Position System (GPS) to identify the location of a mobile device, utilizes a compass (or in combination with other sensors) to determine the direction that the device is heading to. Based on the location and direction of a device, a MAR system could determine which virtual data should be associated with the current scene. After that, the device' motion is tracked based on motion sensors (also known as Inertial

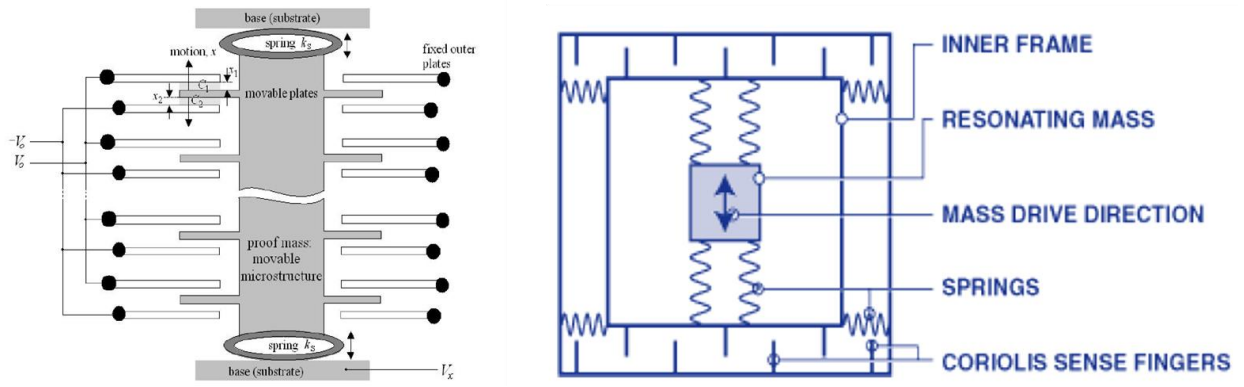


**Fig. 7.** (Left) A typical one-dimensional MEMS capacitive accelerometer and (Right) A vibrating mass gyroscope.

Measurement Unit IMU). Since location recognition using the GPS is straightforward, in the following we mainly focus on common motion sensors used in today's smart mobile devices, their functionalities and tracking algorithms based on these sensors.

### 3.2.1 Sensor-based Object Tracking

With recent advances in Micro-electro-mechanical Systems (MEMS) technology, Inertial Measurement Units (IMUs) are now commonplace in most smart mobile devices. These IMUs are used by mobile apps for tracking the movement of a mobile device and consequently enabling the device to interact with its surrounding. The most common IMUs found in these smart devices today include accelerometers, magnetometers, and gyroscopes. Each of these sensors provides a unique input to the overall tracking system. In the following, we first briefly review their functionalities, hardware and bias of these sensors and then present tacking algorithms based on the sensor data.

**a. Accelerometer**

An accelerometer measures the acceleration forces exerted on a mobile device. The accelerometer reading is a summation of two forces: the gravity force due to the weight of the

device and the acceleration force due to the motion of the device. Today's mobile devices are equipped with a 3-axis accelerometer which measures the forces in the x, y, and z directions with respect to the surface plane of the mobile device.

There are several types of accelerometers and the type used in mobile devices is the capacitive accelerometer. The left figure of Fig. 7 illustrates the structure of a 1-axis MEMS capacitive accelerometer. The accelerometer data is acquired by measuring the force exerted on an object which is able to flex up or down. The amount the device flexes is monitored by a set of fingers which are attached to a moveable inertial mass and flex with the device. As these fingers/plates move, they get closer to, and move further apart from, a set of stationary fingers/plates. The proximity of these fingers/plates can create a change in the measured capacitance between multiple fingers/plates, which can be monitored to measure the displacement of the center inertial mass. This structure can be extended to build a 3-axis accelerometer for measuring the displacement along all 3 axes.

