Signal Processing: *Image Communication* 28 (2013) 334–344

Contents lists available at SciVerse ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

# Rotation-invariant fast features for large-scale recognition and real-time tracking

Gabriel Takacs [a,b,*], Vijay Chandrasekhar [b], Sam Tsai [b], David Chen [b], Radek Grzeszczuk [a], Bernd Girod [b]

[a] *Microsoft, United States*
[b] *Stanford University, United States*

ARTICLE INFO

Available online 22 December 2012

*Keywords:*
Feature descriptors
Keypoints
Tracking
Visual search

ABSTRACT

We present an end-to-end feature description pipeline which uses a novel interest point detector and rotation-invariant fast feature (RIFF) descriptors. The proposed RIFF algorithm is 15 × faster than SURF [1] while producing large-scale retrieval results that are comparable to SIFT [2]. Such high-speed features benefit a range of applications from mobile augmented reality (MAR) to web-scale image retrieval and analysis. In particular, RIFF enables unified tracking and recognition for MAR.



## 1. Introduction

Image content description is used in a wide range of applications, including hand-held product recognition, set-top-box video content detection, web-scale image search, and augmented reality. Many such applications are constrained by the computational power of their platforms. Even in unconstrained cases, such as web-scale image search, processing millions of images can lead to a computational bottleneck. Therefore, algorithms with low computational complexity are always desirable.

Augmented reality applications are further constrained because the mobile device's resources must be shared between camera pose tracking and image content recognition. These two tasks are typically decoupled from each other. However, if we can extract robust local features in real-time, then we can track video content at very little additional cost. Current technologies that are fast enough for real-time tracking do not perform well at recognition from large-scale databases. Conversely, algorithms which excel at recognition are not fast enough for real-time tracking on mobile devices. We therefore propose a new end-to-end feature extraction algorithm which is state-of-the-art in both speed and accuracy.

### 1.1. Contributions

We propose a local image feature algorithm that is an order of magnitude faster than current methods and achieves state-of-the-art image retrieval. We do so by improving the RIFF descriptor [3] and proposing a new interest point detector that tightly couples with the descriptor. We demonstrate our algorithm on a difficult, real-world dataset of one million images. The high speed of our algorithm is possible for several key reasons.

- *Box filtering*: the filter response of our interest point detector is extremely low-complexity.
- *Low scale-space overhead*: we use a scale-space representation that computes very few filter responses, while capturing the full space.
- *Filter re-use*: an image scale-space is required to compute descriptors. While computing a filter-response scale-space, we obtain an image scale-space at no additional cost.
- *No pixel interpolation*: at no time are pixels or gradients interpolated or rotated. All computation is on the original image raster.

* Corresponding author at: Microsoft, United States.
*E-mail address:* gtakacs@alumni.stanford.edu (G. Takacs).

0923-5965/$ - see front matter © 2013 Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.image.2012.11.004

- *Radial gradients*: unlike $(x,y)$-gradients, radial gradients allow us to place gradients into oriented spatial bins without rotation.

### 1.2. Prior work

Currently, the predominant method of image content description is local features. Early algorithms, such as SIFT [2], perform very well at content recognition, but are computationally expensive. Since SIFT's introduction there have been many algorithms that aim to reduce the computational load of image feature extraction. Such algorithms include SURF [1], Ferns [4], FAST [5], CenSurE [6], RIFF [3] and BRIEF [7]. In addition to algorithmic enhancements, people have made GPU implementations to allow for accurate, high-speed feature extraction [8,9].

The SURF algorithm is an end-to-end feature extraction pipeline which performs well for a wide range of image matching tasks. In contrast, many other fast algorithms are only partial solutions for feature extraction or only work well in constrained applications. For example, the FAST corner detector is extremely low complexity, but does not provide scale or orientation, and is only a partial solution because feature descriptors must still be computed. The CenSurE interest-point detector is not quite as efficient as the FAST detector, but has the added benefit of localizing features in scale-space. However, CenSurE still lacks orientation, thereby producing only upright features.

Calonder et al. [7] have proposed the BRIEF descriptor which can be computed and matched very efficiently when paired with the FAST detector. Unfortunately, BRIEF descriptors also lack scale and orientation. Recently, others have added orientation invariance to the BRIEF descriptor, resulting in ORB [10], and BRISK [11]. These algorithms cannot be easily used in many retrieval algorithms because they must be compared with a Hamming distance, which is not easily adapted to accelerated search structures such as vocabulary trees or approximate nearest neighbors (ANN). We have previously proposed the RIFF descriptor [3] which is rotation invariant, and can be efficiently computed. However, the previous RIFF is only a partial solution, and does not solve the interest point problem. We propose extensions to RIFF that make it a full-fledged feature descriptor pipeline. The new pipeline is extremely fast, and can be used with standard search algorithms.

In this paper, we first introduce our interest point detector in Section 2. Then, in Section 3 we show how to compute descriptors at interest points. Section 4 demonstrates RIFF's recognition performance, and Section 5 covers real-time tracking. Finally we conclude in Section 6.

## 2. Interest point detection

To perform well on large-scale retrieval tasks, we require interest points that can be localized in both location and scale. Although FAST corners can be detected at different scales, they are inherently insensitive to scale changes. Also, replicating them at many scales can create an excessively large database and unwanted redundancy. Conversely, blob detectors such as Laplacian of Gaussian (LoG), Difference of Gaussians (DoG), Determinant of Hessian (DoH), and Difference of Boxes (DoB) are all sensitive to scale variation and can thus be localized in scale space. The DoB filter is the simplest of these options, and is therefore our choice for a fast detector.

We adopt the filters proposed by Agrawal et al. [6]. The filter response is the simple weighted difference of two box filters that are centered on the same point but have different scales. For a scale parameter, $s$, the inner box has width $2s+1$ and the outer box is roughly twice the size with width $4s+1$. The filter response is thus given by

$$(2s+1)^{-2}\Sigma_{\text{in}}-(4s+1)^{-2}\Sigma_{\text{out}}, \tag{1}$$

where $\Sigma$ is a sum of pixel values within the box. We compute these sums efficiently by using an integral image. Given the filter response, we find local maxima and minima in scale-space whose absolute values are above a threshold. More details of extrema detection are given later in Section 2.4. For each of these extrema, we then eliminate edge responses by thresholding the Harris corner score within a radius of $5s$ pixels. The surviving interest points are sorted by their absolute responses.

### 2.1. Image scale-space

A key element of our proposed algorithm is the reuse of filter responses for subsequent descriptor computation. To compute a descriptor from a given location in scale-space, anti-aliased pixels values must be computed at the correct scale. Instead of recomputing these values with the integral image, or via a mipmap with trilinear interpolation, we simply reuse our DoB filter results. To do so, we store the inner box-filter values, $(2s+1)^{-2}\Sigma_{\text{in}}$, in a separate image scale-space memory buffer.

