An Evaluation of Open Source SURF Implementations

David Gossow, Peter Decker, and Dietrich Paulus

Active Vision Group
University of Koblenz-Landau
Universitätsstr. 1
56070 Koblenz, Germany
dgossow@uni-koblenz.de
http://robots.uni-koblenz.de

Abstract. SURF (Speeded Up Robust Features) is a detector and descriptor of local scale- and rotation-invariant image features. By using integral images for image convolutions it is faster to compute than other state-of-the-art algorithms, yet produces comparable or even better results by means of repeatability, distinctiveness and robustness. A library implementing SURF is provided by the authors. However, it is closed-source and thus not suited as a basis for further research.

Several open source implementations of the algorithm exist, yet it is unclear how well they realize the original algorithm. We have evaluated different SURF implementations written in C++ and compared the results to the original implementation.

We have found that some implementations produce up to 33% lower repeatability and up to 44% lower maximum recall than the original implementation, while the implementation provided with the software Pan-o-matic produced almost identical results.

We have extended the Pan-o-matic implementation to use multithreading, resulting in an up to 5.1 times faster computation on an 8-core machine. We describe our comparison criteria and our ideas that lead to the speed-up. Our software is put into the public domain.

1 Introduction

Many problems of computer vision can be solved by finding point correspondences between images using local features. Examples are object recognition, depth reconstruction and self localization. The first step in computing local features consists of *detecting* salient locations such as corners, blobs and t-junctions. From the neighbourhood regions of these *interest points*, image features are then calculated, yielding a *descriptor* for each one. Corresponding points between two images can then be found by comparing these descriptors.

SURF (Speeded Up Robust Features) [BTVG06, BETvG08] aims at being invariant to weakly affine transformations (scaling and in-plane rotation) and homogeneous changes in intensity, robust to other typical geometric and photometric transformations and fast to compute. It detects blob structures of arbitrary size in grayscale images by searching maxima of the determinant of the

J. Ruiz-del-Solar, E. Chown, and P.G. Ploeger (Eds.): RoboCup 2010, LNAI 6556, pp. 169–179, 2011.

Hessian matrix over the spacial dimensions as well as a wide range of scales. The dominant orientation of gradients in the blob's neighbourhood, together with its scale, are used to construct a normalized descriptor window. From this window, a feature vector containing gradient information is calculated.

SURF makes use of box filters instead of Gaussian kernels to construct the scale space representation. These are a combination of filters calculating the mean value of a rectangular image region in constant time. This is achieved by using integral images as data structure.

Its invariance and run time properties make SURF particularly suited for applications in highly uncontrolled environments where computation time is a critical issue, such as in the field of autonomous robotic systems.

Several evaluations of different local feature detectors and descriptors have been made [MS05, MTS+05, MP07]. However, in practice the actual implementation of an algorithm can also have a large influence on its performance.

In this paper, we evaluate the performance of different implementations of the SURF algorithm. We focus on open source implementations of SURF, as they are best suited as a basis for further research and improvements. A list of the implementations can be found in section 2.

To assure the comparability of our results, we rely on an established evaluation framework already used in several other publications [BTVG06, BETvG08, MS05, MTS+05]. It consists of image sequences with different transformations and tools for calculating performance values, as described in section 3. The results show significant differences between the implementations, as described in section 4.

In section 4.3, we take a closer look on the implementations which produced the best results in our evaluation. We show that the use of linear interpolation between the bins of the descriptor window significantly increases its performance. In section 5 we describe how to parallelize parts of the algorithm, speeding up its computation on multi-core machines.

Section 6 gives hints to related publications. A resume of the contributions made by this paper is given in section 7.

2 The Contestants

All tested implementations are available as open source software, except for the library released by the authors of SURF[1].

- *OpenSURF*[2] is a dedicated library implementing the SURF algorithm. We have compared two different releases of it. The first one, released on 22 March 2009, implements the algorithm as described in the original SURF publication [BETvG08, Low04]. The second release of 31 August 2009 contains a modified algorithm for calculating the descriptor as described in [AKB08].