**b. Gyroscope**

A 3-axis gyroscope provides a 3-dimensional vector which measures the rotational (angular) velocity of a device around three axes of the device's coordinate system. The gyroscope was first introduced into smartphones by Apple in iPhone4 in June 2010. The first Android phone in which a 3-axis gyroscope was integrated is Google's Nexus S in December 2010. Gyroscopes work off the principles of the Coriolis force, and are usually implemented within an integrated circuit (IC) using a vibrating mass attached to a set of springs. If the device is rotated about the axis defined by the first set of springs, the inner frame will be pushed away from the axis of rotation, causing a compression in the second set of springs due to the Coriolis acceleration

experienced by the vibrating mass. An example of a MEMS gyroscope is depicted in Fig. 7 (Right).

These types of gyroscopes are relatively cheap to manufacture; however, they are often noisy and could introduce significant errors if their measurements are not modeled properly. On the contrary, many advanced Inertial Navigation Systems (INSs) today have begun using optical gyroscopes instead, which prove to be much more accurate. Because of its broad application and increasing popularity, more advanced gyroscopes are being developed, and integrated into new devices, that can yield more accurate results.

Although latest MEMS gyroscopes have smaller errors than the previous generations, all gyroscopes used in smartphones today still experience a small amount of bias. Given that the gyroscope measures a rate (change over time), the bias itself is a rate. A gyroscope bias can be envisioned as the rotational velocity observed by the device when it is not in motion. This view is somewhat simplified, as a bias can also occur when the device is moving as well. In addition, this bias is sensitive to several factors including the temperature and often randomly varies over time, thus is difficult to compensate for. This bias is often estimated as a random variable by many filtering algorithms.

**c. Magnetometer**

The magnetometer measures the strength of the earth's magnetic field, which is a vector pointing towards the magnetic north of the earth. The magnetometer found in most smart devices is primarily one of two possible types; a Hall Effect magnetometer, or an anisotropic magneto resistive (AMR) magnetometer. The Hall Effect Magnetometers are the most common, and provide a voltage output in response to the measured field strength and can also sense polarity. The AMR magnetometers use a thin strip of a special kind of alloy that changes its resistance

whenever there is a change in the surrounding magnetic field. Using AMR magnetometers usually yield much better accuracy; however, they are more expensive.

One primary source of a magnetometer's error is called the magnetometer bias. This bias is caused by the surrounding environment (external to the magnetometer itself) and can cause a wide range of errors in the magnetometer readings. The bias itself can be separated into one of two types. The first is called hard iron bias. This type of bias is primarily caused by devices which produce a magnetic field. The errors observed by a hard iron bias are constant offsets, usually applied to all axes of the magnetometer equally. This bias is not time or space varying and can be compensated for by simply adjusting the readings of the magnetometer by some constant value. The other type of bias commonly experienced by the magnetometer is called soft iron bias. This type of bias is caused by any distortions in the magnetic field surrounding the magnetometer, thus can have many forms and is difficult to compensate for.

**d. Kalman Filtering for Sensor-based Tracking**

The goal of tracking is to obtain the translation and orientation of a device in the 3-dimensional earth coordinate system. Each of the three sensors alone can provide the orientation of a mobile device. Once we get the orientation information, we can derive the gravity force components along three axes of the mobile device and then subtract the gravity force from the accelerometer data to obtain the motion-induced acceleration force. By double integration of motion-induced acceleration force, we can obtain the translation of the devices in the 3-dimensional earth coordinate system.

However, since each type of sensor data is quite noisy, relying on a single type of sensor cannot achieve an accurate tracking. Many approaches apply a filtering-based method to fuse three types of sensor data for a more reliable and precise tracking result. In the following, we

overview the most standard filtering method - Kalman filtering, which is also the algorithm implemented in Android operating system for estimating smartphone's orientation. For more advanced yet computationally expensive filters such as Unscented Kalman filters or Particle filters, please refer to [Li et al. 2013] and [Cheon et al. 2007] for details.

