Speeding Up SURF

Peter Abeles

Robotic Inception
pabeles@roboticinception.com

Abstract. SURF has emerged as one of the more popular feature descriptors and detectors in recent years. While considerably faster than SIFT, it is still considered too computationally expensive for many applications. In this paper, several algorithmic changes are proposed to create two new SURF like descriptors and a SURF like feature detector. The proposed changes have comparable stability to the reference implementation, yet a byte code implementation is able run several times faster than the native reference implementation and faster than all other open source implementations tested.

1 Introduction

The problem of associating features inside one image against another related image is image correspondence. Knowing image correspondence allows for the scene's structure and camera motion to be determined, in addition to object recognition. A typical processing flow for point based image correspondence involves, interest point detection, description of local regions, and feature association.

In recent years, Bay *et al.*'s Speeded-Up Robust Features (SURF) [1] has emerged as a popular choice for interest point detection and region description. Building upon previous work (e.g. SIFT [2]), SURF is primarily designed for speed and invariance to scale and in-plane rotation. While skew, anti-isotropic scaling, and perspective effects are considered second order.

SURF's speed is a significant improvement over its predecessors, but it is still considered to be too slow for many applications. In particular, embedded systems with constrained computational resources. More recently developed feature detectors/descriptors [3,4,5] have focused on improving speed while maintaining about the same level of stability found in SURF.

In this paper, several algorithmic changes to SURF are proposed. With the intent of improving upon the original algorithm's runtime performance while maintaining stability. The proposed algorithmic changes are justified through adherence to the smoothness rule (defined below) and a performance study. Two different SURF like descriptors are created from these proposed changes, SURF-S and SURF-F, along with a SURF like detector. SURF-S provides comparable stability to the reference implementation while running several times faster than the reference library. SURF-F sacrifices a bit of stability for more than two times speed improvement relative to SURF-S. Source code for the proposed

algorithm is freely available and included in the open source computer vision library BoofCV [6].

2 Speeded-Up Robust Features

The following is a high level overview of the SURF detector and descriptor. For a complete discussion consult the SURF paper [1]. SURF achieves speed across a range of scales through the use of integral images [7,8]. Transforming an image into an integral image allows the sum of all pixels contained inside arbitrary axis aligned rectangle to be found in four floating point operations.

The value of each pixel (x, y) in the integral image I_Σ is computed by summing pixel intensities within a rectangle up to (x, y):

$$I_\Sigma(x,y) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \tag{1}$$

Then to find the sum of pixel values contained in a rectangle R_Σ compute:

$$\begin{array}{c} R_\Sigma(x_1,y_1,x_2,y_2) = I_\Sigma(x_2,y_2) - I_\Sigma(x_2,y_1-1) \\ -I_\Sigma(x_1-1,y_2) + I_\Sigma(x_1-1,y_1-1) \end{array} \tag{2}$$

where $(x_1, y_1) \le (x_2, y_2)$.

Interest point detection is done using an approximation of the Hessian determinant scale-space detector [9]. The Hessian's determinant is found by approximating the Gaussian's second order partial derivatives (D_{xx}, D_{yy}, D_{xy}) using box integrals, as described in [1].

$$det(H) = D_{xx}D_{yy} - (wD_{xy})^2 \tag{3}$$

This is done across different sized regions and scales. Interest points are defined as local maximums in the 2D image and across scale-space. Scale and location are interpolated by fitting a 3D quadratic [10] to feature intensity values in the local 3x3x3 region.

Several different variations on the SURF descriptor are described in [1], but only the oriented SURF-64 descriptor is considered in this paper. Orientation is estimated by computing the Haar wavelet (effectively the image gradient) inside a neighborhood of radius of $6s$, where s is the feature's scale. The gradient is weighted by a Gaussian centered at the interest point, its angle computed, and saved into an array. Using a moving window of $\frac{\pi}{3}$ radians, the window with the largest gradient sum is found and the feature's orientation computed from its sum.

The feature description is computed inside a square region of size $20s$, aligned to the found orientation. This region is then broken up into a 4 by 4 grid for a total of 16 sub-regions, which are of size $5s$. For each sub-region the sum of the gradient and sum of the gradient's absolute value is computed:

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{4}$$

These responses are weighted using a Gaussian distribution. Each subregion contributes 4 features (v), resulting in a total of 64 features for the descriptor.