[1] `http://www.vision.ee.ethz.ch/~surf/`
[2] `http://code.google.com/p/opensurf1/`

- *OpenCV*[3] and *dlib 17.20*[4] are image processing libraries which contain implementations of SURF. We used the OpenCV repository version of 22 April 2010 for our evaluations. It supports multithreading using OpenMP[5].
- *Pan-o-matic 0.9.4*[6] is a tool for computing point correspondences between images for use in panorama creation tools, which includes an implementation of SURF.
- The library *libmv*[7] contains an implementation of SURF. In the current version (April 2010), it does not support rotation invariance and was not included in the evaluation.
- *P-SURF*[8] is a modified version of OpenSURF using multi-threading as described in [Zha10]. According to the author, it is based on the March 2009 release of OpenSURF or an earlier one and contains only minor modifications besides the multi-threading support. Due to time reasons, it was not included in the evaluation.

3 Evaluation Framework

Our evaluation is based on Mikolajczyk's software framework and image sequences[9]. It was also used in other evaluations [MS05, MTS+05] and in the original SURF publications [BTVG06, BETvG08].

The parameters of all implementations were set to the same values. The initial sampling step was set to 1 pixel and the number of searched octaves to 4. No image doubling was used. The descriptor was configured to yield a rotation-invariant 64-dimensional feature vector.

3.1 Image Sequences

The sequences used in this evaluation each contain 6 images of natural textured scenes with increasing geometric and photometric transformations. They are either planar or the camera remained at a fixed position during the acquisition, so the images of one sequence are related by a homography (plane projective transformation). The homography matrices are provided with the image data. They were calculated by manually defining point correspondences between the images and refining the resulting homography using a robust small-baseline homography estimation algorithm.

- The *Bark* and *Boat* sequences (Figures 4a and 4b) contain images of two scenes with increasing zoom and different rotations of the image.

[3] http://sourceforge.net/projects/opencvlibrary/
[4] http://dclib.sourceforge.net/
[5] http://openmp.org
[6] http://aorlinsk2.free.fr/panomatic/
[7] http://code.google.com/p/libmv/
[8] http://www.xjtlu.edu.cn/depts/csse/csse1511
[9] http://www.robots.ox.ac.uk/~vgg/research/affine/

- The *Graffiti* and *Bricks* sequences (Figures 4c and 4d) contain images of planar scenes. The viewpoint changes up to approximately 60°.
- In the *Bikes* and *Trees* sequences (Figures 4e and 4f), the focus of the camera was altered to create an increasing effect of image blur.
- In the *Cars* sequence (Figure 4g), the aperture of the camera was altered, varying from under- to over-exposition.
- The *UBC* sequence (Figure 4h) contains several versions of the same image with an increasing degree of JPEG compression artefacts.

3.2 Performance Criteria

The performance criteria are *repeatability* for the detection step and *precision and recall* for the descriptor.

The *repeatability* of a detector is defined by the percentage of interest points detected again under a given transformation of the image. To calculate it, the regions representing the interest points are mapped from one image to the other using the homography. Is is then checked how many of them overlap with a region from the other image by a given minimal percentage. Only interest points are taken into account that are visible in both images (see Figure 1).

In the case of SURF, each interest point with scale σ is assigned a circular region with radius 10σ. If the overlap ratio of two regions A and B is above a threshold ϵ_0, then the interest points represented by them are considered as corresponding. The overlap ratio of two regions is determined by the ratio of their intersection and union (see Figure 2):

$$\frac{A \cap B}{A \cup B} \geq \epsilon_0$$

If the images contain n_1 and n_2 relevant interest points, then the repeatability is defined as

$$\text{repeatability} = \frac{\#\text{correspondences}}{\min(n_1, n_2)}$$

(a) Image 1 (b) Image 2 (c) Intersection

Fig. 1. (a) and (b) show the first two images of the *Graffiti* sequence. (c) shows the region visible in both images (green) determined by the homography (gray).

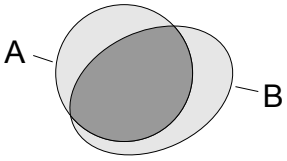


Fig. 2. Two overlapping regions. One region is circular, while the other one was mapped from another image by a homography and is thus distorted.

