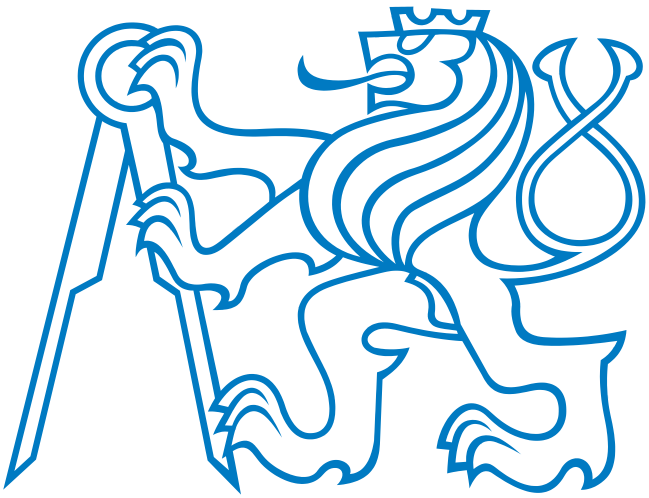
Czech Technical University in Prague
Faculty of Electrical Engineering

MASTER'S THESIS



Karel Lenc

Evaluation and Improvements of Image Interest Regions Detectors and Descriptors

Thesis supervisor: **Prof. Ing. Jiří Matas, Ph.D**

Prague, 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Karel Lenc**

Study programme: Open Informatics
Specialisation: Computer Engineering

Title of Diploma Thesis: **Evaluation and Improvements of Image Interest Regions Detectors and Descriptors**

Guidelines:

1. Investigate the state-of-the-art local feature detectors and descriptors.
2. Implement existing and propose new benchmarks for testing detectors and descriptors from the practical point of view.
3. Using the benchmarks, propose improvements and tune their parameters.
4. Explore the possibility of fast emulation of interest regions detectors and propose improvements.
5. Eventually develop new detector of local features.

Bibliography/Sources:

[1] D. A. Forsyth, J. Ponce: Computer Vision: A Modern Approach; Prentice Hall 2011.
[2] R. Szeliski:Computer Vision: Algorithms and Applications; Springer 2010.
[3] R. Hartley, A. Zisserman: Multiple View Geometry in Computer Vision; Cambridge University Press 2004.
[4] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir and L. Van Gool, A comparison of affine region detectors; IJCV, 2005.
[5] J. Sochman, J. Matas: Learning a Fast Emulator of a Binary Decision Process; ACCV 2007.

Diploma Thesis Supervisor: Prof.Ing. Jiří Matas, Ph.D.

Valid until the summer semester 2013/2014

prof. Ing. Michael Šebek, DrSc.
Head of Department

L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 21, 2013

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne .....

.....
podpis

Acknowledgements

Hereby I would like to thank my supervisor Jiří Matas for his leading and advices. Further, I would like to mention my colleagues from Centre of Machine Perception, mainly Michal Perďoch and Jan Šochman, who helped me greatly with numerous advices.

Thanks to Andrea Vedaldi for leading me during my stay in Visual Geometry Group at University of Oxford and that he allowed me to have this valuable experience. I would like to thank also to Relja Arandjelović from VGG for his help with the Retrieval Benchmark design.

My thanks also belong to my family and friends for their support, without which I would not be able to finish this work.

I acknowledge the financial support of PASCAL Harvest programme without which I would not be able to stay in Visual Geometry Group in University of Oxford participating to VL-Benchmarks project.

Abstract

A reliable and informative performance evaluation of local feature detectors and descriptors is a difficult task that needs to take into account many applications and desired properties of the local features. The main contribution of this work is the extension of the VLBenchmarks project which intends to collect major evaluation protocols of local feature detectors and descriptors.

We propose a new benchmark which evaluates local feature detectors in the image retrieval tasks and simple epipolar criterion for testing detectors and descriptors in the wide baseline stereo problems. Using the extended benchmarks we investigate several parameters of the local feature detection algorithms.

We propose a new algorithm for building a scale space pyramid which significantly improves the detector repeatability in the case of apriori knowledge of the nominal Gaussian blur in the input image. On the image retrieval tasks, we show that features with a small value of the response function improve the performance more than features with small scale, contrary to the observations in the geometry precision benchmarks. By altering the computation of the SIFT descriptor, we show that it is not necessary to weight the patch gradient magnitudes when input images are similarly oriented and that for blob-like features increasing the measurement region improves the performance.

Finally we propose an improvement of emulated detectors that allows finding new image features with better geometric precision. We have also improved the classification time of the emulated detectors and achieved higher performance than the handcrafted OpenSURF detector implementation.

Abstrakt

Spolehlivé a dostatečně informativní měření výkonnosti detektorů zájmových oblastí není snadným úkolem a závisí jak na konkrétní aplikaci, tak i na jejich požadovaných vlastnostech. Hlavním přínosem této práce je rozšíření open source projektu VLBenchmarks, který se snaží nashromáždit hlavní protokoly pro měření vlastností detektorů zájmových oblastí a jejich deskriptorů.

V rámci práce jsme navrhli nový testovací protokol měřící výkonnost detektorů a deskriptorů zájmových oblastí použitelných v systémech pro vyhledávání instancí obrazů v rozsáhlých databázích. Dále jsme navrhli jednoduché kritérium pro testování detektorů a deskriptorů ve wide-baseline dvou-pohledových geometriích. S pomocí těchto nových testovacích protokolů jsme prozkoumali několik nejdůležitějších parametrů algoritmů pro detekci zájmových oblastí.

Navrhli jsme nový algoritmus pro stavbu pyramid prostoru měřítek, jenž zvyšuje opakovatelnost detektorů v případě znalosti Gaussovského jádra, kterým byl obrázek rozmazán. Na systému pro vyhledávání obrazů jsme ukázali, že při zahrnutí zájmových oblastí s nižším kontrastem, lze významněji zvýšit jejich přesnost, než při zahrnutí oblastí s menší velikostí. Ukazujeme ale, že toto neplatí pro případy použití, kde je upřednostňovaná geometrická přesnost detekce. Na případě široce používaného SIFT deskriptoru ukazujeme, že pokud data, na nichž je počítán, neobsahují významné rotace, není potřeba vstupní data deskriptoru vážit Gaussovským jádrem. S těmito protokoly jsme také změřili výkonnost detektorů a deskriptorů z hlediska velikosti oblasti, která je použita pro výpočet deskriptoru. Ukazuje se, že pro detektory isotropních oblastí se vždy dosáhne lepších výsledků, pokud je do výpočtu deskriptoru zahrnuto více kontextu detekované oblasti.

V poslední části této práce navrhujeme vylepšení emulátorů detektorů zájmových oblastí, které dovoluje detekovat nové typy zájmových oblastí a zvyšuje jejich geometrickou přesnost. Také jsme významně zvýšili efektivnost těchto emulátorů tak, že dosahují vyšší rychlosti detekce, než z hlediska rychlosti pečlivě navržený OpenSURF detektor.

.

Contents

Chapter 1

Introduction

In the field of computer vision, local image feature detection is an intermediate step of several algorithms. It is used for representing an image by a set of well defined and well localised image structures with an informative neighbourhood. Typically some set of measurements are taken from the feature neighbourhood to form a descriptor. Image features and their descriptors are used in many applications e.g. in multiple view geometry (e.g. image stitching, structure from motion and 3D reconstruction), object recognition and image retrieval.

A local image feature is an image pattern which differs from its immediate neighbourhood. They are interesting because they can provide a limited set of well localised and individually identifiable anchor points. What they represent is not relevant so far as their location can be determined accurately and in a stable manner over time. In multiple view geometry what is important is their location (centre) as they are used for estimation of the scene model. In other cases the set of local features and their descriptors can be used as a robust image representation that allows to recognise objects and scenes without a need for segmentation. In this case they do not have to be localised precisely since to analyse their statistics is more important [41].

A detector is a tool that extracts features from an image (usually corners, blobs etc.). An ideal extracted feature should have several properties such as repeatability, distinctiveness (informativeness) and precise localisation. However, these properties change with the detector parameters not only depending on the type of features which it extracts, but also on a particular implementation. This puts us in a difficult task how to properly measure these properties. Existing and widely used performance evaluation protocols are limited to test detectors only on planar scenes which does not address problems such as occlusion or depth discontinuities. Also it does not directly address the properties required for image retrieval where the feature localisation is not so important. This motivates our first goal which is to make it easier to test any feature detector or descriptor algorithm not only with existing tests but also with newly proposed benchmarks which would confront other use cases of these algorithms. With this evaluation framework we would like to examine performance of feature detectors and subsequently their descriptors in order to set their parameters depending on their particular use.

Next goal of this work is to examine emulated feature detectors proposed in [38]. These emulators offer a possibility to speed up several feature detector algorithms. In the original article [37] they had been tested only with benchmarks consisting of planar scenes experiments. Thus they are tested in the proposed new benchmarks in order to asses their performance in retrieval and epipolar geometry tasks. Subsequently we train emulators of different local image features and examine their properties.

1.1 Main contributions

One of the contribution is participation in VLBenchmarks framework which is an auxiliary suite of tests for VLFeat computer vision library algorithms. We have finalised implementation of existing evaluation protocols and proposed new evaluation protocols such as the Epipolar geometry test and image retrieval benchmark. These tests has been implemented under supervision of Andrea Vedaldi in Visual Geometry Group, University of Oxford during May and August 2012. The VLBenchmarks project was presented as a part of a tutorial at ECCV 2012 in Firenze. Using VLBenchmarks software we have tested several properties of existing detectors.