The Kalman filtering process for orientation estimation can be broken down into two primary steps: the prediction step and the updating step. In the predicting phase, the filter uses the gyroscope measurement to predict the dynamics of the device rotation. The gyroscope measurements are thus integrated directly into the state transition equation and used to provide a predicted state estimate. In particular, the state equation is defined using a seven-element state vector consisting of the quaternion $\bar{q}(t)$ (i.e. four-element orientation vector) and the gyro-bias $b(t)$

$$X(t) = \begin{bmatrix} \bar{q}(t) \\ b(t) \end{bmatrix} \tag{7}$$

We define an error angle vector $\delta\theta$ as a small rotation between the estimated and the true orientation of a device in the earth coordinate system. Similarly, an error bias vector $\delta b$ is defined as small differences between the estimated and the true bias of the device. Accordingly, the error state propagation model is given by

$$\begin{bmatrix} \dot{\delta}\theta \\ \dot{\delta}b \end{bmatrix} = \begin{bmatrix} -[\omega\times] & -I_{3\times3} \\ 0_{3\times3} & -0_{3\times3} \end{bmatrix} \bullet \begin{bmatrix} \delta\theta \\ \delta b \end{bmatrix} + \begin{bmatrix} -I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix} \bullet \begin{bmatrix} n_\omega \\ n_b \end{bmatrix} \tag{8}$$

where $\omega$ is the rotation velocity around three axes. $n_w$ and $n_b$ model the gyroscope noise and bias respectively. In most cases, $n_w$ is assumed to be an independent white Gaussian distribution along each axis of the gyroscope input. Therefore, its expected value is given as $E[n_\omega] = 0_{3\times1}$.

The gyroscope bias model is usually defined as $\dot{b} = n_b$, where $n_b$ is an independent white Gaussian distribution along each axis.

The solution to this differential equation has the closed form solution found in [Trawny et al. 2005] and yields the following state transition matrix:

$$\Phi = \begin{bmatrix} \Theta & \Psi \\ 0_{3\times3} & I_{3\times3} \end{bmatrix} \tag{9}$$

where $\Theta = I_{3\times3} - [\hat{\omega}\times] \bullet \frac{\sin(|\hat{\omega}|\Delta t)}{|\hat{\omega}|} + [\hat{\omega}\times]^2 \bullet \frac{(1-\cos(|\hat{\omega}|\Delta t))}{|\hat{\omega}|}$

and $\Psi = [\hat{\omega}\times] \bullet \frac{(1-\cos(|\hat{\omega}|\Delta t))}{|\hat{\omega}|} - [\hat{\omega}\times]^2 \bullet \frac{(|\hat{\omega}|\Delta t - \sin(|\hat{\omega}|\Delta t))}{|\hat{\omega}|^3} - I_{3\times3}\Delta t$

Note that in the state transition matrix $\hat{\omega} = \omega - b$ is treated as a system input which has already been bias corrected.

In the updating phase, the filter combines the previously estimated state with the recorded accelerometer and magnetometer measurements come directly from sampling the IMUs to revise the orientation estimation. Each recorded measurement complies with a model which describes its relationship with the estimated states and noise errors of some measurements. Specifically, the measurement model is of the form:

$$z = R_E^B(q) \bullet z^0 + n_z \tag{10}$$

where $E[n_z] = 0$, $E[n_z n_z^T] = R$, and $R_E^B(q)$ is a rotation matrix from the earth coordinate system to the predicted device coordinate system. The rotation matrix is obtained using the propagated quaternion from the process model. $z^0$ is a unit vector representation of north in the earth coordinate system. The measurement residual is defined as $r = \hat{z} - z$, where $\hat{z}$ is the input measurement. Since this residual represents the error between the measurement vector and the

predicted vector, it is a close approximation of the error angle vector $\delta\theta$. This approximation is derived in [Trawny et al. 2005] and defined as

$$r \approx \begin{bmatrix} \left[ R_E^B(q) \bullet z^0 \times \right] & 0 \end{bmatrix} \begin{bmatrix} \delta\theta \\ \delta b \end{bmatrix} + n_z \tag{11}$$

which gives a result for the measurement model H:

$$H = \begin{bmatrix} \left[ R_E^B(q) \bullet z^0 \times \right] & 0 \end{bmatrix} \tag{12}$$

After the measurement update, the residual obtained in the measurement model is used to update the quaternion which in turn is used as the filter result.