### 2.2. Scale-space subsampling

Agrawal et al. [6] propose computing a full, dense scale-space. However, the computational complexity of doing so is linear in the number of scales. For speed, we propose using a pyramid scale space, where each scale is downsampled by a factor that matches the filter scale. In our scheme, the first scale is computed on the full resolution, and the subsequent scales are downsampled by factors of $2\times$, $3\times$, $4\times$, etc. As shown in Section 2.3, this dramatically reduces the complexity of interest point detection.

To prevent aliasing when down-sampling, we must low-pass filter the image. For efficiency, we re-use the inner box filter value from the DoB computation. Each pixel at scale $s$ is thus filtered by a rectangular filter of width $2s+1$. To show that this filter is appropriate for anti-aliasing, we consider the 1D impulse response

$$h[k]=\begin{cases}(2s+1)^{-1}, & |k|\le s\\ 0 & \text{otherwise}\end{cases}. \tag{2}$$

The associated frequency response, $H(\omega)$, is given by

$$H(\omega) = \frac{\sin\left[\omega\left(s+\frac{1}{2}\right)\right]}{(2s+1)\sin(\omega/2)}. \quad (3)$$

The first zero crossing of which falls at $\omega_0 = 2\pi/(2s+1)$. To preventing aliasing while down-sampling by a factor of $s$, we must suppress frequencies larger than the Nyquist rate of $\omega_c = \pi/s$. Because $\omega_0 < \omega_c$, the main lobe of the frequency response is contained within the Nyquist rate, and aliased frequencies are suppressed by at least 10 dB.

### 2.3. Complexity analysis

We compare the computational complexity of the proposed scale-space with that of SURF. To do so, we consider the number of samples in each scale space and the effort required to compute each sample. Let $C$ be the number of samples in the scale space, $M$ be the number of octaves used by SURF, and $N$ be the number of scales used by RIFF.

Typically, SURF computes four scales per octave. Except for the first octave, two scales from each octave use the same filter size as scales in the previous octave. Therefore, an efficient implementation will require computing the following number of samples

$$C_{\mathrm{SURF}}(M) = 2wh\left(1+\sum_{i=0}^{M-1} 4^{-i}\right). \quad (4)$$

For the typical value of $M=3$ octaves, there is a 362.5% overhead to compute the SURF scale-space

$$C_{\mathrm{RIFF}}(N) = wh\sum_{i=1}^{N} i^{-2}. \quad (5)$$

This is a power sum which can be given by the Riemann–Zeta function as $N$ goes to infinity

$$\lim_{N\to\infty} C_{\mathrm{RIFF}}(N) = \frac{\pi^2}{6}wh \approx 1.645wh. \quad (6)$$

Therefore, RIFF has at most 64.5% scale-space overhead compared to the base image. Compared to the full scale-space, as proposed by CenSurE [6], there is a speedup of approximately $6(N-1)/\pi^2+1$. For $N=8$ scales, this equates to a 5.25 × speed-up. This low overhead is key to enabling real-time performance.

Not only does RIFF compute fewer samples, but each is significantly simpler to compute. In Fig. 1, we compare the number of operations per pixel for RIFF and SURF. As described in [1], SURF uses an approximate determinant of Hessian, $|\mathcal{H}| = D_{xx}D_{yy} + (\kappa D_{xy})^2$. This requires a total of eight box filters; two for each of $D_{xx}$ and $D_{yy}$, and four for $D_{xy}$. Each box filter requires three additions, and four memory accesses. Each of $D_{xx}$ and $D_{yy}$ also require a multiplication. Assembling the filters into $|\mathcal{H}|$ requires another three multiplications, one addition, and a memory access to store the result.

In contrast, RIFF only uses two box filters, each requiring three additions, multiplication by a weighting term, and four memory accesses. Assembling the filters into the DoB response requires one more addition and two memory accesses to store the filter and image scales-space results. Overall, RIFF computes one third as many filter responses and requires one-third as many operations per response.

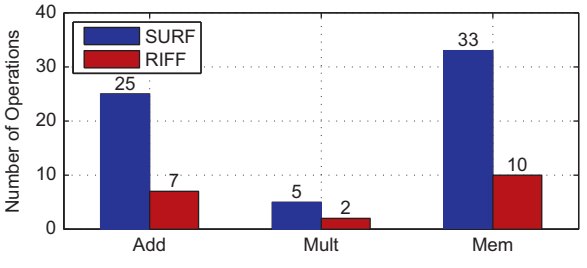


**Fig. 1.** Comparison of the number of additions, multiplications, and memory accesses required per pixel to compute the filter response. RIFF requires about a third as many operations as SURF.

### 2.4. Scale-space extrema

As with SIFT, SURF and CenSurE, we use local extrema to find repeatable points in scale space. However, unlike these other detectors, adjacent layers of our scale space do not have the same resolution. Because of this, a simple 27-pixel 3D neighborhood is not possible, and we therefore propose a method to compensate for the resolution change.

We store the scale-space in a full resolution stack of images, but only compute pixel values with a sampling stride equal to the scale parameter. Fig. 2 illustrates this arrangement. To find the neighbors of a pixel at position $(x,y,s)$, we first consider the eight neighbors within the same scale, given by $\{(x\pm s,y\pm s,s), (x,y\pm s,s), (x\pm s,y,s)\}$. We then find the nearest existing pixels in the scales above and below, $(x_+,y_+,s+1)$ and $(x_-,y_-,s-1)$, where

$$x_- = (s-1)\lfloor x/(s-1)+0.5\rfloor \quad (7)$$

$$x_+ = (s+1)\lfloor x/(s+1)+0.5\rfloor \quad (8)$$

and similarly for $y_-$ and $y_+$. Given these central pixels above and below, we find their eight neighbors as before. We call this the *inter*-scale detection scheme. For speed, we also propose a much simpler *intra*-scale scheme wherein we consider a point to be a local extrema if it is maximal or minimal relative to its eight neighbors on the same scale. While the *inter* scheme provides full scale-space localization, the *intra* scheme describes points at multiple salient scales, and is faster. Additionally, it has been suggested by

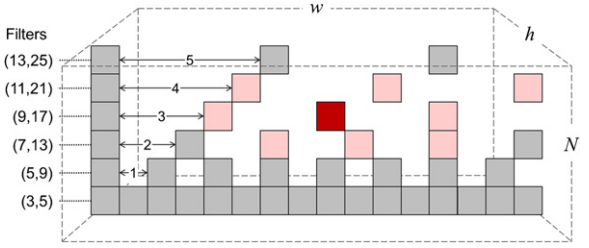


**Fig. 2.** Example slice through the proposed sub-sampled scale-space. There are $N$ scales formed from the original $w \times h$ pixel image. We subsample pixels according to the scale, but store them relative to the full scale. The pink pixels are the neighbors of the red pixel that we use for inter-scale local extrema detection. Also shown are the (inner, outer) filter sizes for each scale.