We propose a new algorithm to build a pyramid Gaussian scale space which allows to take into account any nominal Gaussian blur in a processed image. With a-priory knowledge of this blur it is able to increase repeatability of scale space local feature detectors significantly.

A new method to control number of detected features has been proposed by setting lower initial scale. Employing VLBenchmarks tests we have found that for geometry precision tasks it is better to increase the number of detections by detecting smaller features with lower initial scale. However for retrieval tasks, features with lower response function threshold are more advantageous than features with small scale. We have shown, that for some retrieval tasks, where the scale of queried and database instances is similar, it is possible to decrease the number of features in the database as the smallest features has got little impact to retrieval system performance.

Last parameter examined is measurement region of a local feature used for descriptor calculation. We show that for blob detectors, bigger measurement region generally improves their performance mainly for planar scenes. This however does not hold for Harris and MSER detector where the ideal measurement scale is limited. Also we have shown that for input data without significant rotations descriptors are more distinctive without weighting.

In the last part we train new emulated detectors using different local image features than in the original article. We have shown that it is also possible to train detectors with anisotropic features however with the cost of worse rotation invariance. Using VLBenchmarks, newly trained DoG emulated detector gains better geometric precision than the original Hessian-Laplace emulator. Besides that we have improved the classification speed so that the emulators achieve faster evaluation than SURF detector.

1.2 Structure of this thesis

In Chapter 2 we describe existing feature detection algorithms and their emulators. In the next part, Chapter 3, existing evaluation protocols for feature detectors and descriptors are described. Then we follow with description of our work on VLBenchmarks project in Chapter 4. Using this evaluation framework we examine detectors parameters in Chapter 5. In Chapter 6 we follow with description of improvements and tests performed with emulated detectors. In the last part, Chapter 7, we summarise the results.

Chapter 2

Local image features detection and description

In this chapter we describe state of the art algorithms for local feature detection and description. Local image features can be divided based on the invariance to image transformations or by the feature type. The major types of local features are *blobs* (LoG, DoG and Hessian), *corners* (Harris) and *regions* (MSER). In the first section, we describe classes of invariance to geometric transformations. Then we follow with a detailed description of feature detectors which use scale space extrema of second order derivative operators as a tool to obtain both feature location and scale. Then other feature detectors and main algorithms for feature description are briefly described. The local image feature detection can also be seen as a decision process, we will show that it is possible to use machine learning techniques to emulate it. The underlying theory for boosting and emulated detectors is described in the last part of this chapter.

Notes on terminology In the text we use several terms for a local image feature. This ambiguity arises from their different use. For some applications (3D reconstruction or camera calibration) the spatial extent of the local image feature is not important and only the feature location is used. The term *interest point* is then often used. In most applications, where the regions has to be described, such that they can be identified and matched, one typically uses term *region* instead of interest point [41, p. 182]. In this work we use the term *local feature* or term *frame*, which will be described below.

2.1 Local image features and their invariance

Local image features are deterministic local image statistics (features) robust to several types of image distortion factors such as viewpoint change or change of illumination [42], and which differ from their immediate neighbourhood. By definition they are well localised to a compact subset of an image which will be further referred as a region of the feature. The desired property of the feature is an invariance to a particular parameter or function applied to the original image domain.

Local features are detected with *covariant local image feature detectors* (detectors) which select a number of distinctive image regions from the input image. The detector is covariant with a particular family of transformations when the shape of detected image regions does change with the image transformation [41, p. 181]. One way to obtain invariance of the local features comes from the geometric, photometric normalisation [41] of the detected image regions. The idea is that once the local image regions that were found under the image distortion are normalised into a canonical shape, any measurement on the normalised regions are invariant to the deformation. The image regions obtained by the local feature detectors

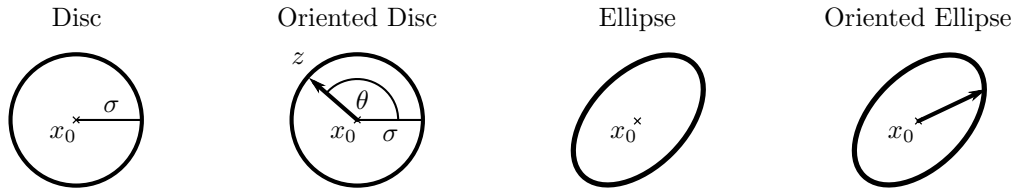


Figure 2.1: Classes of the local image frames.

Frame class	**Attributes**	**Covariant to**
Disc	Frame coordinates $\mathbf{x}_0$, scale (radius) σ	Translations and scaling
Oriented Disc	Frame coordinates $\mathbf{x}_0$, scale (radius) σ and single point $\mathbf{z}$ which defines orientation θ	Translations, scaling and rotation (similarities)
Ellipse	Frame coordinates $\mathbf{x}_0$ and shape matrix Σ with three degrees of freedom	Translation and affinities up to residual rotations
Oriented Ellipse	Frame coordinates $\mathbf{x}_0$ and affine transformation A	Translation and affinities.

Table 2.1: Attributes of image frame invariance classes.

are represented by geometric entities denoted as *frames*. A frame is a subset of the image, defined by a set of attributes that uniquely specifies particular geometric shape, for example disc or ellipse. A class of frames are frames that can be specified by the same set of attributes.

The overview of frequently used classes of local image feature frames is given in table 2.1 and visualised together with their attributes in Figure 2.1. This categorisation and description of the frames is based on [42]. Let us now describe the properties of these frame classes:

Disc Compact subset of the image, $\Omega = \{|\mathbf{x} - \mathbf{x}_0| < r\}$, represents a circular region in the image.

Oriented disc Set Ω (defined as above) and a point $\mathbf{z}$ which defines the major orientation of the circular region. This major orientation can be also expressed by an angle θ, which is the angle of the the line $\mathbf{x}_0 \times \mathbf{z}$.

Ellipse Frames defined by their centre $\mathbf{x}_0$ and the moment of inertia (covariance) matrix (also further referred as second moment matrix).

$$\Sigma = \frac{1}{\int_\Omega 1 \, d\mathbf{x}} \int_\Omega (\mathbf{x} - \mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)^T d\mathbf{x} \tag{2.1}$$

The covariance matrix is positive semi-definite 2×2 matrix that can be visualised by an ellipse [13] as:

$$(\mathbf{x} - \mathbf{x}_0)^T \, \Sigma^{-1} \, (\mathbf{x} - \mathbf{x}_0) = 1 \tag{2.2}$$

Ellipse behaves similarly as a covariance matrix of image region under the affine transformations and is useful for intuitive prediction of the feature frame behaviour. This also is a reason why we further refer to the covariance matrix as an ellipse matrix. For details and a proof see [13, p. 5].

Oriented ellipse Oriented ellipses are defined by an affine transformation A which transforms an oriented ellipse Ω_C into an oriented unit disc $A\Omega = \Omega_C$ with orientation $\theta = 0$ uniquely. This affine transformation is also further referred as de-normalisation matrix.

The local feature detector is an algorithm that finds a mapping F from image domain $I(\mathbf{x})$ to attributed frames domain Φ, where the the detections are uniquely described by the values of frame attributes.

2.2 Detection of local image features with scale space

The scale space based feature detectors search for a local feature in *spatial* and *scale* domains of an image using some "distinctiveness" measurement (either a cornerness or a blob measure, further noted as a *response function*). This is generally a search in a three dimensional space, where the third dimension can be seen as a sampling of spatial frequency domain. The feature detection can be divided into three steps:

1. **The scale space extrema localisation** Search of local extrema in the scale space of the image response function.

2. **Location refinement** Sub-pixel localisation of the frame centroid and scale based on a local optimisation.

3. **Additional measurements** that improves covariance of the detector. The image regions obtained as local extrema in the scale space are uniquely determined by a disc frame attributes $(\mathbf{x}, \mathbf{y}, \sigma)$. The disc frame can be "upgraded" by further examination of the image region to an oriented disc, ellipse or oriented ellipse. The ellipse frame can be estimated by using Baumberg iteration [4]. The oriented frames can be obtained by computing the dominant orientation of the image gradients in the normalized coordinate frame [24].

2.2.1 Gaussian scale space properties

The Gaussian scale space is a multi-scale representation of the original image computed by consecutive filtering of the input image. The consecutive application of a low pass Gaussian filter allows to analyse lower resolutions of the image. The choice of the Gaussian filter among the low pass filters is not random, it has been shown [21] that the Gaussian scale space is a natural solution to a diffusion equation. The Gaussian scale space also holds several important properties, as *shift invariance*, *scale invariance*, *rotational symmetry* and mainly *non-creation of local extrema or zero-crossings in the one-dimensional case* (non-enhancement property)[23, p. 3].

The Gaussian scale space is a continuous function $L(\mathbf{x}, \sigma)$ of some input image $I(\mathbf{x})$ defined as:

$$L(\mathbf{x}, \sigma) = G(\mathbf{x}, \sigma) * I(\mathbf{x}) \tag{2.3}$$

where $*$ denotes a convolution of the input image and the Gaussian kernel:

$$G(\mathbf{x}, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\mathbf{x}^T\mathbf{x}/2\sigma^2} \tag{2.4}$$

The local feature detectors based on the Gaussian scale space use properties of second order derivative operators. The amplitude of spatial derivatives

$$L_{x^i y^j}(\cdot, \sigma) = \partial_{x^i y^j} L(\cdot, \sigma) = G_{x^i y^j}(\cdot, \sigma) * I(\cdot) \tag{2.5}$$

generally *decrease with scale* due to non-enhancing of local extrema property of the Gaussian kernel. In order to be able to compare the derivatives across scales, the derivative operator responses across the scales, the amplitude of Gaussian derivatives must be normalised by a factor [22]:

$$\hat{L}_{x^i y^j}(\cdot, \sigma) = \sigma^{(i+j)\gamma} L(\cdot, \sigma) \tag{2.6}$$

Where in the most cases ([24], [27]) the parameter γ is selected as $\gamma = 1$.