The gradient can only be efficiently computed along the image's axis. To accommodate for the feature's orientation, the gradient is rotated so that it is oriented along the feature's axis.

3 Implementation Details

To create a stable region descriptor or feature detector, small changes in location and scale must cause a proportionally small change to the feature's description or location, respectively. The preceding statement is referred to as the smoothness rule. Similar statements are made by D. Lowe [2] and justified by a biological vision model [11].

The following are several general techniques for enforcing the smoothness rule: 1) Use interpolation functions with continuous values when sampling pixel intensity. 2) Increase a sample region's size to reduces the fractional change in value when crossing a pixel border. 3) Avoid interacting with image and object borders. Technique 2 and 3 can be conflicting since as the region size increases it is more likely to interact with the boundary conditions.

3.1 Descriptor Interpolation

Interpolation of the gradient's response when computing the descriptor (v in Eq. 4) is not fully described in [1]. This has resulted in several different algorithmic interpretations. The most straight forward interpretation is to use nearest-neighbor interpolation. However, this method does not have a smooth transition between pixel boundaries, degrading descriptor stability.

Agrawl *et al.* [12] propose to have each subregion overlap by adding a padding of 2s and to weigh the gradient using a subregion centered Gaussian distribution. The resulting descriptor has a region of size $24s$. Pan-o-Matic [13] samples the gradient using a variable number of points depending on the ratio of region size to sample size and then uses trilinear interpolation to compute the descriptor values, similar the approach used by SIFT. SURF-S uses the technique proposed by Agrawl *et al.*, while SURF-F uses nearest-neighbor interpolation.

3.2 Image Border

Interest points can have descriptors which extend outside of the image border. How this edge case is handled has a significant effect on stability. For example, simply discarding features which touch the border causes a 17% drop in stability when compared to the method proposed below. The SURF paper does not discuss how to handle this case. A good balance between speed and stability is found by setting the response of any operator crossing the image border to be zero. This approach is used by both SURF-S and SURF-F.

3.3 Interest Point Interpolation

In SURF, after an interest point has been detected using non-maximum suppression, its position (x, y, s) is interpolated as the extreme of a 3D quadratic, see Brown and Lowe [10]. This technique uses second order derivatives (Laplacian) computed using pixel differences, which amplifies noise. Ad hoc modifications are required to filter out illogical solutions generated with this approach.

To avoid these issues, it is proposed that a 1D quadratic is used instead. If the minimum number (which is 3) of points are used and the center point is the peak, the interpolated peak must lie inside the local region. Both SURF-S and SURF-F work by fitting quadratic 1D polynomials across each (x, y, s) axis independently. While not capturing off axis structural information, empirical tests show comparable stability, is easier to implement, requires fewer operations, and does not require numerical differentiation.

3.4 Derivative Operator

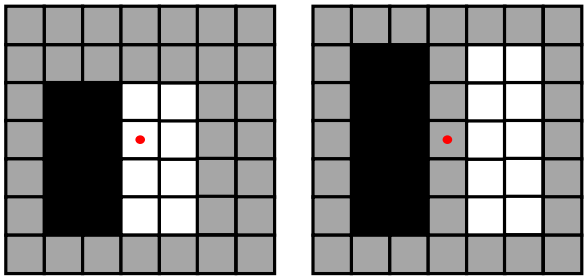

Fig. 1: Left: A common interpretation for the Haar-like derivative operator. Right: Proposed symmetric derivative operator. Red dot indicates the region's center. The Haar kernel lacks symmetric, causing a bias. Dark squares indicate a weight of -1 and white squares +1.

The SURF paper states that a Haar wavelet is compute at regularly spaced sample points inside each sub-region when computing the description. A common interpretation of this statement is to use a template similar to the one shown in the left side of Figure 1, as has been done by OpenSURF [14,15] and OpenCV [16]. This template lacks symmetry, which causes a directional bias. An alternative symmetric derivative operator is proposed that overcomes this issues, see Figure 1. The alternative kernel has a width of $w = \text{rnd}(2rs) + 1$, where r is the radius at a scale of one. A value of $r = 1$ is recommended for descriptor computations.

Both SURF-S and SURF-F use the symmetric derivative operator.