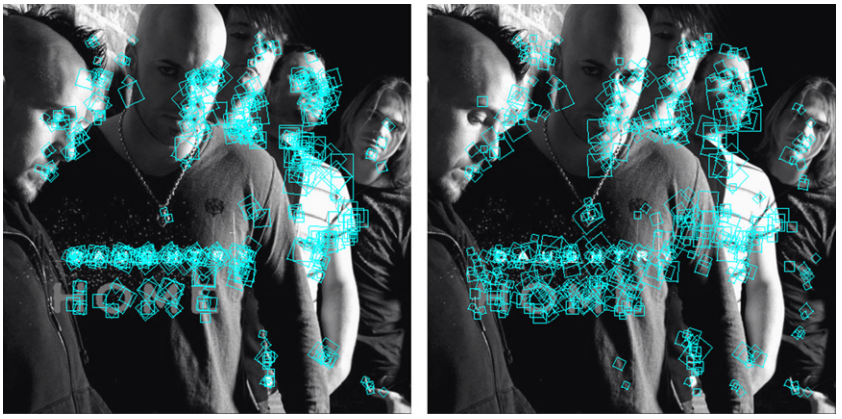

**Fig. 3.** Example interest point detection for *intra*-scale (*left*) and *inter*-scale (*right*) modes. There are 500 features in each image.

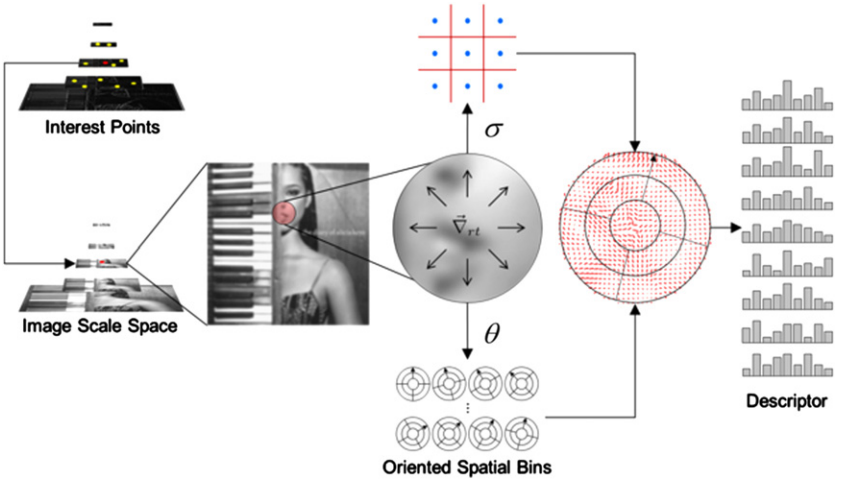


**Fig. 4.** RIFF descriptor pipeline. Given an interest point, we determine an orientation using $(x,y)$-gradients. We then place quantized radial gradients in the appropriate spatial bin.

Wu et al. [12] that multi-scale description can be beneficial. Fig. 3 shows an example of interest points from both schemes. Note that the interest points are oriented during subsequent descriptor computation.

## 3. Descriptor computation

Given interest point locations and an image scale-space, we now compute feature descriptors. We have improved on the RIFF descriptor proposed in [3] in two significant ways. First, we have coupled the descriptor with the interest point scale-space. Second, we have added orientation assignment and improved the spatial binning scheme. As illustrated in Fig. 4, the proposed RIFF descriptor is computed with the following steps.

*Patch extraction*: we compute a descriptor on a circular patch of diameter $25s$, centered on a point $(x,y,s)$. The pixels in the patch are sampled with a stride of $s$ pixels from the image scale-space that was precomputed during interest point detection.

*Orientation assignment*: we compute $(x,y)$-gradients for each pixel in the patch, using a $[-1,0,1]$ centered difference filter. We then form a 72-bin, magnitude-weighted histogram of the gradient orientations. For speed, we use a look-up table to convert pixel differences into angle and magnitude. With 8-bit pixel values, there are only $512 \times 512$ possible gradient values. For robustness, we apply a simple [1,1,1] low-pass filter to the histogram. Finally, we robustly find the dominant direction. If the value of the second most dominant angle bin is within 90% of the dominant bin's value, then we choose the bin that is to the right of the angle that bisects the two bins. Note that the patch is never actually rotated, we need only find the angle.

*Radial gradient quantization*: we first compute the standard deviation, $\sigma$, of the patch. Then, we compute the approximate radial gradient transform (ARGT), as described in [3]. The ARGT must incorporate proper baseline normalization because diagonal pixel neighbors are farther than horizontal or vertical neighbors. Let $b$ be the distance between two pixels in the ARGT, and $q$ be the desired gradient quantizer step-size. We combine the quantizer parameter, intensity and baseline normalization by multiplying pixel differences by $(bq\sigma)^{-1}$. Finally, we obtain

quantized radial gradients by rounding to each component to $\{-1,0,1\}$, yielding one of the nine possible gradients.

*Spatial binning*: given the descriptor orientation, $\theta$, we select a spatial layout that is rotated by $-\theta$. For speed, the spatial bins are precomputed for each possible orientation. We use a layout with a central bin and two outer rings of four bins each, for a total of nine bins, as shown in Fig. 4. In each spatial bin we form a histogram of quantized gradients which we then normalize to sum to one. The resulting descriptor is 81-dimensional. Note that the radial gradients are already rotation invariant, thus by placing them in the proper spatial bin, the entire descriptor is rotation invariant.

### 3.1. Orientation invariance

We use pairwise image matching to demonstrate that the proposed RIFF pipeline is invariant to image rotation. We perform pairwise matching on 100 pairs of images of CDs from the MPEG dataset described in Section 4.1. We rotate one of the images in 5° increments and record the number of geometrically verified feature matches. To ensure that there are not edge effects, we crop the images to circular regions and pad the borders with 100 pixels on all sides. In Fig. 5, we show these results for RIFF with and without approximate radial gradients, as well as for SURF. We note an oscillation in the SURF results with a period of 90° which is due to the anisotropy of box filters. There is a similar oscillation in the exact-RGT RIFF from the DoB filter. Using the approximate RGT introduces a higher frequency oscillation with a period of 45° which is caused by the 8-direction RGT approximation. However, note that this approximation generally improves matching performance.