2.2.2 The scale space implementation

The scale space is by definition a continuous function, however the input image is not of infinite resolution. Therefore, a discrete approximation has to be made. The input image is assumed to be pre-smoothed in the discretisation process by a Gaussian low pass filter with some nominal standard deviation $\sigma_n = 0.5$. Due to computation efficiency, the scale space is sampled into a finite number of scales by cascade convolution of the image using the Gaussian kernel. This is possible thanks to the following property of the Gaussian kernel:

$$G(\mathbf{x}, \sigma_2) * G(\mathbf{x}, \sigma_1) * I(x, y) = G\left(\mathbf{x}, \sqrt{\sigma_1^2 + \sigma_2^2}\right) * I(x, y) \tag{2.7}$$

This means that for computation of each successive layer of the scale space we can reuse the previous layer. The scale space construction is made even faster by constructing a pyramid where the scale space is divided into octaves v such that each following octave doubles the scale of previous one. Scales are then sampled exponentially, such that after each octave, the scale doubles:

$$\sigma = \sigma_0 \; 2^{v + \frac{s}{S}} \quad s = 0, ..., S-1 \quad v = 0, ..., v_{max} \tag{2.8}$$

Parameter $\sigma_0 = 1.6$ is a scale of a pyramid base and was empirically set in [24, p. 10] and represents the sampling frequency in the image domain ("given that extrema can be arbitrary close together, there will be a similar trade-off between sampling frequency and rate of detection" [24, p. 10]). The maximum number of octaves v_{max} is set to a value where the longer size of the image is still bigger than 2^3. Each octave is down-sampled by factor 2.

2.2.3 Local image feature localisation

Local features are detected as *local extrema* of some response function $\text{Resp}(\mathbf{x}, \sigma_D)$. The size of the searched local neighbourhood is usually $3 \times 3 \times 3$. The definition of the response function differs for each detector. The most widely used are:

Determinant of Hessian. For detecting distinctive features, the determinant of Hessian [29] matrix measure is used. The Hessian matrix is defined as:

$$H(\mathbf{x}, \sigma_D) = \begin{pmatrix} L_{xx}(\mathbf{x}, \sigma_D) & L_{xy}(\mathbf{x}, \sigma_D) \\ L_{xy}(\mathbf{x}, \sigma_D) & L_{yy}(\mathbf{x}, \sigma_D) \end{pmatrix}, \tag{2.9}$$

where values $L_{ab}(\mathbf{x}, \sigma_D)$ are second order Gaussian derivatives of scale space values $L(\mathbf{x}, \sigma_D)$ (σ_D is differential scale, also called local scale). The measure itself is defined as a determinant of the Hessian matrix:

$$\text{Resp}_{Hess}(\mathbf{x}, \sigma_D) = \sigma_D^2 |H(\mathbf{x}, \sigma_D)|, \tag{2.10}$$

factor σ_D^2 is the normalisation factor of the second derivatives, in order to maintain scale invariance of the response function across the scale levels. The determinant of Hessian response can be computed efficiently by computing symmetric differences in a 3×3 window around each pixel. The response function has three types of extrema which correspond to the following types of Hessian features:

a) **Bright blob** when $|H(\mathbf{x}, \sigma_D)| > 0$ and $L_{xx} < 0$. This feature correspond to "hill top" blobs in image brightness.

b) **Dark blob** when $|H(\mathbf{x}, \sigma_D)| > 0$ and $L_{xx} > 0$. This feature correspond to "valley" blobs in image brightness.

c) **Saddle point** when $|H(\mathbf{x}, \sigma_D)| < 0$ and $L_{xx} > 0$.

Laplacian of Gaussian (LoG). The Laplacian of Gaussian response for blob detection is defined as the trace of the Hessian matrix:

$$\mathrm{Resp}_{LoG}(\mathbf{x}, \sigma_D) = \sigma_D \mathrm{trace}(H(\mathbf{x}, \sigma_D)) \tag{2.11}$$

And the features can be of type:

a) **Bright blob** when $\mathrm{Resp}_{LoG}(\mathbf{x}, \sigma_D) < 0$

b) **Dark blob** when $\mathrm{Resp}_{LoG}(\mathbf{x}, \sigma_D) > 0$

Difference of Gaussian (DoG). The difference of Gaussian [24] is an approximation of Laplacian of Gaussian. The advantage of this response is that, in the scale space, it is computed without the convolution simply by subtracting the successive levels of scale-space:

$$\mathrm{Resp}_{DoG}(\mathbf{x}, \sigma_D) = \frac{\sigma_D}{\Delta} \left(L(\mathbf{x}, \sigma_D + \Delta) - L(\mathbf{x}, \sigma_D - \Delta)\right) \tag{2.12}$$

Feature types are categorised in the same way as for the LoG response.

Scale adapted Harris response (Harris). The response function of the Harris corner detector [27] is based on the *second moment matrix* of image gradients, also called an auto-correlation matrix. It is defined as:

$$M(\mathbf{x}, \sigma_D, \sigma_I) = \sigma_D^2 G(\mathbf{x}, \sigma_D) * \begin{pmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x(\mathbf{x}, \sigma_D) L_y(\mathbf{x}, \sigma_D) \\ L_x(\mathbf{x}, \sigma_D) L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{pmatrix} \tag{2.13}$$

where L_x are first order Gaussian derivatives with derivation scale σ_D. The outer products of the gradients are averaged in the point neighbourhood with a Gaussian window of scale σ_I (integration scale). The eigenvalues of this matrix represent the principal changes of image intensities in two orthogonal directions. Their ratio is effectively computed with the Harris corner measure:

$$\mathrm{Resp}_{Harr}(\mathbf{x}, \sigma_D) = |M(\mathbf{x}, \sigma_D, \sigma_I)| - \lambda \ \mathrm{trace}(M(\mathbf{x}, \sigma_D, \sigma_I)), \tag{2.14}$$

with typically $\lambda = 0.04$ and $\sigma_I = \sqrt{2}\ \sigma_D$ (in case of CMP implementation). Only response function maxima are considered by the detector.

For scale estimation, the Laplace operator is usually used in a hybrid detector. Hybrid detectors are described below.

Sub-pixel/Sub-scale localisation

At higher levels in the scale space pyramid, it is important to localise the feature more precisely as there the pixels correspond to large areas relative to the base image [6]. It is supposed that the response function is smooth enough around the extrema that it can be approximated with a 3D quadratic function. The 3D quadratic function is then fitted into a $3 \times 3 \times 3$ neighbourhood of the local extrema and its peak is taken as a sub-pixel and sub-scale location [6]. The function is expressed in a Taylor expansion (up to quadratic elements) of the scale space function $\mathrm{Resp}(\mathbf{x}, \sigma)$ with the origin in a location of the scale space extremum.

$$\mathrm{Resp}(\mathbf{z}) = \mathrm{Resp} + \frac{\partial \mathrm{Resp}}{\partial \mathbf{z}}^T \mathbf{z} + \frac{1}{2} \mathbf{z}^T \frac{\partial^2 \mathrm{Resp}}{\partial \mathbf{z}^2} \mathbf{z} \tag{2.15}$$

Where vector $\mathbf{z} = (\mathbf{x}', \sigma')^T$ is the offset from the localised point. The location of the extremum of this function $\hat{\mathbf{z}}$, i.e. our solution offset can be obtained as:

$$\hat{\mathbf{z}} = \frac{\partial^2 \mathrm{Resp}}{\partial \mathbf{z}^2}^{-1} \frac{\partial \mathrm{Resp}}{\partial \mathbf{z}} \tag{2.16}$$

This is an easily solvable 3×3 linear system. If the offset is bigger than $\hat{\mathbf{z}} > 0.6$, it means that the extremum lies closer to a neighbouring point, so the detected location is changed and the optimisation is run again in an iterative procedure (in our case limited to maximum of five steps) until the location is found stable. The final offset $\hat{\mathbf{z}}$ is added to the feature location.

2.2.4 Low contrast and edge regions suppression

The second order derivatives based detectors fire on the edges, where at least one of the second derivatives is high. To eliminate the edge responses which tend to be unstable under geometric transformations[1], the Hessian matrix H of the response function is used. Its eigen-values $(\lambda_{max}, \lambda_{min})$ are proportional to the principal curvatures of the response function. Because only the ratio between the eigen-values $r = \lambda_{max}/\lambda_{min}$ is important, it can be calculated as comparison of the trace and determinant of the Hessian matrix:

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} = \frac{(\lambda_{max} + \lambda_{min})^2}{\lambda_{max}\lambda_{min}} = \frac{(r+1)^2}{r} \tag{2.17}$$

and only regions which fulfil the following condition

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} < \frac{(r+1)^2}{r} \tag{2.18}$$

are preserved [24].

Also in order to suppress regions with low contrast, only regions with response higher than a *response function threshold* R_t (also called peak threshold) are preserved.

2.2.5 Estimation of feature orientation

The local features detected in the scale space are assigned with a location and scale, and represented by a disc frame that does not provide information about feature orientation. However, the feature can be upgraded to an oriented disc to increase the invariance of the descriptors computed on the detection. One of the ways how to find a robust orientation of the feature is to exploit local gradients in the vicinity of the feature.