3.5 Orientation Estimation

The region orientation algorithm proposed by SURF is computationally expensive, see Section 2 for a summary. An alternative and much faster approach is to compute a weighted sum of the gradient $(\sum d_x, \sum d_y)$ and then find the angle using atan2$(\sum d_y, \sum d_x)$, where the gradient is found using pixel differences, e.g. Sobel. However, the improved speed comes at the cost of some stability.

SURF-F uses the orientation estimation technique described above, while SURF-S uses the original algorithm proposed in SURF.

3.6 Laplacian Sign

Another smaller performance boost can be found in delaying the Laplacian's sign computation. It is stated in [1] that the Laplacian's sign can be computed with no loss in performance. This is not quite true; the computation requires an additional operation for each pixel and scale, plus storage. Instead if the Laplacian sign is computed for found interest points only, then 24 additional operations are required per feature. Since the number of pixels is much greater than the number of found features, the latter is many times faster and requires no additional storage.

4 Test Setup

Since SURF was originally proposed, numerous implementations have released for various programming languages and hardware platforms. A reference implementation [17] has been released by the original authors, but only in an out-of-date binary format. To validate the performance of the proposed changes, a comparison of several open source libraries and the reference library is performed.

Implementation	Cite	Version	Language	Comment
SURF-F	[6]	v0.12	Java	Faster but less accurate
SURF-S	[6]	v0.12	Java	Slower but more accurate
JavaSURF	[18]	SVN r4	Java	No orientation
JOpenSURF	[15]	SVN r24	Java	Java port of OpenSURF
OpenCV	[16]	2.4.3	C++	
OpenSURF	[14]	12/04/2012	C++	
Pan-o-Matic	[13]	0.9.4	C++	
Reference	[17]	1.0.9	C++	Original author

Fig. 2: List of evaluated implementations in alphabetical order. If a formal version is lacking or insufficient then a repository version is referenced. The proposed algorithms (SURF-F and SURF-S) are included in the BoofCV library.

Evaluation is performed using test image sequences from Mikolajczyk and Schmid [19]. Each sequences has a set of known image homographies relating

images to the first image in the sequence. A homography provides a one-to-one relationship for pixel locations between two images. Each sequence is designed to test different types of distortion and image noise. The evaluated data sets include "bark", "bikes", "boat", "graf", "leuven", "trees", "ubc", and "wall".

Evaluated libraries are listed in Table 2. Only single threaded libraries are considered. The two proposed modifications to SURF have been written in Java, but both C++ and Java libraries are considered. Libraries written in C++ have a 2 to 3 times speed advantage due to less overhead.

Several parts of SURF are easily parallelized, such as the descriptor calculation. While beyond the scope of this paper, multi-threaded [16] and GPU [20] assisted implementations are also readily available. These implementations are not considered since the focus of the paper is on algorithmic improvements which are independent of hardware.

5 Performance Metrics

Performance metrics used to evaluate detector stability, and descriptor stability are described in the sub-sections below. The performance metric for runtime speed is elapsed time.

5.1 Descriptor

Descriptor stability is evaluated using F-measure. F-measure is defined using precision and recall statistics. Precision is defined as true positives divided by true positives and false positives. Recall is defined as true positives divided by true positives and false negatives. Recall combines both metrics into a single number.

$$\mathrm{F} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{5}$$

A true positive is defined as an associated feature for which its location is within tolerance of the true location, as defined by the known homography. False negatives are features with corresponding points in both image, but which are not associated. Association is done using a greedy algorithm that minimized Euclidean error.

All libraries use interest points detected by the reference library to compute descriptors from. By doing so, the descriptor's stability can be decoupled from the detector's performance. Each library is configured to describe SURF-64 features. Summary statistics shown in the left side of Figure 4 are found by summing the F-measure across each image in every sequences.

5.2 Detector

Detector stability is measured using a modified version of repeatability. As stated in [21] repeatability is a measure that "signifies that detection is independent of changes in imaging conditions". One problem with repeatability is it favors

detectors that detect more features [22]. Excessive detections increase computational cost without improving association quality. An extreme example is if every pixel is marked as an interest point, it would have perfect repeatability.

To compensate for this issue, the definition of repeatability has been modified to ignore regions with closely packed points. By only considering interest points with unambiguous matches, repeatability bias is reduced.