### 3.2. Execution profile

We compare the execution profiles of RIFF and SURF on an Intel Xeon 2.8 GHz processor. To do so, we use the same $512 \times 512$ image as input to both algorithms, and we tune the detector threshold so that each yields 500 features. For this experiment, we use the original ETH implementation of SURF [1]. We report the average extraction time from 10 runs, as measured by the SURF code itself. For RIFF, we average over 100 runs to reduce the effect of timing jitter on such short run-times. RIFF is $15\times$ faster than SURF, requiring only 15 ms compared to SURF's 250 ms. We also profile the relative proportions of the major components of each algorithm using the Google profiler [13]. Fig. 6 shows this comparison.

### 3.3. Feature compression

Because the RIFF descriptor is composed of normalized histograms, we can easily apply the compression techniques proposed by Chandrasekhar et al. [14]. We jointly quantize and compress an entire histogram such that the $L_1$-norm is preserved. In particular, we use the *type coding* technique, with a quantization parameter equal to the number of gradient bins. This yields a compressed-RIFF (C-RIFF) descriptor that can be stored in 135 bits using fixed length codes, or $\sim$100 bits with variable length

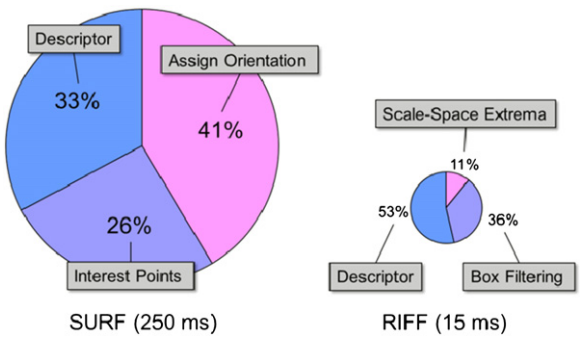


**Fig. 6.** Execution profile for feature extraction with RIFF and SURF from a $512 \times 512$ pixel image using an Intel Xeon 2.8 GHz. The depicted areas are to scale.

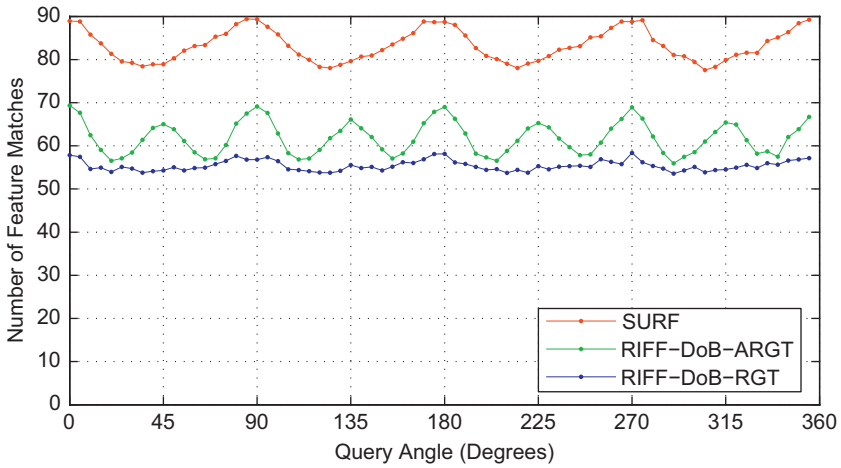


**Fig. 5.** Number of pairwise feature matches at different query orientations. SURF gives the most matches, and oscillates with a 90° period because of its box filtering. RIFF oscillates with a 45° period because of its 8-direction approximate RGT. The oscillation disappears when we use the exact RGT.

codes. This is 6.5× less than an 8-bit per dimension, uncompressed descriptor. The performance of this descriptor is shown later in Section 4.2.

## 4. Image recognition

The primary goal of feature extraction is image recognition by matching against a set of database images. In this section, we demonstrate RIFF's recognition performance on a challenging, large-scale, real-world dataset. We also compare RIFF with other state-of-the-art algorithms.

### 4.1. MPEG dataset

The motion picture experts group (MPEG) is undertaking an effort to standardize compact descriptors for visual search (CDVS) [15]. To do so, they have proposed a rigorous performance evaluation which consists of pairwise image matching and large-scale retrieval of five different classes of image content. The classes are text/graphics, museum paintings, video frames, common objects, and buildings/landmarks. There are 2500, 455, 500, 14,935, and 10,200, images in each of the respective classes. In the first three classes, there exists a clean reference image. Example images from this dataset are shown in Fig. 7.