The gradient orientations $\theta(\mathbf{x})$ and magnitudes $m(\mathbf{x})$ in a circular region with radius $r_{\mathrm{win}} = 1.5\sigma$ are computed (VLFeat SIFT[2]) at the closest scale level of the pyramid corresponding to the detected scale of the feature. Then, the gradient orientation are weighted by gradient magnitudes and collected in an orientation histogram additionally weighted by the gradient magnitudes. Gradient magnitudes are weighted by a Gaussian kernel with $\sigma = 1/3\ r_{\mathrm{win}}$ in order to emphasise gradients closer to local feature's centre. The number of bins of the histogram is usually set to 36.

After smoothing the histogram values, in order to remove coarse peaks, local peaks within 80% of the global maxima are returned as the detected orientations. The maximum number of extrema is usually limited to 4. In case the full affine invariance (oriented ellipse) is desired, the orientation is computed on patch normalised to a unit disc, to allow affine covariant measurements of the gradient orientations.

2.2.6 Hybrid detectors

We denote scale-space based detectors as hybrid, when the response function in the spatial domain differs from the response function used to find the scale of the feature. The hybrid detectors proceeds as folloe. First, a spatial local feature location is found using a non-maxima-suppression in the 3×3 neighbourhood on each scale level independently. Then, the best scale is found an extremum of the second measure over all scales. Common combinations of response functions used in the literature are *Harris-Laplace* [27] or *Hessian-Laplace* [29] mainly for the good localisation of the corner-like features by the Harris response function.

2.2.7 Affine invariance

Baumberg [4] has shown, that a disc frame can be extended to an ellipse frame by an algorithm called *Baumberg iteration*. This algorithm iteratively estimates the affine distortion of the

[1] Location on the edge is well defined only in the direction orthogonal to the edge.

[2] `http://www.vlfeat.org/api/sift.html#sift-tech-detector-orientation`

gradients of the detected feature by computing a structure tensor with the second moment matrix of image gradients, which is used as a measure of anisotropy. In each step, the inverse square root of the second moment matrix is accumulated and the accumulated transformation used as an estimate of affine distortion. The estimated affine transformation is then used to normalise an image patch around the feature. The iteration continues, until sufficient isotropy of the normalised patch is achieved. This iteration removes frames where the iteration does not converge. This usually happens for too elongated features (such as edges). The result of the iteration is an affine transformation that is fixed up to an unknown rotation[3]. This rotation can be fixed as upright [31] (ellipse frame), or dominant orientation can be estimated as described above to form an oriented ellipse. Detectors which use Baumberg iteration are further referred with the *-Affine* suffix.

2.3 Other state of the art detectors

Besides the already introduced scale space detectors plentiful of other feature detectors exist. We present a small selection of the most important ones.

MSER [26] Region detector based on a modified watershed algorithm with different stability criteria. In general it detects regions, however they are usually converted to ellipses where the centre is defined as their centre of gravity and the shape is specified by moments of the detected region interior. The scale of the ellipse is then usually twice the scale defined by the moments. Generally it detects fewer regions but the detected features are naturally affine invariant.

SURF [5] Speed Up Robust Features. The similarity invariant frame detector, which comes with the SURF descriptor, detects frames using an approximation of the Hessian response with box filters. The orientation of the frames is fixed using box filters. In some sense, it is similar to emulated detectors, as the box filters resemble Haar features computed with assistance of integral images. However, the particular filters are carefully engineered and their response is used directly. The scale of the features is detected over a pyramid of box filter size which partly resembles the scale space pyramid and it uses the same non-maxima suppression as the scale space detectors, together with sub-pixel localisation.

FAST [34] A translation and rotation invariant corner detector (no scale invariance) which uses a decision tree to classify a possible feature presence by comparing central pixel to its 12 pixel circular neighbourhood (in this sense it is somehow similar to local binary patterns). The decision tree is learnt using machine learning techniques to achieve decision in the shortest time. In comparison to other detectors, FAST simplicity brings *fast* detection though without scale invariance.

BRISK [20] The detector used in this framework is an extension of the FAST detector with a Gaussian scale space. A classical non-maxima suppression over the FAST score is performed in order to locate the features in the scale dimension. Sub-pixel localisation is also performed.

2.4 Local image feature description

The local feature descriptors provide a description invariant to the image transformation. This can be achieved mainly by two means: either by descriptors invariant to the transformation e.g. rotation, or by descriptors that are not invariant to the transformation, however computed on the neighbourhood normalised using the detected covariant frame. This two approaches can be also combined. The geometric normalization proceeds as follows. At first,

[3] caused by the degree of freedom in the computing of the matrix square root

the detection scale is multiplied by a magnification factor ν in order to obtain more distinctive neighbourhood a feature's *measurement region*. The measurement region of the feature is then normalised (usually rotation or affine shape normalisation) to a small patch P (usually of size 41 pixels) which is then used for calculation of an invariant descriptor.

Descriptors are robust image transformations which characterise the image brightness structures and are invariant to photometric transformations. Let us introduce some of the commonly used local feature descriptors:

SIFT [24] The SIFT descriptor is a 3D spatial histogram capturing the spatial distribution, magnitude and orientation of the image gradients [43]. The image gradients are collected in a 4×4 grid (to preserve spatial information) where each cell is described by a histogram of image gradient orientations inside the cell. Each histogram is expressed by eight values yielding a $4 \times 4 \times 8 = 128$ values long descriptor. Thanks to this coarse spatial division, this descriptor is robust to some errors in local feature position and orientation. Collected values are then quantized to fit into one byte. The similarity measure is the Euclidean distance or Hellinger distance (RootSIFT, [3]).

SURF [5] *SURF* descriptor is calculated with box filter similarly as the SURF features. The spatial and orientation information is stored in a similar way as in SIFT. However the descriptor is only 64 bytes long. The similarity measure used is usually the Euclidean distance.

BRIEF [7] (Binary Robust Independent Elementary Features). A binary vector which consists of results of a certain number of pairwise (or n-wise) intensity comparisons in a smoothed patch. The patch is sampled randomly with a bias towards the patch centre. The advantage of binary descriptors is their small memory footprint as results of each comparison are stored in a single bit. Similarity is computed with Hamming distance, which can be quickly performed with an XOR operator and a processor instruction which counts the number of zero bits.

DAISY [40] This descriptor collects its values by sampling orientation maps (gradients of the patch in particular directions) by averaging values in a small neighbourhood with a Gaussian kernel. Samples are taken on a circular pattern where the size of the averaged neighbourhood increases with the circle radius. This creates a Daisy-like structure. This descriptor was developed mainly for dense stereo matching. In [48], machine learning techniques are used for learning to pick the best samples with a goal to form a compact and distinctive descriptor.

LIOP [47] Instead of gradients, LIOP collects histograms of pixel orderings. First, it divides image brightness values into several bins. In each bin, N pixel samples are taken on a circular pattern around the centre of the feature and pixel intensities ordering are collected into a histogram of $N!$ bins. Concatenated ordering histograms over all bins form the LIOP descriptor. The similarity measure used is Euclidean distance, however as it is formed by histograms, Hellinger distance may be used as well.

MROGH [10] Extension of LIOP descriptors where several LIOPs over different measurement regions are collected into a single descriptor.

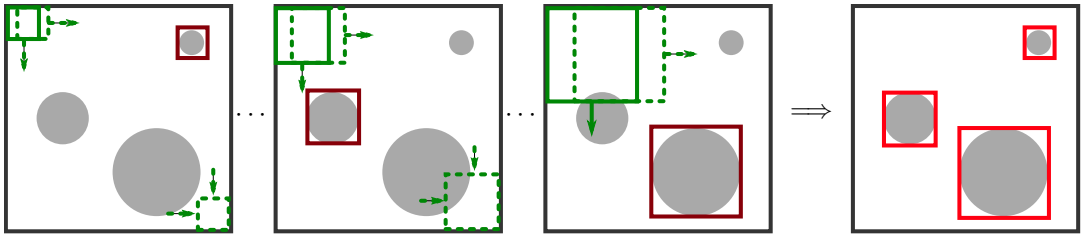


Figure 2.2: The multi-scale sliding window approach to detection. A detection window is swept through the image and an object vs. background classifier is evaluated at each position and scale.

2.5 Emulated local feature detectors

The concept of emulated local image feature detectors has been introduced in [38]. It is based on WaldBoost learning framework [37], applied to detection of the local image features. The task is to learn a sequential classifier which is able to decide whether a given image sample contains an image feature or not.

The key idea is, that some existing feature detection algorithm is used as a black box algorithm performing a binary decision task. Running this algorithm on a big set of images it provides a large amount of positive and negative samples for training of the classifier. Then the classifier is used to *emulate* the black box algorithm of the feature detector. It detects features in the image by classifying all rectangular regions (patches) of various sizes and in various positions, by using a multi-scale sliding window technique. The sliding window approach is illustrated in Figure 2.2.

Considering the size of the image and the number of possible scales, the sliding window technique produces ample amount of patches which need to be classified. This task was first solved in real time by Viola and Jones [44]. Based on the extensive research in the field of real-time image detectors the emulated feature detectors use a Waldboost algorithm [37] to emulate the Hessian-Laplace [27] and Kadir-Brady saliency detector [16]. The main advantage of the emulated detectors is their speed, as the WaldBoost sequential classifier is able to eliminate an ample amount of background patches by early decision.

In the following, we describe some basics about the WaldBoost sequential classifier and RealBoost used for selection of the weak classifiers. Then, we continue with details related to emulated local image feature detectors.

2.5.1 WaldBoost

WaldBoost is a meta algorithm, an extension of AdaBoost meta-algorithm [12], which allows to make an early decision and therefore speed-up the classification. But instead of using a cascade of separate strong classifiers as Viola and Jones [44], the number of measurements needed is decided using a sequential probability ratio test.