The modified repeatability measure r_i is defined below:

$$r_i = \frac{|A_i| - |T_i|}{|P_i| - |T_i|} \tag{6}$$

where P_i is the set all points, A_i is the set of actual matches, and T_i is the set of ignored matches.

$$P_i = \{x_a \in F_o | H_i x_a \in I_i\} \tag{7}$$

$$A_i = \{x_c \in F_i | \parallel H_i x_b - x_c \parallel < \epsilon, x_b \in P\} \tag{8}$$

$$T_i = \{x_d \in F_i | \parallel x_c - x_d \parallel < \epsilon, x_d \neq x_c\} \tag{9}$$

where I_i is image i, H_i is the homography transform from image 1 to i, F_i is the set of all detected interest points, x is an interest point, and ϵ is the match tolerance.

Two interest points are considered a match if their position and scale are within tolerance. The true position is found using the provided homography. Scale is computed by 1) sampling four evenly spaced points one pixel away from the interest point, 2) applying homography transform to each sample point and interest point, 3) finding the distance of transformed sample points from transformed interest point, and 4) setting expected scale to average distance.

Tuning each library to detect the same number of features in all images proved to be impossible. Instead they are tuned to detect about 2,000 features in image 1 in the graf sequence.

Detector configuration:

1. Octaves: 4
2. Scales: 4
3. Base Size: 9
4. Pixel Skip: 1

Tolerance for position is 1.5 pixels and 25% for scale. Relative ranking were found to be insensitive to changes in threshold values.

Summary statistics shown in right side of Figure 4 are found for each implementation by summing repeatability across each image in every sequence and dividing by the best implementation's score.

5.3 Runtime Speed

Runtime performance is measured by having each library detect and describe about 2,000 features inside image 1 in graf sequence. Evaluation procedure:

1. Measure elapsed time to detect and describe features.
2. Repeat 10 times in the same process and output best result.
3. Run the whole experiment 11 times for each library and record the median time.

All tests are performed on a desktop computer with Ubuntu 10.10 installed and an Intel Q6600 2.4GHz CPU. Native libraries are compiled using g++ 4.4.5 with the -O3 flag. Java libraries are compiled and run using Oracle JDK 1.6.38 64 bit. No additional flags are passed to the Java Runtime Environment, the -server flag is implicit.

Native library runtime speeds are highly dependent upon the level of optimization done by the compiler and which instructions they are allowed to use. Additional hardware specific flags are not manually injected into build scripts beyond what was provided by the authors. Elapsed time is measured in the actual application using System.currentTimeMillis() in Java and clock() from time.h in C++.

6 Performance Results

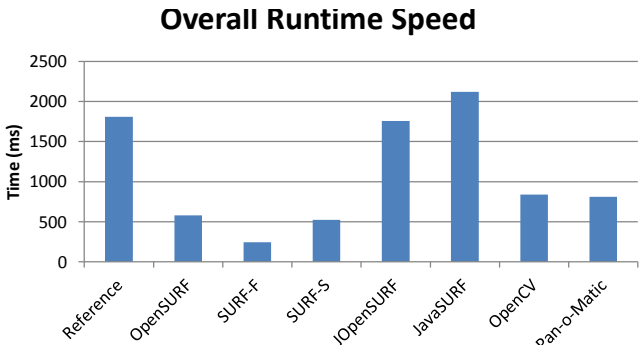

Fig. 3: Runtime speed for detecting and describing image features. Lower bars are better. Each library is tuned to detect approximately 2000 features in a 850x680 image.

Summary results for runtime performance, descriptor stability, and detector stability are shown in Figures 3, and 4. Stability results for describe and detection for individual sequences are shown in Figures 5 and 6, respectively.

Pan-o-Matic and the reference library have the best descriptor stability, closely followed by SURF-S. For detector stability, the proposed detector has the best stability, followed by Pan-o-Matic and the reference library. SURF-F is the fastest implementation, despite being written in Java. The runners-up are OpenSURF and SURF-S, which have nearly the same runtime speed, but are two times slower than SURF-F.