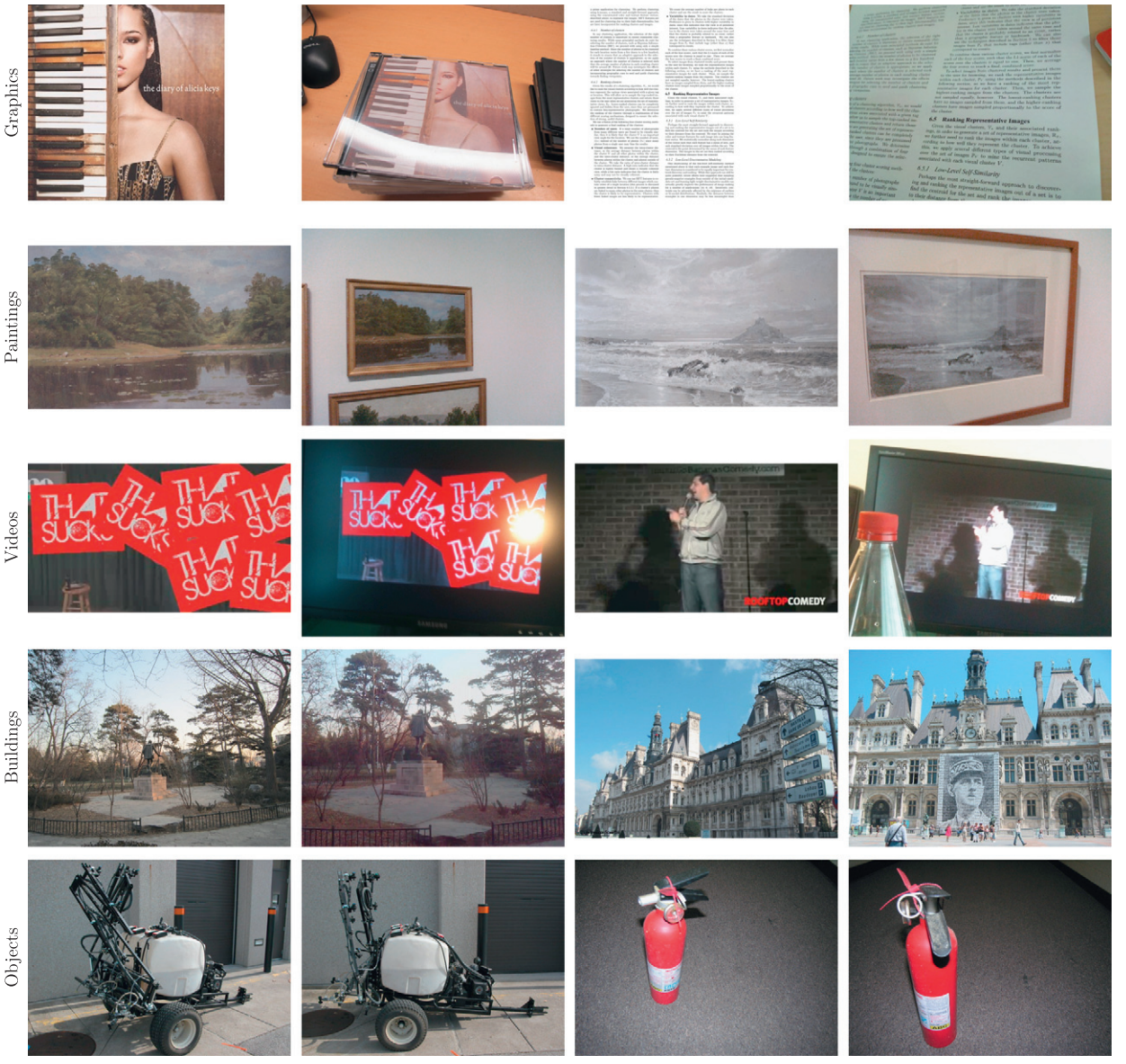


**Fig. 7.** Example matching image pairs from the MPEG CDVS dataset. Each row shows one of the five image classes, mixed text and graphics, museum paintings, video frames, common objects, and buildings/landmarks.

Using this set of images, MPEG defines a pairwise matching experiment with 16 k matching pairs, and 172 k non-matching pairs of images. As required, we report the true positive rate for each class, subject to the constraint that the overall false positive rate is less than 1%. For retrieval, MPEG defines an 18 thousand image database and 11 thousand query images. None of the queries exist in the database. An additional 1 million distracter images from Flickr are added to the database images. Using these images, we report the precision of the first match, or precision at one (PAO), as well as the mean average precision (MAP) for each query class. The CDVS standard is concerned with recognition performance as a function of query size. However, to demonstrate scalability, we report recognition as a function of the number of database images.

### 4.2. Recognition results

We use a state-of-the-art image retrieval system that uses a vocabulary tree with and weak geometric re-ranking, followed pairwise matching and strong geometric consistency checking. Our system is similar to

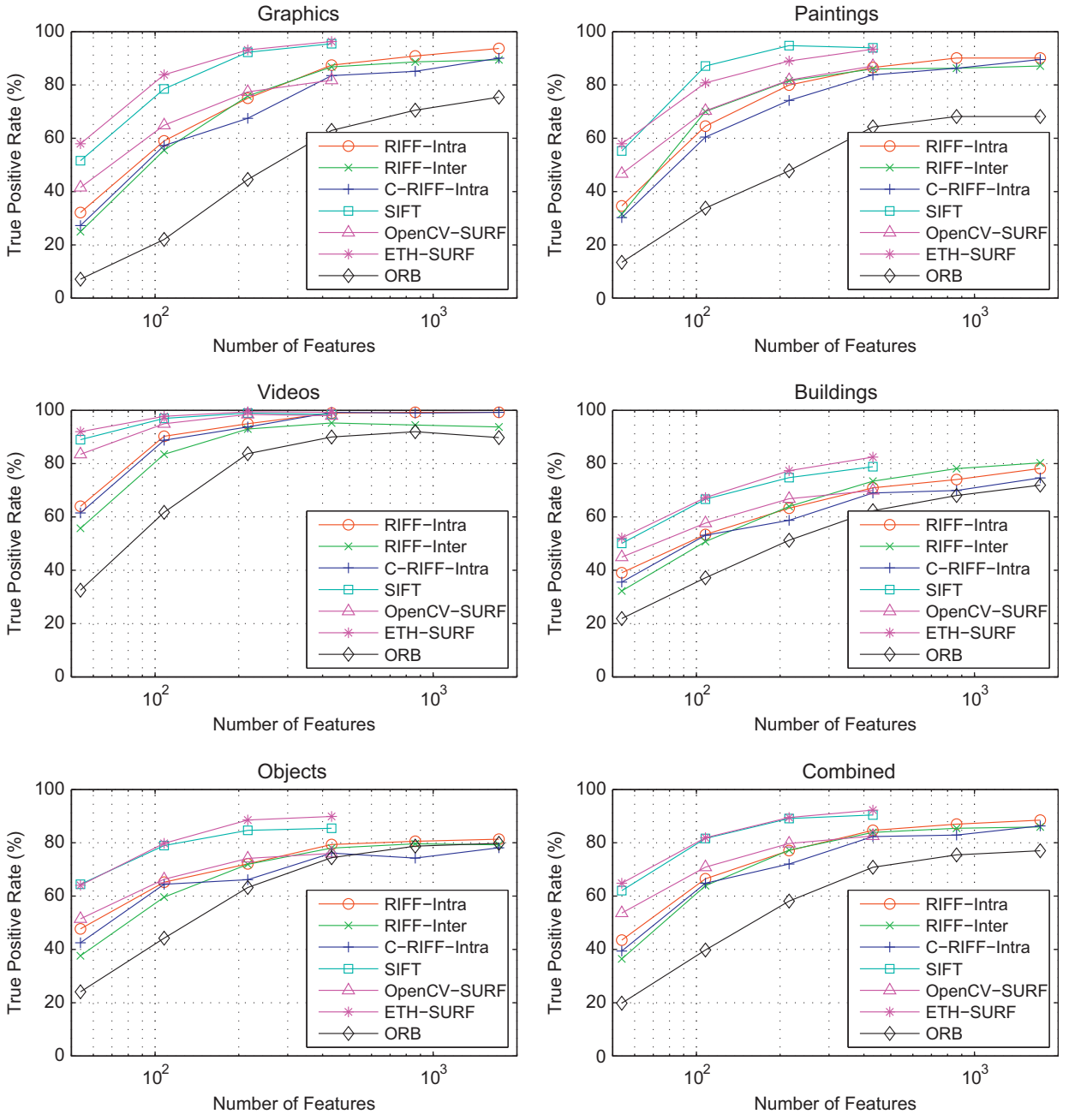


**Fig. 8.** True positive rate (TPR) for pairwise matching subject to a false positive constraint of 1%. We show results for three variants of the proposed RIFF algorithm (Intra, Inter, Compressed Intra), as well as SIFT and two SURF implementations.