Sequential probability ratio test *Sequential probability ratio test* (SPRT) is a statistical framework developed for quality control in manufacturing proposed by Wald [46]. It is useful for decision making when the number of samples is not known in advance. By continuously collecting the data, the decision is postponed until sufficient information is available. The advantage of this approach is that the decision can be made much earlier than in the case of non-sequential, monotonic, algorithms.

Let x be an object with hidden state (class, label) $y \in \{-1, +1\}$ which is not directly observable. The hidden state is determined based on successive measurements $x_1, x_2, \ldots$ and the knowledge of joint conditional probabilities $p(x_1, \ldots, x_t | y = c)$ of the sequence of measurements $x_1, \ldots, x_t$ for $x \in \{-1, +1\}$ and for all t. SPRT sequential strategy S^* is

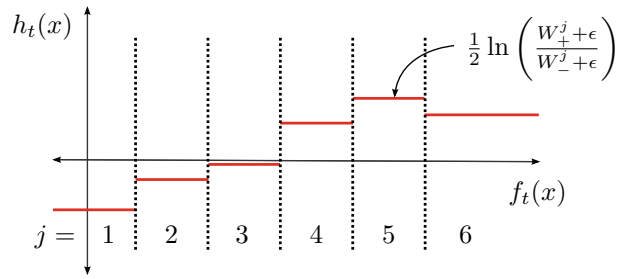


Figure 2.3: The domain-partitioning weak classifier, also known as RealBoost weak classifier. The response of features $f(x)$ on a sample x is partitioned into bins $j = 1, \ldots, K$. The left-most and right-most bins cover respective half-spaces. In each bin j, the response of the weak classifier $h(x)$ is learnt from the sum of positive (W_+^j) and negative (W_-^j) weights falling into the bin. The sample weights are set according to AdaBoost learning strategy [12]. To avoid numerical problems, a smoothing factor ϵ is used [35]. Adopted from [38].

defined as [46]:

$$S_t^* = \begin{cases} +1 & \text{if } R_t \leq B \\ -1 & \text{if } R_t \geq A \\ \# & \text{if } B < R_m < A \end{cases} \tag{2.19}$$

The symbol # stands for "continue", i.e. do not decide yet. R_t is defined as a likelihood ratio:

$$R_t = \frac{p(x_1, \ldots, x_t | y = -1)}{p(x_1, \ldots, x_t | y = +1)} \tag{2.20}$$

And the constants A and B are set accordingly depending on the required probability of error of first kind α (x belongs to $+1$ but is classified as -1) and probability of error of the second kind β (x belongs to -1 but is classified as $+1$). It is generally difficult to estimate the values of A and B. Wald [46] suggests to set them to their upper and lower bounds respectively:

$$A' = \frac{1-\beta}{\alpha}, \qquad B' = \frac{\beta}{1-\alpha} \tag{2.21}$$

Following this decision, in [37] the t-dimensional space is projected into a one dimensional space by a boosted strong classifier function H_t of length t and the likelihood ratio is estimated as:

$$\hat{R}_t = \frac{p(H_t(x) | y = -1)}{p(H_t(x) | y = +1)} \tag{2.22}$$

Moreover, Šochman and Matas [37, p. 4] shows that this projection sufficiently approximates the $\hat{R}_t$. The strong classifier of length T is defined as:

$$H_T(x) = \sum_{t=1}^{T} h_t(x) \tag{2.23}$$

Where $h_t(x)$ are responses of the weak classifiers.

Weak classifier selection The RealBoost domain partitioning weak classifiers h_t [35] are used, each one based on a single (visual) feature f_t (see Figure 2.3).

The input to the WaldBoost learning algorithm is a pool of samples $\mathcal{P}$, a set of features $\mathcal{F}$ and the bounds on the final false negative rate α and false positive weight β. In the learning, the selection of the weak classifier is by far the most time consuming operation. Therefore only a subset $\mathcal{T}$ of the samples pool $\mathcal{P}$ is used for selection of the weak classifier. In each round, the already decidable samples in the pool are removed from the learning process and new training set $\mathcal{T}$ is sampled. This process is called Bootstrapping. Details about the learning with bootstrapping are shown in Algorithm 1.

Algorithm 1 WaldBoost learning with bootstrapping, adopted from [38]

Input: Sample pool $\mathcal{P} = \{(x_1, y_1), \ldots, (x_m, y_m)\}; x_i \in \mathcal{X}, y_i \in \{-1, +1\}$, set of features $\mathcal{F} = \{f_s\}$ maximal false negative rate α and false positive rate β, number of weak classifiers T

1: Sample randomly the initial training set $\mathcal{T}$ from the pool $\mathcal{P}$
2: Set $A = (1 - \beta)/\alpha$ and $B = \beta/(1 - \alpha)$
3: **for** $t = 1, \ldots, T$ **do**
4: Choose best weak classifier h_t by RealBoost [35] using $\mathcal{F}$ and $\mathcal{T}$ and add it to the strong classifier H_t. See Figure 2.3.
5: Estimate the likelihood ratio according to eq. 2.22
6: Find thresholds $\theta_A^{(t)}$ and $\theta_B^{(t)}$ for H_t based on A, B
7: Bootstrap: Throw away samples from training set for which $H_t \geq \theta_B^{(t)}$ or $H_t \leq \theta_A^{(t)}$ and sample new samples into the training set $\mathcal{T}$ using QWS+ [18]
8: **end for**
return strong classifier H_t and thresholds $\theta_A^{(T)}$ and $\theta_B^{(T)}$.

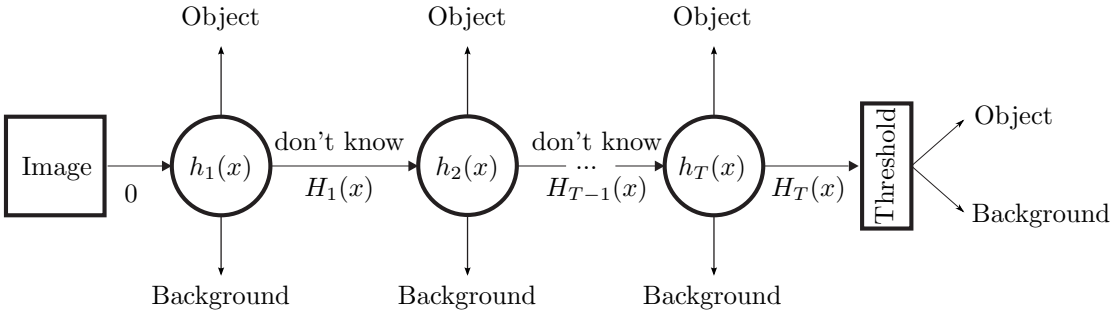


Figure 2.4: Pipeline of sequential binary classifier. Adopted from [17, p. 13].

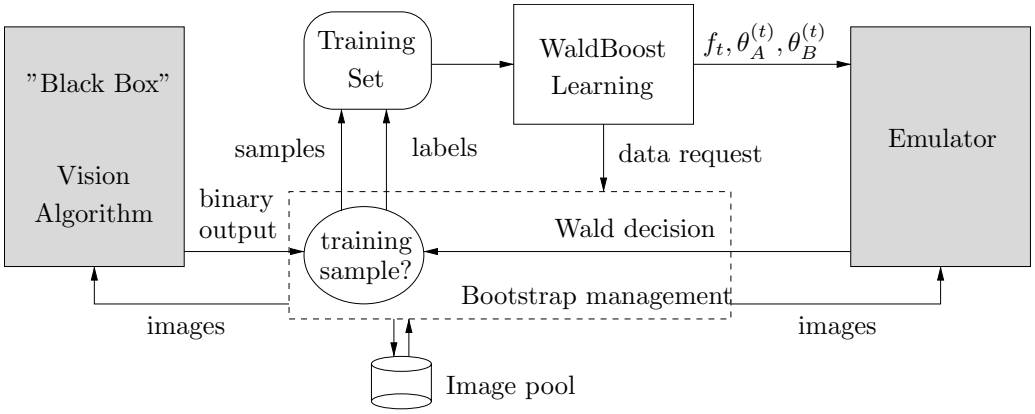


Figure 2.5: Learning scheme of feature emulator [38].

.

Sequential binary classifier The sequential binary classifier, as defined in [37], consists of a set of weak classifiers as illustrated in Figure 2.4. A sequential classifier, denoted as S_t, sequentially computes responses of weak classifiers $h_1(x) \dots h_t(x)$ on input image patch x. The responses $h_t(x)$ are summed up to the final response $H_T(x)$, the strong WaldBoost classifier. The response $H_T(x)$ is then compared to the corresponding thresholds and it is either classified positive or negative, or the next weak classifier is evaluated [38]. The difference from the traditional AdaBoost strong classifier is that in any step, based on the collected response, the final decision can be made. The sequential strategy is:

$$S_t(x) = \begin{cases} +1 & \text{if } H_t(x) \geq \theta_B^{(t)} \\ -1 & \text{if } H_t(x) \leq \theta_A^{(t)} \\ \# & \text{if otherwise} \end{cases} \tag{2.24}$$

If a sample x is not classified even after evaluation of the last weak classifier, a user defined threshold γ is imposed on the real-valued response $H_T(x)$ [38].

2.5.2 Feature detector emulators

In the scheme proposed in [38], the existing local feature detection algorithms are approached as black-box algorithms which perform a binary decision task. A scheme of the learning process is shown in Figure 2.5. Negative samples are any image patches which are not detected as local image features by the detector. For the emulators, the set $\mathcal{F}$ includes Haar-like features proposed by Viola and Jones [44], plus a centre surround Haar-like features which has been shown to be useful in blob detection. An advantage of Haar features is that they can be efficiently evaluated using a integral image.