SURF uses a threshold on the determinant of the Hessian matrix to filter out weakly localized interest points. This threshold was set to zero for all implementations, thus effectively disabling it. The 500 interest points having the strongest determinant of Hessian response were then selected for each image and implementation and used for the evaluation.

The measures we use for descriptor performance are *precision* and *recall*. The precision is represented by the number of correct matches relative to the total number of found matches:

$$\text{precision} = \frac{\#\text{correct matches}}{\#\text{matches}}$$

Recall is the number of correct matches relative to the total number of corresponding interest regions:

$$\text{recall} = \frac{\#\text{correct matches}}{\#\text{correspondences}}$$

To calculate precision and recall, each interest point in one image is matched to the closest neighbour in the other image based on the euclidean distance of their feature vectors. Matches are rejected if their *distance ratio* [Low04] is above a threshold t_Φ. It is defined as the ratio of the distances to the closest neighbour and to the second-closest neighbour. By varying t_Φ, a sequence of precision-recall pairs is obtained which allows the comparison of different descriptors.

4 Results

4.1 Detector Comparison

Figures 5 and 6 show the repeatability of all implementations under various image transformations.

The implementation included in Pan-o-matic produces nearly identical results to the original SURF library. In the *Bark*, *Graffiti*, *Bricks*, *Trees* and *UBC* sequences, they produce the highest repeatability, in the others OpenCV shows a higher performance. In the *Bark* sequence, OpenCV does not detect any interest points in the region of interest of the third and all following images.

4.2 Descriptor Comparison

Figure 7a shows the evaluation results of the different descriptor implementations. The Pan-o-matic implementation produces the same results as the original SURF library, followed by the August release of OpenSURF, OpenCV and the March release of OpenSURF. The Dlib implementation performs significantly worse in comparison to the others.

4.3 Effect of Linear Interpolation

In contrast to the description given in the original SURF publication [BTVG06, BETvG08], Pan-o-matic as well as the second release of OpenSURF interpolate between the bins of the descriptor window. Pan-o-matic uses linear interpolation, while OpenSURF uses Gaussian functions centred at each bin as described in [AKB08].

To measure the effect of interpolation, we removed the interpolation step from Pan-o-matic. Figure 7b shows that the recall drops by approximately 7 percentage points if no interpolation is used.

By analysing the source code of Pan-o-matic, we have found that several aspects of it differ from the description given in [BTVG06, BETvG08]. However, as the source code is publicly available, further study of these details will be left to the reader.

5 A Multi-threaded Implementation of SURF

We extended the implementation included in Pan-o-matic so it performs parts of the calculations in parallel by employing the thread pool pattern. A thread pool contains a fixed number of threads. An arbitrary number of tasks can be scheduled for execution. Each thread consecutively removes tasks from the queue and executes them. We used the library *threadpool*[10], which relies on the *Boost threads*[11] library.

In the detection step, the scales of each octave are built and searched in a single task, resulting in a maximal parallelism of 5 threads of execution. In the description process, the orientation and feature vector of each interest point is computed in a single task. Thus, the maximal parallelism in this step is limited by the number of interest points.

The source code is made publicly available[12] under the GPL license.

Figure 3 shows the mean computation time of the multi-threaded implementation for the images used in the previous evaluation. The red bars show the hypothetical upper limit of the speed increase, which is a linear scaling depending on the number of cores used. The experiment was performed on an 8-core 3.0 GHz Intel Xeon machine with 16 GB RAM. The program was compiled using *gcc* with compiler optimizations turned on. The parameters for the SURF

[10] `http://threadpool.sourceforge.net/`
[11] `http://www.boost.org/`
[12] `http://parallelsurf.sourceforge.net/`