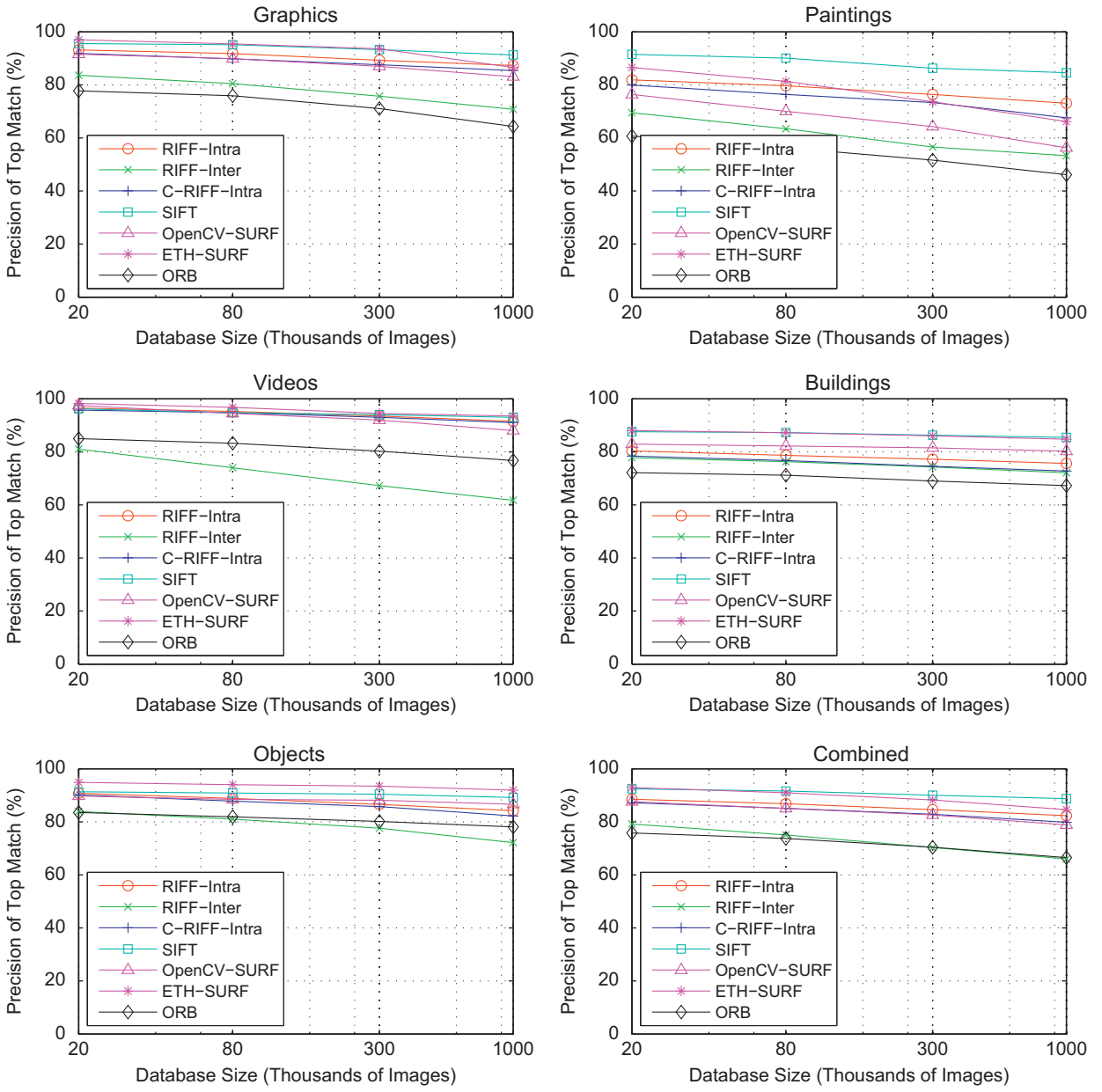


**Fig. 9.** Precision at one (PAO) measure of retrieval performance. We show results for three variants of the proposed RIFF algorithm (Intra, Inter, Compressed Intra), as well as SIFT and two SURF implementations.

the ones used by Girod et al. [16] and Philbin et al. [17]. For pairwise matching we use a ratio test as proposed by Lowe [2], followed by an affine model RANSAC. To ensure the best results, given the false positive constraint, we jointly vary the ratio-test threshold over the values [0.8, 0.85, 0.9], and the RANSAC inlier threshold over the values [4, 8, 12]. However, we use the same values across all image classes.

Figs. 8–10 show the results of these experiments for SIFT [18], OpenCV SURF [19], ETH SURF [1], ORB [10], RIFF-Intra, RIFF-Inter and Compressed-RIFF. We observe that RIFF-Intra significantly outperforms RIFF-Inter in all categories. This is likely because the feature budget is spent on the most salient features. Compression only causes a small drop in RIFF's performance. Given enough descriptors with pairwise matching, RIFF-Intra performs on par with SIFT and SURF. For retrieval, RIFF-Intra performs on par with SURF when the database has 1 million images. ORB performs significantly worse than the other algorithms.

## 5. Tracking

With the proposed RIFF pipeline, we can perform both video tracking and content recognition by extracting features at every frame and using the tracking algorithm proposed by