### 3.3 Hybrid-based Recognition and Tracking

At present the processing power and memory capacity of mobile devices are still too limited for scalable MAR apps solely relying on sophisticated visual recognition and tracking methods. On the other hand, the built-in sensors (e.g. accelerometer, gyroscope, magnetometer and GPS) usually lack sufficient accuracy, thus cannot provide satisfactory performance for recognition and tracking tasks. Several studies proposed to combine these vision-based and sensor-based methods. For instance, GPS positioning alone is insufficient for augmented reality apps, but it can be combined with a visual tracking method to achieve a desired level of accuracy [Reitmayr et al. 2007]. The basic idea behind these methods is to use GPS to identify the position (i.e. the location on earth) and this information is used to initialize the visual tracking system, which in turn gives the user's local pose and the view direction. In [Naimark et al. 2002], 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.

Another promising direction is to combine vision information with motion sensor data (i.e. gyroscope, accelerometer and magnetometer) to provide a more accurate and efficient object tracking. 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 6 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 multi-model information.

## 4. Software Development Toolkits

OpenCV is one of the most popular software development libraries for computer vision tasks. A mobile version of OpenCV has been released for running on mobile platforms [OpenCV for Android]. Other mature libraries such as Eigen [Eigen main page] or LAPACK for linear algebra [LAPACK – Linear Algebra PACKage] are also become available for mobile platforms even though the support and the optimization level are still limited.

Qualcomm has released a mobile-optimized computer-vision library, named FastCV [FastCV main page], which includes the most frequently-used vision processing functions and can be used for camera-based mobile apps. The CV functions offered by FastCV include gesture recognition, text recognition and tracking, and face detection, tracking and recognition. FastCV can run on most ARM-based processors, but is particularly tuned for Qualcomm's Snapdragon processor (S2 and above) and utilizes hardware acceleration to speed up some of the most compute-intensive vision functions.

Built on top of FastCV, Qualcomm further offers an MAR software development kit (SDK), named Vuforia™ [Vuforia main page]. Vuforia offers software functions for app developers that can recognize and maintain a variety of 2D and 3D visual targets, frame markers, text and user interactions (e.g. interactions with a virtual button). In addition, it provides APIs to easily render 3D graphics or video playback on top of the real scene. To manage visual targets, Vuforia provides two ways to store target databases: on a mobile device or on the cloud. Device databases do not require network connectivity for the recognition, and thus can avoid the overhead for data transfer and are free to use in mobile apps. However, due to the limited storage space and computing power of mobile devices, device databases can only store a limited number of targets, so far the max of targets can be stored in a device database is 100. Cloud databases are managed using either the Target Manager UI provided by Qualcomm or the Vuforia Web Service API. They enable you to host over 1 million targets on the cloud. The Vuforia cloud recognition service is an enterprise class solution with various pricing plans determined by the total number of image recognitions per month your app. Generally speaking, Vuforia development infrastructure facilitates, and significantly simplifies, the development of MAR apps.

## 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 augmented reality into the mainstream of mobile apps. Following the widespread popularity of a handful of killer MAR applications already commercially available, it is believed that MAR will expand exponentially in the next few years. The advent of MAR will have a profound and lasting impact on the way people use their smartphones and tablets. These emerging MAR 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 towards truly ubiquitous computing and a technologically converged paradigm.

The scalability, accuracy and efficiency of the underlying techniques (i.e. object recognition and tracking) are key factors influencing user experience of MAR 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 recognition and tracking 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 and device tracking 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.