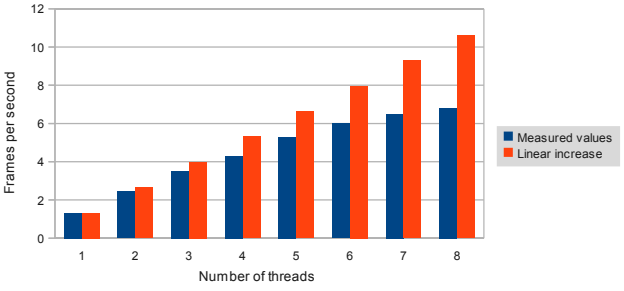
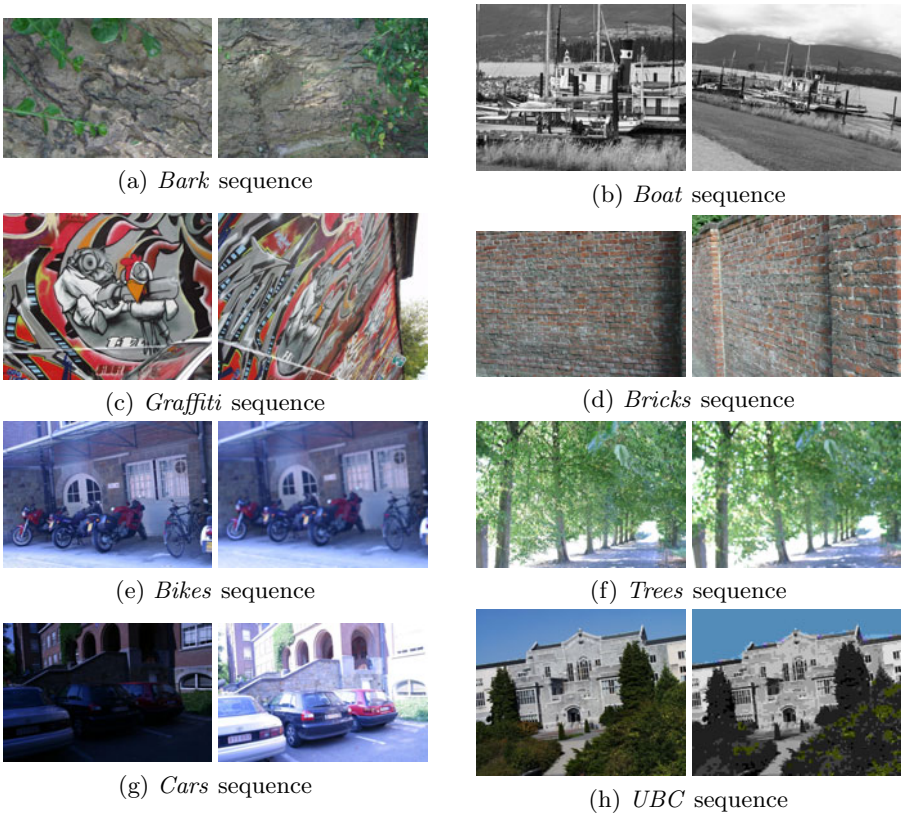


Fig. 3. Mean calculation speed when using different numbers of threads



(a) *Bark* sequence

(b) *Boat* sequence

(c) *Graffiti* sequence

(d) *Bricks* sequence

(e) *Bikes* sequence

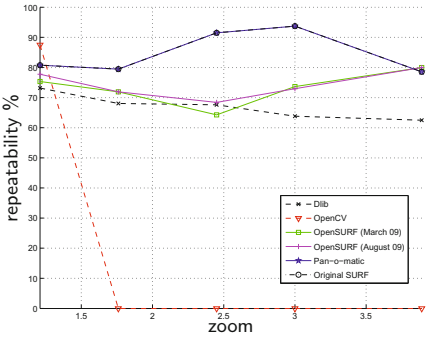
(f) *Trees* sequence

(g) *Cars* sequence

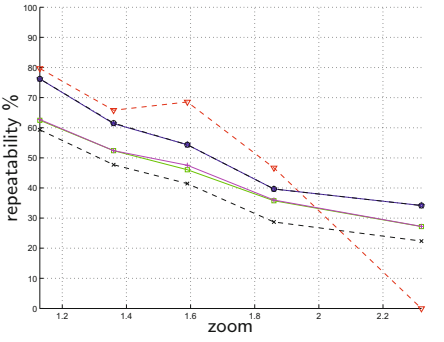
(h) *UBC* sequence

Fig. 4. Examples from the image data set. Shown are the first and last image of each sequence.