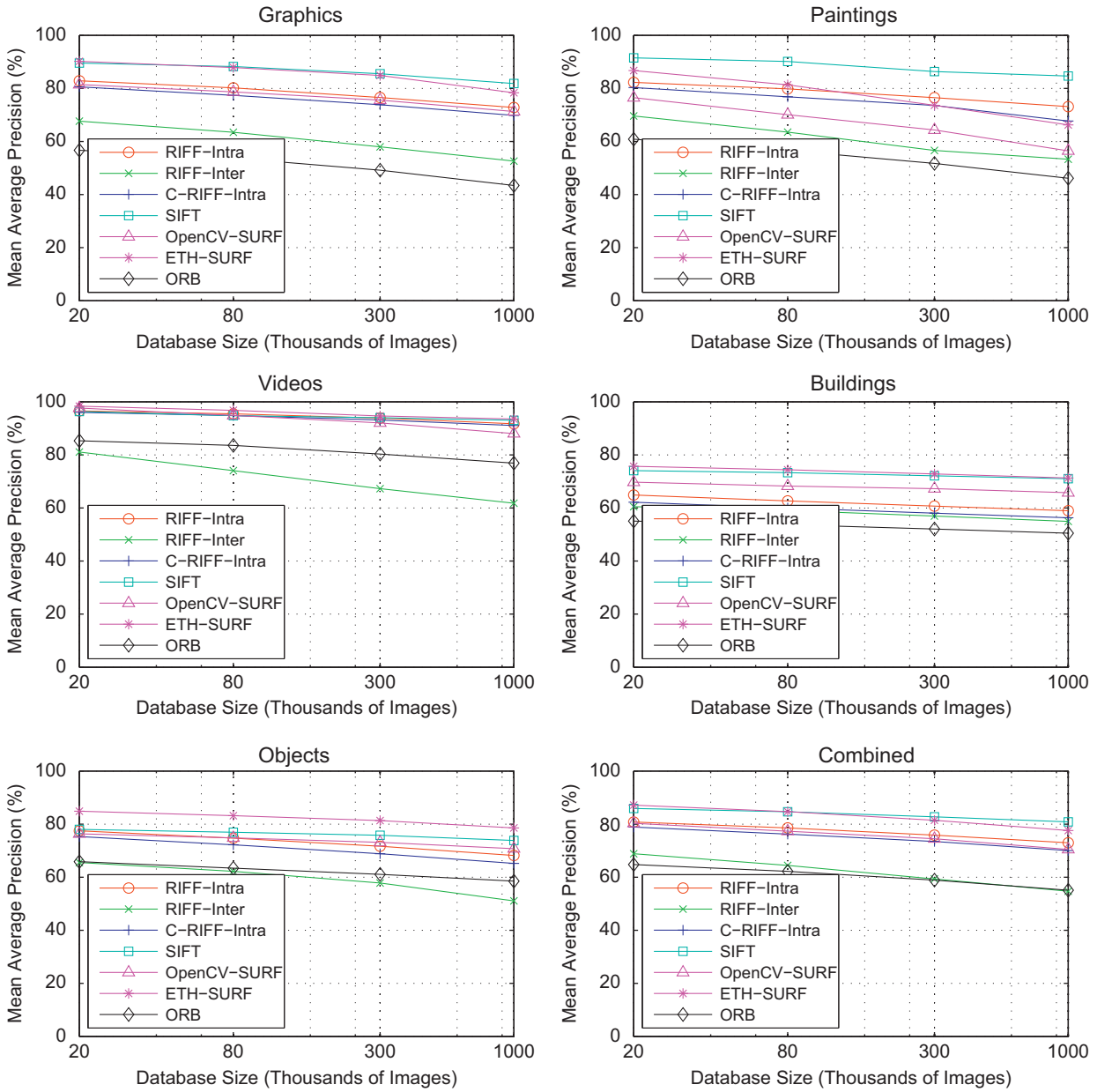


**Fig. 10.** Mean average precision (MAP) measure of retrieval performance. We show results for three variants of the proposed RIFF algorithm (Intra, Inter, Compressed Intra), as well as SIFT and two SURF implementations.

Takacs et al. [3]. For MAR, we must extract features in real-time on a mobile device. To meet the computational constraints of such a platform, we use a $320 \times 240$-pixel video and extract 100 DoB features per frame.

### 5.1. Palindromic tracking error

We propose a measure of tracking drift that can be used on any video sequence without the need for expensive or tedious ground-truth measurements. Note that if a sequence is played forward in time, and then played in reverse, the resulting video will be symmetric in time. We call such a sequence *palindromic*. The tracking results of a palidromic sequence must also be time-symmetric, and any deviation from this symmetry represents a drift in the tracker.

Because different algorithms track different points in the video, we have selected video sequences that are well approximated by global affine motion models. Assume that we have a frame-to-frame model, $A_i$, which moves a point, $x_{i-1}$, in frame $i-1$ to a point, $x_i$ in frame $i$, such that $x_i = A_i x_{i-1}$. We also consider the cumulative model, $C_i$, for all movement between the first and $i$th frames, such that $C_i = \prod_{j=1}^{i} A_j$ using a left matrix multiply. Time symmetry of a tracked point can then be described by $C_i x = C_{N+1-i} x$,

where $N$ is the length of the palindromic video sequence. The location error for point $x_i$ is thus given by

$$\varepsilon(x_i) = (C_i - C_{N+1-i})x_i = \begin{bmatrix} e_{11}^{(i)} & e_{12}^{(i)} & e_{13}^{(i)} \\ e_{21}^{(i)} & e_{22}^{(i)} & e_{23}^{(i)} \\ 0 & 0 & 0 \end{bmatrix} x_i \tag{9}$$

We then integrate over the set of all point locations, $R$, in the frame to obtain a total frame error, $\xi$. To avoid numerically unstable values, we normalize image coordinates by the width of the image, $w$

$$\xi^2 = \int_{x \in R} \|\varepsilon(x)\|^2 \, dx \tag{10}$$

$$\begin{aligned} \xi^2 = {} & \frac{1}{3}(e_{12}^2 + e_{22}^2)r^3 + \left(\frac{1}{2}e_{11}e_{12} + \frac{1}{2}e_{21}e_{22}\right)r^2 \\ & + (e_{12}\bar{e}_{13} + e_{22}\bar{e}_{23})r^2 + \left(\frac{1}{3}e_{11}^2 + \frac{1}{3}e_{21}^2 + \right)r \\ & + (\bar{e}_{13}^2 + \bar{e}_{23}^2 + e_{11}\bar{e}_{13} + e_{21}\bar{e}_{23})r \end{aligned} \tag{11}$$

where $\bar{e} = e/w$ and $r = h/w$. Given $\xi$ for each frame, we compute the mean palindromic error for the sequence, $\zeta = (1/N)\sum_{i=1}^{N} \xi_i$. This gives the tracking drift as a percent of the frame width, averaged over all points in all frames.

### 5.2. Experimental results

We compare our tracking results to two other algorithms, the Kanade Lucas Tomasi (KLT) and normalized correlation coefficient (NCC) trackers. We use the KLT implementation from Birchfield [20], and our own speed-optimized NCC tracker. In Fig. 11 we plot the palindromic drift versus the tracker's speed. We show results for 21 video sequences captured with a Nokia N900 camera-phone. Each sequence is captured as uncompressed $320 \times 240$ grayscale images at 15 frames per second. We measure tracking speed on an Intel Xeon 2.8 GHz and a Nokia N9 (1.0 GHz ARM Cortex A8) with all frames pre-loaded into memory.

As shown in Fig. 11, all three trackers operate in real-time ($>15$ fps) on the PC. However, on a $10\times$ slower mobile device, only NCC and RIFF are viable candidates. The RIFF tracker runs at $31 \pm 4.7$ fps, which is twice as fast as NCC. This speed provides extra time to perform the other tasks required for MAR, such as processing video