In the original articles, Hessian Laplace [27] and Kadir-Brady saliency [16] detectors are emulated. For learning, same Haar features as in [44] are used together with centre surrounded features which are useful for blob detection. Also, contrary to [37], feature responses are not normalised by the window standard deviation since the image brightness contrast is important for local image features. The classifier length is set to $T = 20$ as longer classifiers slow down the evaluation and do not bring significant improvement in the performance [38, p. 155]. In all experiments performed in [38], $|\mathcal{T}| = 10,000$, half positive and half negative samples.

Non maxima suppression An essential part of the detector is *non-maxima suppression* algorithm. Here, instead of having a real-valued feature response over whole image, sparse responses are returned by the WaldBoost detector. The accepted positions get the real-valued confidence value $H_T(x)$, but the rejected positions have the "confidence" around $\theta_A^{(t)}$, which makes these values incomparable, thus classical local non-maxima search cannot be

used [38, p. 153]. Instead non-maxima suppression based on the overlap of the detected local features is used. Non-maxima suppression in the emulated detector is based on joining overlapped regions where the region with the highest response is preserved. This demands precise confidence values for the accepted patches. Thus, the same asymmetric version of WaldBoost as used in [37], i.e. setting $\beta = 0$. This is also motivated by fact that the error of the first kind (missed local image features) is considered as more serious than the error of the second kind (falsely detected feature). The decision strategy becomes:

$$S_m(x) = \begin{cases} -1 & \text{if } H_t(x) \leq \theta_A^{(t)} \\ \# & \text{if } \theta_A^{(t)} < H_t(x) \end{cases} \tag{2.25}$$

And samples are classified as positive when the response of the strong classifier $H_T(x) > \gamma$, where γ is user defined parameter, further referred as minimal confidence.

In case of the Hessian-Laplace emulator, the false negative rate α balances between the trade-off of detector speed and precision as increasing α leads to faster evaluation. In [38], α is set to 0.2 as a compromise.

Any two features are joined together if their overlap is higher than a given threshold s (user defined parameter) and the feature with the higher response $H_T(x)$ is preserved. In order to speed-up the computation, an overlap of two circles is approximated with a linear function:

$$o = \frac{r^2}{R^2}\left(1 - \frac{d_c}{r + R}\right) \tag{2.26}$$

Where d_c is distance between the centres of two circles, r the radius of the smaller circle and R the radius of the bigger circle ($r < R$).

Chapter 3

Evaluation of local image feature detectors and descriptors performance

Measuring performance of local image feature detectors usually tests to what extent are detected image regions prone to geometric and photometric transformations [29, p. 15]. The way how the repeatable detections are determined may differ, in some cases it is based purely on comparison of geometric location of the feature region, in other cases it uses feature description as well. The performance of local feature descriptors is assessed using a simple classifier based on similarity measure between the descriptors.

In the first section we introduce feature detectors benchmark based on homography ground truth data. Then follows an overview of descriptor benchmarks which uses the same testing data as homography detectors benchmark. Third section is dedicated to description of *DTU 3D Robot* benchmark which based on 3D model of several scenes measure performance of feature detectors.

3.1 Homography transformation based benchmarks

In this section, we describe an evaluation protocol proposed in [29]. The evaluation protocol is defined together with the datasets that tests detectors under various geometric and photometric image transformations. The geometric transformations are limited to homographies (for homography definition see [14, p. 32]).

The repeatability of local feature detectors is in this benchmark measured by comparing frames $\mathrm{F}(I)$ detected in the reference image I with corresponding frames $\mathrm{F}(J)$ detected in the tested image J. A correspondence is a pair of image frames $(a, b) : a \in \mathrm{F}(I), b \in \mathrm{F}(J)$ established using a known image transformation H that transforms points from image J to image I. The repeatability is then defined by the absolute number of correspondences and by the relative ratio of corresponding features to number of detected features.

The process of measuring repeatability has following steps:

1. Measure the similarity score $\textsc{Sim}(a, b)$ for each pair of image frames $(a, b) \in \mathrm{F}(I) \times \mathrm{F}(J)$.
2. Generate a set of tentative correspondences $\mathcal{T}$ by a matching strategy using the similarity score.
3. Using a known homography H find a subset of corresponding frames $\mathcal{C} \subseteq \mathcal{T}$.
4. Calculate repeatability score as:

$$\textsc{DetRep} = \frac{|\mathcal{C}|}{\min\{|\mathrm{F}(I)|, |\mathrm{F}(J)|\}} \tag{3.1}$$

Where similarity score $\textsc{Sim}(a, b)$ can be based on features' geometric location (repeatability score) or can be defined as features' descriptor similarity measure (matching score). In both cases the features are matched using stable marriage algorithm described below.

3.1.1 Matching strategy

Once the similarity between the frames detected in two images is computed for all pairs, we need to establish a set of tentative correspondences. The ideal matching strategy would find a one-to-one assignment that maximise the overall similarity between the assigned frames.

In order to obtain the solution more quickly, a greedy approximation of the matching is computed with Algorithm 2 which solves stable marriage problem. For certain geometric similarity measures in case of severe misplacement of feature regions it is possible to discard this pair of features from the matching. This is the reason why the Algorithm 2

Algorithm 2 $\mathrm{SM}(E, w_{ab})$, Algorithm for solving stable marriage problem

Input: Edges E, and nodes defined as $U = \{a \mid \exists b, (a, b) \in E\}$, $V = \{b \mid \exists a, (a, b) \in E\}$ which are in a bipartite graph $G = (U, V, E)$ and edge weights function $w : E \to \mathbb{R}$

Output: Matching $M \subseteq E$

Order list $S^* = (e_1, e_2, \dots e_k)$ such that $w(e_i) > w(e_{i+1})$
Label $m_v = \mathrm{FALSE}, \forall v \in V$
Label $m_u = \mathrm{FALSE}, \forall u \in U$
Matching $M = \emptyset$
for all $i = 1 \dots k$ **do**
    $e_i = (u, v)$
    if $m_u = \mathrm{FALSE} \wedge m_v = \mathrm{FALSE}$ **then**
        $m_u = \mathrm{TRUE}$
        $m_v = \mathrm{TRUE}$
        $M = M \cup e_i$
    end if
end for

3.1.2 Repeatability Score

The ground truth transformation used in [29] is a homography H between the reference and testing image such that $I(H\mathbf{x}) = J(\mathbf{x})$. This limits the testing datasets into set of images with viewpoint change of planar surfaces or scenes transformed by rotation of camera around optical centre. Hence it is not possible to measure robustness to occlusion etc.

The similarity score $\textsc{Sim}(a, b)$, used to establish one-to-one correspondences is measured as geometric overlap of an ellipse representation of a frame in the reference image and an ellipse re-projected from the tested image using the homography between the images. Only regions located in the part of scene visible in both images are taken into account. Formally for a frame $a = (\mathbf{x}_a, \Sigma_a), a \in \mathrm{F}(I)$ and $b = (\mathbf{x}_b, \Sigma_b), b \in \mathrm{F}(J)$, the overlap is defined as:

$$\textsc{Overlap}(a, b) = \frac{\Omega_{\Sigma_a, \mathbf{x}_a} \cap \Omega_{H^T \Sigma_b H, H\mathbf{x}_b}}{\Omega_{\Sigma_a, \mathbf{x}_a} \cup \Omega_{H^T \Sigma_b H, H\mathbf{x}_b}} \tag{3.2}$$

Where $\Omega_{\Sigma, \mathbf{x}}$ is an elliptical region with centroid $\mathbf{x}$ and covariance matrix Σ. The regions are represented by ellipses so the local orientation in this measure is always ignored.

The overlap defined in Equation 3.2 is a function of the reference region size, i.e. the frames with bigger scale can exhibit much higher spatial localisation error. In [29], frame scale is normalised such that all frames from tested image are with scale σ:

$$\textsc{NormOverlap}(a, b, \sigma) = \frac{\Omega_{s\Sigma_a, \mathbf{x}_a} \cap \Omega_{sH^T \Sigma_b H, H\mathbf{x}_b}}{\Omega_{s\Sigma_a, \mathbf{x}_a} \cup \Omega_{sH^T \Sigma_b H, H\mathbf{x}_b}}, \quad s = \frac{\sigma^2}{\sqrt{|\Sigma_b|}} \tag{3.3}$$

Usually the normalisation is to scale $\sigma = 30$. However then for small regions this yield correspondences even when the original regions does not intersect each other.

Tentative correspondences $\mathcal{T}_r$ are obtained as:

$$\mathcal{T}_r = \mathrm{SM}(\mathrm{F}(I) \times \mathrm{F}(J),\ \textsc{NormOverlap}) \tag{3.4}$$

For a tentative correspondence $(a, b) \in \mathcal{T}_r$ to be considered a correspondence it must hold:

$$(a, b) \in \mathcal{C}_r \iff (a, b) \in \mathcal{T}_r,\ \ 1 - \textsc{NormOverlap}(a, b) < \epsilon_{0r} \tag{3.5}$$

where ϵ_{0r} is the maximal overlap error of two tentatively corresponding regions. Usually, the overlap error threshold is set to $\epsilon_{0r} = 40\%$. The repeatability score is then computed using Equation 3.1.

To asses repeatability of detectors, it is also important to evaluate the absolute number of correspondences. The detector can simply by covering the whole image with a large number of regions of various sizes improve the repeatability. The ideal approach would be to limit the number of detected regions to a certain number which is however generally hard to achieve for all detectors. For many detectors there is not a single measurement of region quality and also each detector detects different types of features. Graphs with relative and absolute repeatability are given in order to compare the number of regions for each detector.

3.1.3 Matching Score

By definition, local features are regions which are *distinctive*. The matching score estimate the distinctiveness of detected local features by using descriptors. This is closer to practice where features are often accompanied by their descriptors.