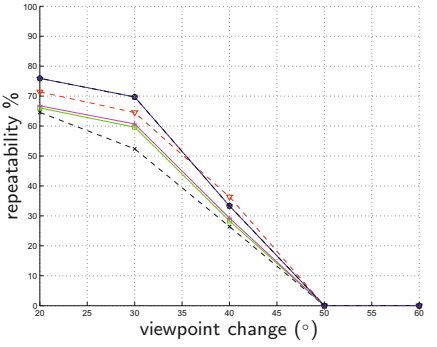
algorithm were set to the default values of the original implementation, except for the initial sampling step, which was set to 1. 6625 interest points were found in the images in average.



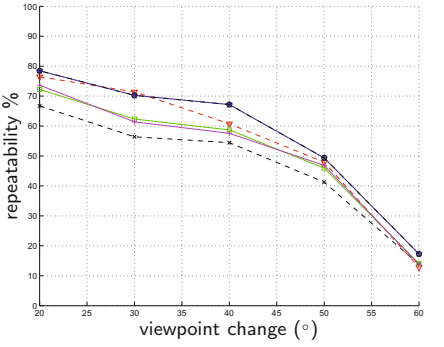
(a) *Bark* sequence



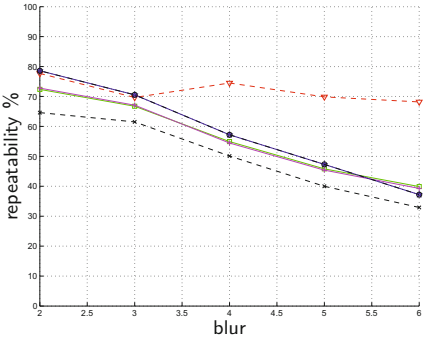
(b) *Boat* sequence



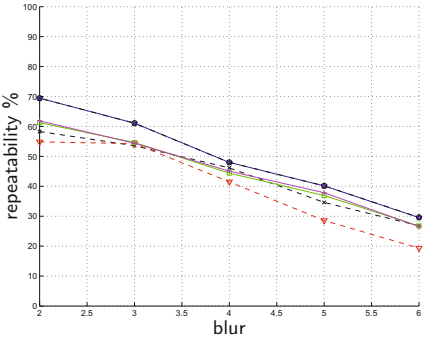
(c) *Graffiti* sequence



(d) *Bricks* sequence



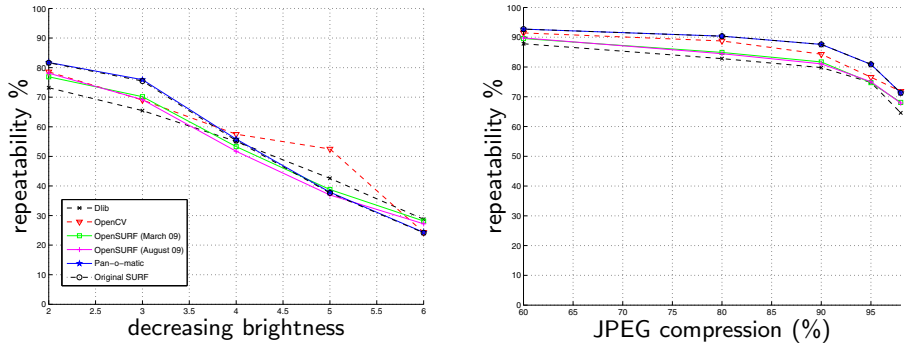
(e) *Bikes* sequence



(f) *Trees* sequence

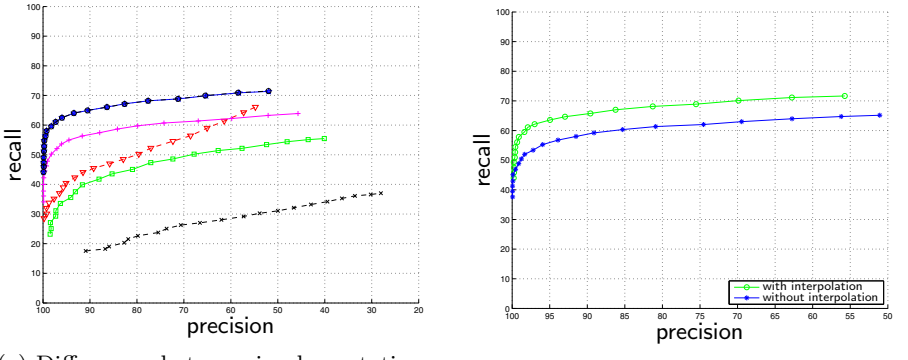
Fig. 5. Repeatability of implementations under various image transformations (pt. 1). Note that the graphs of SURF and Panomatic are identical.