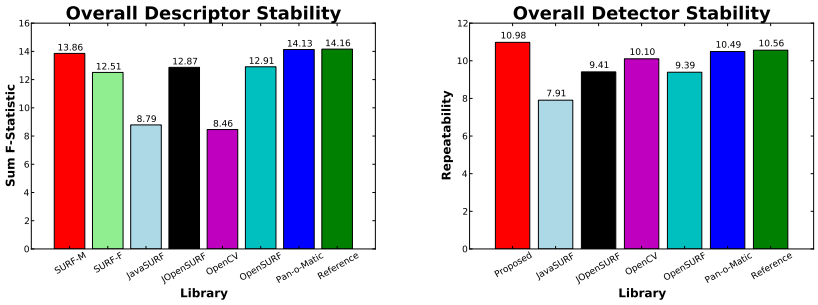

Fig. 4: Left: Summary of descriptor stability using correct association fraction across all image sequences. Right: Summary of detector stability using a modified repeatability measure across all image sequences. Higher bars are better.

7 Conclusions

To improve the runtime speed of SURF, several algorithmic changes have been proposed, resulting in two new descriptor variants and one detector. The proposed changes are designed to be more computationally efficient and follow the smoothness rule to ensure stability. Performance of the proposed changes are validated by a stability and runtime performance study of eight SURF implementations. Source code is available inside the BoofCV open source library.

Descriptor stability varied significantly by implementation, with SURF-S having comparable performance to the reference library, the top performer. SURF-F was shown to be about 10% less stable than SURF-S, which was still comparable to or significantly better than four other implementations. The proposed detector had the best stability. The proposed detector, combined with SURF-S and SURF-F, were the two fastest SURF implementations. SURF-F running more than twice as fast as SURF-S. It is worth noting that proposed algorithms were implemented in Java and run in a virtual machine. Due to overhead, a well written port to C++ is likely to run 2 to 3 times faster.

References

1. Bay, H., Ess, A., Tuytelaars, T., Gool, L.V.: Surf: Speeded up robust features. Computer Vision and Image Understanding (CVIU) **110** (2008) 356–359
2. Lowe, D.: Distinctive image features from scale-invariant keypoints, cascade filtering approach. International Journal of Computer Vision (IJCV) **60** (2004) 91–110
3. Tola, E., Lepetit, V., Fua, P.: Daisy: an Efficient Dense Descriptor Applied to Wide Baseline Stereo. Pattern Analysis and Machine Intelligence **32** (2010) 815–830
4. Juan, L., Gwon, O.: A Comparison of SIFT, PCA-SIFT and SURF. International Journal of Image Processing (IJIP) **3** (2009) 143–152
5. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: BRIEF: Binary Robust Independent Elementary Features. In: European Conference on Computer Vision. (2010)

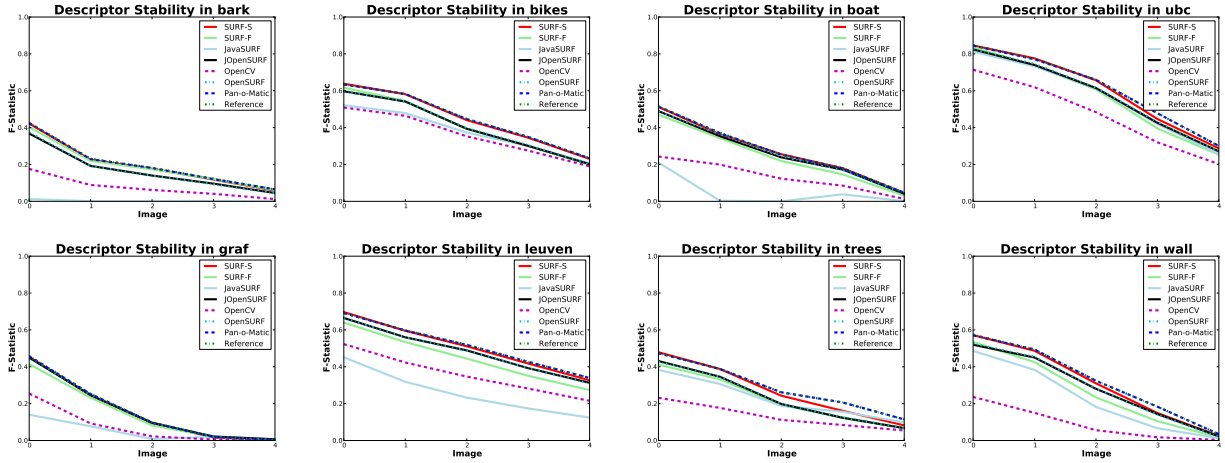

Fig. 5: Overall descriptor stability. Sum of F-statistic across all image sequences. All implementations use the same set of interest points (provided by reference library) to decouple descriptor performance from detector performance.