The idea is to generate a set of tentative correspondences $\mathcal{T}_d$ using the descriptor similarity measure DSIM, $\mathcal{T}_d$ as follows:

$$\mathcal{T}_m = \mathrm{SM}(\mathrm{F}(I) \times \mathrm{F}(J), \mathrm{DSIM}) \cap \mathcal{T}_r \tag{3.6}$$

Note, that the features must be one-to-one matched not only by their descriptor distances but also by their region overlaps[1]. Because there are no constrains on the minimal similarity between descriptors, the search space cannot be pruned which makes the computation of this measure rather slow.

For SIFT descriptors the similarity measure is negative Euclidean distance:

$$\mathrm{DSIM}(a, b) = -\|\mathrm{D}(a) - \mathrm{D}(b)\| \tag{3.7}$$

Mapping $D : I(\Omega) \to \mathbb{R}^d$ stands for a descriptor of frame a where d is a descriptor length.

Contrary to this definition, in some articles ([38]) the tentative correspondences are computed only as:

$$\mathcal{T}'_m = \mathrm{SM}(\mathrm{F}(I) \times \mathrm{F}(J), \mathrm{DSIM}) \tag{3.8}$$

i.e. that only descriptor distances are matched.

The matching score is based on the region descriptors, thus to compute the frame overlap we use the measurement region which was used for descriptor calculation. We define $\textsc{MROverlap}(a, b, \nu)$, measurement region overlap as:

$$\textsc{MROverlap}(a, b, \nu) = \frac{\Omega_{\Sigma_a \nu^2, \mathbf{x}_a} \cap \Omega_{H^T \Sigma_b \nu^2 H, H\mathbf{x}_b}}{\Omega_{\Sigma_a \nu^2, \mathbf{x}_a} \cup \Omega_{H^T \Sigma_b \nu^2 H, H\mathbf{x}_b}} \tag{3.9}$$

Usually the measurement regions size is set to $\nu = 3$. The threshold on maximal overlap error is set to constant $\epsilon_{0m} = 50\%$ and the correspondences are obtained as:

$$(a, b) \in \mathcal{C}_m \iff (a, b) \in \mathcal{T}_m,\ 1 - \textsc{MROverlap}(a, b, \nu) < \epsilon_{0m} \tag{3.10}$$

Matching score is then computed using Equation 3.1.

[1] This fact is not directly mentioned in the original article, however we have observed this when we have tried to reproduce the results from this article.

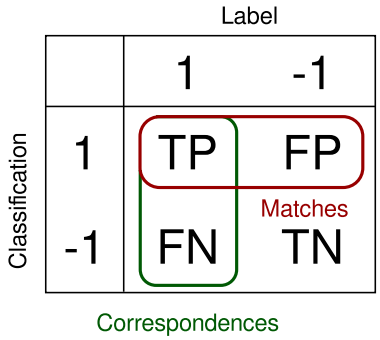


Figure 3.1: Categorisation of local image features in the descriptor benchmark from classification perspective. True Positives (TP) together with False Negatives (FN) are features with sufficient overlap whereas matches are pairs whose descriptor distance is smaller than a threshold.

3.2 Evaluation of feature descriptors

The previous benchmark has been designed mainly for testing local image feature detectors. For testing the descriptors, Mikolajczyk and Schmid [28] propose slightly different framework which tests the image descriptors from an image classification perspective where the goal of the classifier is to classify each pair of image patches using image descriptors. It uses the same datasets as the benchmark for testing detectors which means that the ground truth is defined with homography between a reference image I and testing image J.

For pair of images I, J, related with a homography H, the input data for the classification is a set of ordered pairs of image frames $\mathcal{X}, \mathcal{X} \subseteq F(I) \times F(J)$ with labels $\mathcal{Y} : \{1, -1\}$ where for $x \in \mathcal{X}, x = (a, b)$ label $y = 1$ signifies that $\textsc{Overlap}(a, b) > 1 - \epsilon_d$ or $y = -1$ otherwise and ϵ_d is maximal overlap error. The task is to classify whether the pair of image frames are in correspondence (positive sample) or are not corresponding, i.e. to find function $\textsc{Classify} : \mathcal{X} \to \mathcal{Y}$based only on descriptors of the image frames.

The pairs $x = (a, b)$ with label $y = 1$ are called correspondences (possible correct matches) and pairs where $\textsc{Classify}(x) = 1$ are in the former article called matches, see Figure 3.1. There are several variants of this benchmark described in [28] which differ in the matching strategies:

Threshold-based matching In case of threshold based matching the classification is done simply as:

$$\textsc{Classify}_{\text{tb}}(a, b) = \begin{cases} 1 & \text{if } \|\mathrm{D}(a) - \mathrm{D}(b)\| < t \\ -1 & \text{otherwise} \end{cases} \tag{3.11}$$

Where t is descriptor distance threshold. Correspondences are all frame pairs which fulfil the overlap condition 3.11.

Nearest-neighbours matching In case of nearest-neighbour matching, the classification function is defined as:

$$\textsc{Classify}_{\text{nn}}(a, b) = \begin{cases} 1 & \text{if } b = \underset{b_i \in F(J)}{\arg\min} \ \|\mathrm{D}(a) - \mathrm{D}(b_i)\| \wedge \|\mathrm{D}(a) - \mathrm{D}(b)\| < t \\ -1 & \text{otherwise} \end{cases} \tag{3.12}$$

This classification function achieves higher precisions as the particular value of descriptor distances varies with image transformations. The nearest neighbours matching classifies as a match only the closest descriptor below the threshold so there is smaller number of false

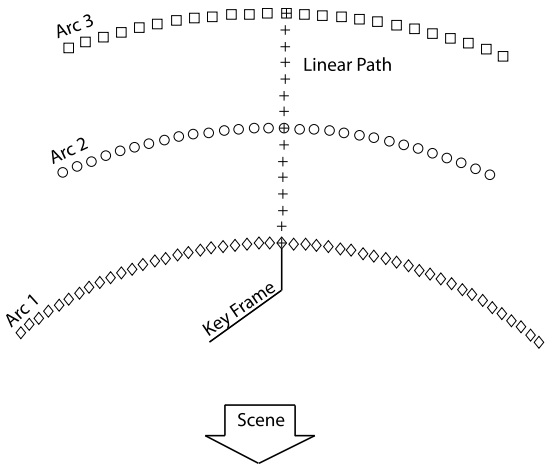


Figure 3.2: Visualisation of viewpoints per categories in DTU Robot dataset [1].

matches [29, p. 1622]. Because for each frame from the reference image there can be only one nearest neighbour, the set of correspondences is also defined differently, i.e. as a subset of matches where the frames fulfil the overlap criterion.

Second closest matching The second closest matching strategy extends the nearest neighbour strategy with a test, whether the distance ratio of the second closest and the closest descriptor is under some threshold. It rejects matches which are too ambiguous. This strategy has been proposed by [24]. The set of correspondences is the same as for nearest-neighbour matching strategy.

The performance of the classifier is visualised with a precision-recall curve varying the threshold of descriptor distances t. The precision and recall is defined as follows (for details how positives and negatives are defined see Figure 3.1):

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}} = \frac{\#\text{Correct matches}}{\#\text{Matches}} \tag{3.13}$$

$$\text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}} = \frac{\#\text{Correct matches}}{\#\text{Correspondence}} \tag{3.14}$$

In the original article, the PR curve is slightly modified so that on x-axis is $1 - \text{Precision}$ and on y-axis is Recall as the PR curve drawn in this way is more intuitive to read and is similar to the curves produced by the repeatability test.

3.3 DTU Robot 3D benchmark

DTU Robot 3D evaluation is a benchmark, proposed in [1], which compares the geometric precision of feature detectors. Contrary to homography benchmarks, ground truth is defined by a dense 3D stereo maps, which allows to find correspondences of points also in non-planar scenes. The dataset was acquired with a camera mounted on an industrial robot hand which provides very accurate camera positioning. Then the 3D model of the scene surface has been computed using structured light (the structured light does not affect the image data). The dataset contains 60 scenes of various objects (from model houses to fabric and vegetables) in 119 camera positions and 19 individual LED lightning. Camera positions are visualised in 3.2 and are grouped into three horizontal arcs and one linear path. All images are of resolution 1600×1200px.

The correspondences are obtained based on three criteria which are visualised in Figure 3.3. Having a pair of image frames $(a, b), a \in \text{F}(I), b \in \text{F}(J)$, it is a valid correspondence when it fulfils the following conditions:

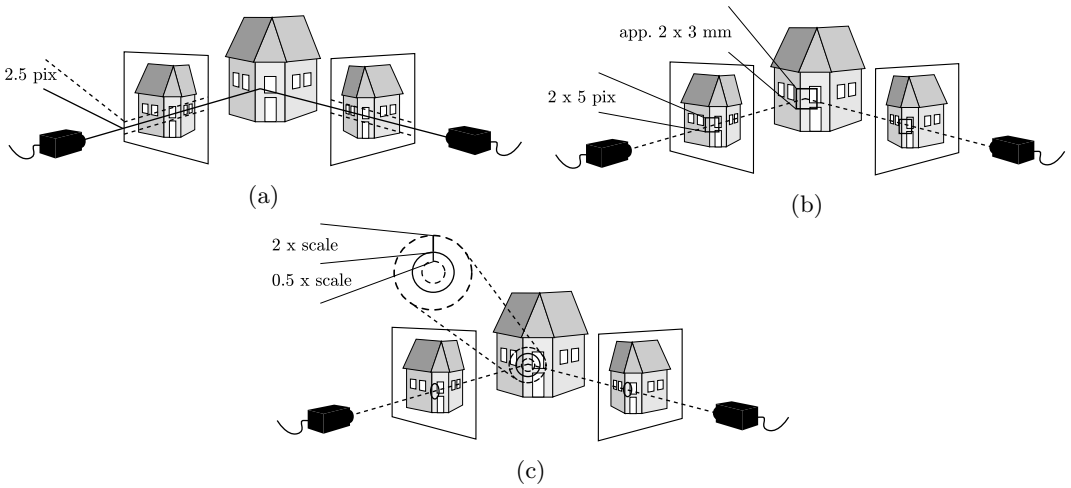


Figure 3.3: Criteria for correspondences in DTU Robot dataset [1].