The algorithm runs 1.8 times faster using 2 threads, 3.3 times faster using 4 threads and 5.1 times faster using 8 threads. The single-threaded version runs 3.9 times faster than the original binary SURF library.



(a) *Cars* sequence (b) *UBC* sequence

Fig. 6. Repeatability of implementations under various image transformations (pt. 2)



(a) Differences between implementations (b) Effect of linear interpolation

Fig. 7. Evaluation results for descriptor performance

6 Further Reading

The SURF algorithm was introduced in [BTVG06] and described in more detail in [BETvG08]. The authors use the same framework as this publication [MS05, MTS+05] to evaluate the algorithm. They show that the detector outperforms SIFT [Low04] as well as Harris- and Hessian-Laplace [MS04] and that the descriptor outperforms SIFT and GLOH [MS05].

The reader should be aware that, although the evaluation framework used in this paper includes a variety of typical image transformations, it focuses mainly on the problem of image registration. The evaluation criteria for selecting an adequate algorithm or, in this case, implementation, are always task-dependent. A discussion of the motivation and validity of evaluations such as the one conducted in this paper is given in [Har94, För96].

Other evaluations have been performed that put a focus on other applications such as the recognition of three-dimensional, non-planar objects based on local grayscale [MP07] or color features [BG09]. In [BG09] and [vdSGS08], the performance of local features for object category recognition is evaluated.

7 Conclusion

In this paper, we have evaluated several open source implementations of the SURF algorithm for detecting and describing local grayscale image features. We have presented an evaluation framework which was previously used in other evaluations. The implementation included in the software Pan-o-matic produces the best results, which are at the same time identical to those of the original implementation.

We have pointed out that the best-performing implementations use interpolation in the descriptor step, as opposed to the description given in the original publications [BTVG06, BETvG08]. We have showed that removing the interpolation from the Pan-o-matic implementation results in lower recall values. We have extended the Pan-o-matic implementation to use multi-threading, resulting in an up to 5.1 times faster calculation.

References

[AKB08] Agrawal, M., Konolige, K., Blas, M.R.: CenSurE: Center surround extremas for realtime feature detection and matching. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part IV. LNCS, vol. 5305, pp. 102–115. Springer, Heidelberg (2008)

[BETvG08] Bay, H., Ess, A., Tuytelaars, T., van Gool, L.: Speeded-up robust features (surf). Journal of Computer Vision 110(3), 346–359 (2008)

[BG09] Burghouts, G.J., Geusebroek, J.-M.: Performance evaluation of local colour invariants. Comput. Vis. Image Underst. 113(1), 48–62 (2009)

[BTVG06] Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 404–417. Springer, Heidelberg (2006)

[För96] Förstner, W.: 10 pros and cons against performance characterization of vision algorithms Technical report, Institut für Photogrammetrie, Universität Bonn (1996)

[Har94] Haralick, R.M.: Performance characterization in computer vision. Computer Vision Graphics and Image Processing 60(2), 245–249 (1994)

[Low04] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2), 91–110 (2004)

[MP07] Moreels, P., Perona, P.: Evaluation of features detectors and descriptors based on 3d objects. International Journal of Computer Vision 73(3), 263–284 (2007)

[MS04] Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. International Journal of Computer Vision 60(1), 63–86 (2004)

[MS05] Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(10), 1615–1630 (2005)

[MTS+05] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Van Gool, L.: A comparison of affine region detectors. International Journal of Computer Vision 65(1-2), 43–72 (2005)

[vdSGS08] van de Sande, K.E.A., Gevers, T., Snoek, C.G.M.: Evaluation of color descriptors for object and scene recognition. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 0, pp. 1–8 (2008)

[Zha10] Zhang, N.: Computing optimised parallel speeded-up robust features (p-surf) on multi-core processors. International Journal of Parallel Programming (2010)