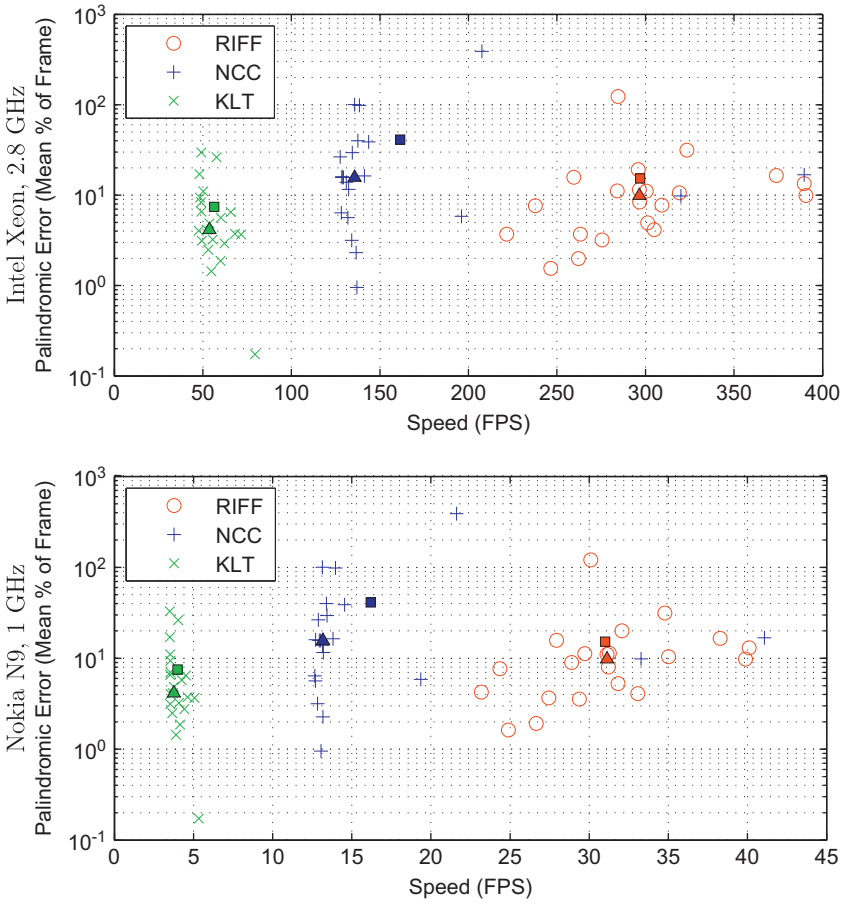


**Fig. 11.** Palindromic tracking error versus speed for RIFF, NCC, and KLT trackers on a PC (*top*) and phone (*bottom*). Each point represents one of 21 QVGA video sequences. We also plot the mean (□) and the median (△) of each algorithm.

frames and content recognition. Although accuracy of all three trackers is acceptable for most tasks, the most accurate tracker is the KLT, followed by RIFF then NCC.

## 6. Conclusions

We have proposed an end-to-end RIFF descriptor pipeline that performs state-of-the-art image recognition on a difficult, large-scale dataset. The RIFF algorithm is $15\times$ faster than SURF. Such low complexity enables new applications for hand-held devices, as well as efficiently processing web-scale databases. We have shown that with RIFF it is possible to unify tracking and recognition for MAR. A RIFF based tracker is both fast and accurate. For large-scale image processing, we can alleviate an existing computational bottleneck. Extracting features from 1 million images with SURF takes $\sim 4$ days on a single core, while doing so with RIFF takes only 4 h. Looking towards even larger databases, it should be possible to process 1 billion images in 11 days using a single machine with 16 cores.

These results are possible for several key reasons. First, our interest point detector has low complexity. Second, descriptor computation re-uses the results of interest point detection. The interest point detector provides a properly anti-aliased and subsampled scale-space at no additional cost. Finally, there is no pixel interpolation or gradient rotation. This is possible because radial gradients enable us to place the gradient, without any modification, in the proper spatial bin. As a result, our algorithm is efficient and elegant.

## References

[1] H. Bay, T. Tuytelaars, L.V. Gool, SURF: speeded up robust features, in: Proceedings of the European Conference on Computer Vision (ECCV), Graz, Austria, 2006 ⟨http://www.vision.ee.ethz.ch/~surf/⟩.
[2] D. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision 60 (2) (2004) 91–110.
[3] G. Takacs, V. Chandrasekhar, D.M. Chen, S.S. Tsai, R. Grzeszczuk, B. Girod, Unified real-time tracking and recognition with rotation invariant fast features, in: Conference on Computer Vision and Pattern Recognition (CVPR), SFO, California, 2010.
[4] V. Lepetit, P. Fua, Keypoint recognition using randomized trees, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (2006) 1465–1479.
[5] E. Rosten, T. Drummond, Machine learning for high-speed corner detection, in: European Conference on Computer Vision, 2006, pp. 430–443.
[6] M. Agrawal, K. Konolige, M.R. Blas, CenSurE: center surround extremas for realtime feature detection and matching, in: European Conference on Computer Vision, 2008, pp. 102–115.
[7] M. Calonder, V. Lepetit, C. Strecha, P. Fua, BRIEF: binary robust independent elementary features, in: European Conference on Computer Vision (ECCV 2010), Berlin, Heidelberg, 2010.
[8] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, M. Pollefeys, Building Rome on a cloudless day, in: European Conference on Computer Vision, ECCV'10, Berlin, Heidelberg, 2010.
[9] S. Heymann, B. Frhlich, F. Medien, K. Mller, T. Wiegand, SIFT implementation and optimization for general-purpose GPU, in: International Conference in Central Europe on Computer Graphics and Visualization (WSCG), 2007.
[10] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, ORB: an efficient alternative to SIFT or SURF, in: Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain, 2011.
[11] S. Leutenegger, M. Chli, R.Y. Siegwart, BRISK: binary robust invariant scalable keypoints, in: Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain, 2011.
[12] Z. Wu, Q. Ke, J. Sun, H.-Y. Shum, A multi-sample, multi-tree approach to bag-of-words image representation for image retrieval, in: International Conference on Computer Vision, 2009.
[13] C. Silverstein, Google Performance Tools, 2011 ⟨http://code.google.com/p/google-perftools⟩.
[14] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, Y. Reznik, R. Grzeszczuk, B. Girod, Compressed histogram of gradients: a low-bitrate descriptor, International Journal of Computer Vision 96 (3) (2012) 384–399.
[15] Y. Reznik, G. Cordara, M. Bober, Evaluation framework for compact descriptors for visual search, in: ISO/IEC JTC1/SC29/WG11/N12202, 2011.
[16] B. Girod, V. Chandrasekhar, D.M. Chen, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S.S. Tsai, R. Vedantham, Mobile visual search, IEEE Signal Processing Magazine 28 (2011) 61–76.
[17] J. Philbin, O. Chum, M. Isard, J. Sivic, A. Zisserman, Lost in quantization – improving particular object retrieval in large scale image databases, in: Conference on Computer Vision and Pattern Recognition (CVPR), Anchorage, Alaska, 2008.
[18] A. Vedaldi, UCLA Vision Library's SIFT Code ⟨http://www.vlfeat.org/⟩.
[19] G. Bradski, The OpenCV Library, Dr. Dobb's Journal of Software Tools ⟨http://opencv.willowgarage.com/wiki/⟩.
[20] Stan Birchfield, KLT: An Implementation of the Kanade-Lucas-Tomasi Feature Tracker, 2007 ⟨http://www.ces.clemson.edu/~stb/klt⟩.