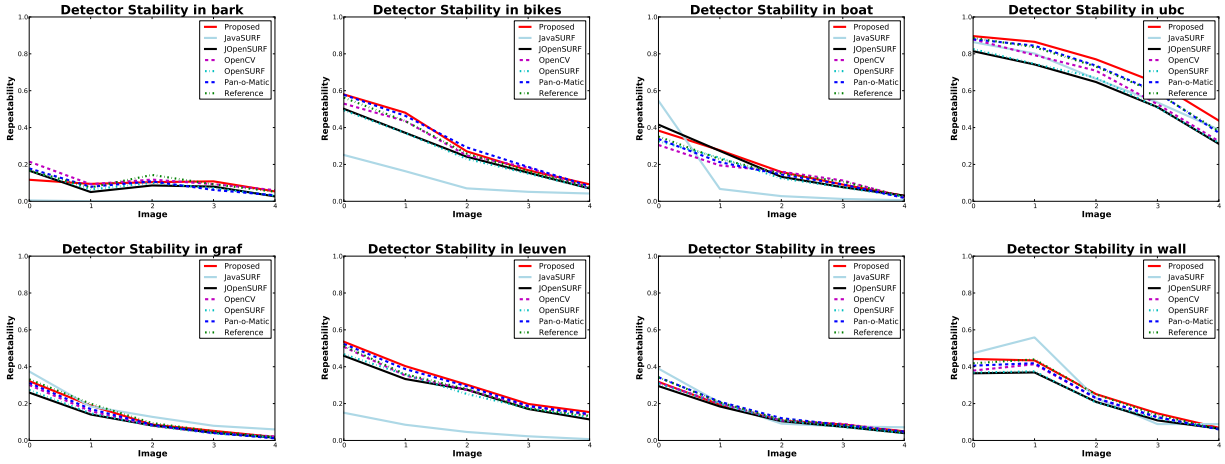

Fig. 6: Overall detector stability. Sum of repeatability across all image sequences.

6. Abeles, P.: Boofcv. http://boofcv.org (Version 0.5)
7. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition (CVPR). (2001) 511–518
8. Simard, P., Bottou, L., Haffner, P., LeCun, Y.: A fast convolution algorithm for signal processing and neural networks. In: NIPS. (1998)
9. lindeberg, T.: Feature detection with automatic scale selection. IJCV **30** (1998) 79–116
10. Brown, M., Lowe, D.: Invariant features from interest point groups. In: BMVC. (2002)
11. Edelman, S., Intrator, N., Poggio, T.: Complex cells and object recognition. http://kybele.psych.cornell.edu/ edelman/archive.html (1997)
12. Agrawal, M., Konolige, K., Blas, M.: Censure: Center surround extremas for real-time feature detection and matching. In: Computer Vision ECCV 2008. Volume 5305. (2008) 102–115
13. Orlinski, A.: Pan-o-matic. http://aorlinsk2.free.fr/panomatic/ (Version 0.9.4)
14. Evans, C.: The opensurf computer vision library. http://www.chrisevansdev.com/computer-vision-opensurf.html (Build 27/05/2010)
15. Stromberg, A., jojopotato, N.: Jopensurf. http://code.google.com/p/jopensurf/ (SVN r24) Note: Port of OpenSURF.
16. Liu, L., Mahon, I.: Opencv. http://opencv.willowgarage.com/wiki/ (Version 2.3.1 SVN r6879)
17. Bay, H., Gool, L.V.: Surf: Speeded up robust feature. http://www.vision.ee.ethz.ch/ surf/ (Version 1.0.9)
18. Fantacci, C., Martini, A., Mitreski, M.: Javasurf. http://code.google.com/p/javasurf/ (SVN r4) Note: Refactored P-SURF.
19. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell. **27** (2005) 1615–1630
20. Cornelis, N., Van Gool, L.: Fast scale invariant feature detection and matching on programmable graphics hardware. In: Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on. (2008)
21. Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. Int. J. Comput. Vision **37** (2000) 151–172
22. Gauglitz, S., Hllerer, T., Turk, M.: Evaluation of interest point detectors and feature descriptors for visual tracking. International Journal of Computer Vision **94** (2011) 335–360