Reprojection error From the known camera positions and camera matrices, the 3D point can be triangulated from the local feature centroid $\mathbf{x}_a$ and $\mathbf{x}_b$. The triangulation is computed using the linear solution produced by SVD [14, p. 312] (which minimises algebraic error) and the geometric reprojection error is further optimised using Levenberg-Marquardt algorithm [14, p. 314] to obtain optimised positions $\tilde{\mathbf{x}}_a$ and $\tilde{\mathbf{x}}_b$. The reprojection error is then calculated as:

$$\textsc{RepErr}(a,b) = \|\mathbf{x}_a - \tilde{\mathbf{x}}_a\|^2 + \|\mathbf{x}_b - \tilde{\mathbf{x}}_b\|^2 \tag{3.15}$$

Frame pair (a,b) is a correspondence when:

$$(a,b) \in \mathcal{A} \iff \textsc{RepErr}(a,b) < \epsilon_{rep} \tag{3.16}$$

Where ϵ_{rep} is maximal reprojection error usually set to $\epsilon_{rep} = 2.5$px.

Consistence with scene surface In this criterion the 3D surface reconstruction is used. In order to decide whether a pair of frames $(a,b) \in \mathcal{B}$, the frame centre $\mathbf{x_a}$ is reprojected onto a scene surface. Then a box of size 3mm in the scene is reprojected to the tested image and the correspondence is correct only when $\mathbf{x}_b$ lies in this reprojected box.

Absolute scale consistence This condition tests whether the frame scales are consistent. It is fulfilled only when:

$$(a,b) \in \mathcal{C} \iff \max\left\{\frac{\sigma'_a}{\sigma'_b}, \frac{\sigma'_b}{\sigma'_a}\right\} < 2 \tag{3.17}$$

Where σ'_a and σ'_b are scales of the detected features reprojected to the scene surface.

Chapter 4

Improvements to VLBenchmarks project

In this chapter we describe our contribution to VLBenchmarks project. It has been mainly motivated by the necessity of having easy and fast benchmark software to assess the performance of various detectors and compare the results to the other state-of-the-art detectors.

Early version of the VLBenchmarks project, including parts presented in this work (Homography and Retrieval benchmark), has been presented at ECCV 2012 in Florence as a part of the tutorial *Modern features: advances, applications, and software*[1].

In this chapter we introduce the VLBenchmarks project and the state of the project before our commitment. Then we follow with implementation details and improvements of particular benchmarks. Among these benchmarks are two new benchmarks which has been newly introduced (epipolar benchmark and retrieval benchmark).

4.1 VLBenchmarks

VLBenchmarks is a project implemented in Matlab and originally developed by Varun Gulshan and Andrea Vedaldi. Its main goal is to gather several benchmarks of computer vision algorithms. The former version implemented only computation of the repeatability score of local image feature detectors, see Section 3.1 for details. It contained wrappers for the original evaluation protocol by Kristian Mikolajczyk and some basic management structures to download detector binaries. It is important to note that the repeatability test implemented was several times faster than the original implementation however yielded rather different results than the KM test. The code of the original version is still available in public repository [2]. In our implementation the overall structure and most of the original code has been changed.

We have extended this project with additional tests. First, we have implemented matching score computation from [29] and created a versatile framework which allows to easily include new datasets, features extractors and evaluation benchmarks. Second, we have implemented novel direct evaluation of feature detectors and descriptors for image retrieval (further denoted as retrieval benchmark) which mainly tests the distinctiveness of the detected regions and their descriptors. Another frequent use-case of interest region detectors is in 3D scene reconstruction pipelines where the precision of the feature location is crucial. For these cases we have implemented simple epipolar geometry test where we measure number and ratio of true correspondences in the set of tentative correspondences. The software was available as an open source project, Anders Boesen Lindbo Larsen had contributed by including the DTU Robot benchmark into the VLBenchmarks projects, which uses much more detailed ground truth data than our epipolar geometry benchmark.

Finally we have extended the project with the tests proposed in [28] for testing the local image features descriptors. Originally this test is limited for homography ground truth but it was extended to use the epipolar geometry ground truth.

[1] `https://sites.google.com/site/eccv12features/home`
[2] `https://github.com/varungulshan/VLFeat_benchmarks`

The architecture of the VLBenchmark project has been altered significantly in comparison to the original code. The project has been divided into three packages of Matlab classes:

Benchmarks - Implementation of the benchmarks.

Datasets - Wrappers around datasets used in benchmarks.

Local Features - Wrappers around interest regions detectors and their description algorithms.

The implementation of newly added features are also supports parallel processing and results caching. The installation process has been improved so that all datasets and software are downloaded, installed or compiled automatically on demand (simple dependencies between the modules are also implemented).

The modifications described in this thesis were published and shared under BSD license as a subsidiary project of VLFeat library. The latest version is available in GIT repository[3]. Documentation with tutorials is available on the project website[4]. Some examples of the new application programming interface of the benchmarks is presented in Appendix B, which contains tutorials for repeatability and retrieval benchmarks.

In the following text we describe our contributions to the benchmark algorithms together with the available datasets wrappers. In the end of this chapter we list all the implemented wrappers of local image feature extractors.

4.2 Homography benchmarks

The homography benchmark was extended by the matching score calculation, for which we needed to include the support of feature descriptors. The repeatability score calculation was modified to closely resemble results of the original benchmark implemented by Mikolajczyk et al. [29]. On top of that, we have significantly improved the evaluation speed and added several new datasets.

4.2.1 Speed improvements

Similarly to the original implementation for calculating ellipse overlap (Equation 3.2) we use the same numerical approach where the ellipse areas are sampled and overlap is approximated as the ratio of overlapped samples. In the original code,the pairs taken into account, were only limited by the distance of the frames centre and the size of feature regions (approximated by a bounding box and maximal elongation based on eigenvalues of the ellipse). A new condition is used in order to speed up the calculation, which compares the ratio of the regions areas and used them as an upper bound of ellipse overlap.

The matching strategy that uses the stable bipartite matching of descriptors has been implemented with an inefficient way by a naive algorithm with asymptotic computational complexity bounded by $O(n^3)$, where n is the number of descriptors. Simple by sorting the weights we got $O(n^2 \log n^2)$ algorithm and with another implementation improvements we have achieved computation time in order of seconds instead of minutes for pair of images.

4.2.2 Comparison to original implementation

We have also finished implementation of wrappers of the original Mikolajczyk's code to compare our implementation to the original benchmark. The implementations we compared on the datasets from [29] and the average relative errors are given in table 4.1. Note that the relative error observed is usually caused by a single missed correspondence/match.

We have also compared the processing times which are given in table 4.2. The big difference between the mean and median suggests that we have not only improved the constant factors, but also overall complexity of the computation.

[3] `https://github.com/vlfeat/vlbenchmarks`
[4] `http://www.vlfeat.org/benchmarks`

	Repeatability [%]	Matching score [%]
Average rel. error	0.33	0.15

Table 4.1: Average relative error in repeatability and matching score of the new implementation per image pair for all 120 image pairs (8 datasets per 5 image pairs for 3 detectors).

	Repeatability mean/median [s]	Matching score mean/median [s]
Original impl.	21.54 / 3.09	22.14 / 4
VLBenchmarks	0.7 / 0.45	3.15 / 1.37

Table 4.2: Mean and median of the processing times of old and new implementation of the benchmark. Values measured for all 120 image pairs (8 datasets per 5 image pairs for 3 detectors).

4.2.3 Datasets

Among other features we have also implemented a "Synthetic dataset" which is able to generate new homography datasets by successively applying one or more geometric or photometric transformations such as:

- Change of image scale
- Image rotation
- Change of affine camera viewpoint (specified with longitude and latitude)
- Image noise (Gaussian, salt&pepper)
- Blur (Gaussian, median)
- Jpeg compression
- Gradient brightness transformations (circular, linear)

All of these transformations can be parametrised by amount of distortion and arbitrary combined for any input image and the transformed images are cached.

Besides that we have also implemented a wrapper of the *Hannover Affine Dataset* [9] which adds some new challenging scenes with repeatable structures and also improves the precision of the homography estimation for the VGG Affine dataset scenes [29].

4.3 Epipolar geometry benchmarks

So far presented benchmark is limited to test repeatability of the detection algorithms on scenes which are related by a homography transformation. This means that the scenes are planar and it is usually not possible to test robustness to occlusions or depth discontinuities. Introduction of the non-planar scenes allows to better tune the size and shape of the measurement region for computations of the local features descriptors. This is hard to achieve on planar scenes, as the bigger measurement region always gathers more information without the risk of contamination by the background clutter or occurrence of depth discontinuities. This does not have to hold for 3D structured scenes where the local neighbourhood changes more significantly with the viewpoint change.

To asses correspondence for non-planar scenes, we have proposed a simple epipolar geometry test. The main goal was to create a benchmark which can be used for measuring spatial precision of the local feature detectors. The benchmark also estimates the performance of the local feature detectors in the 3D scene reconstruction and wide baseline matching, where the precise geometric localisation significantly influences the results.

The epipolar geometry is a bilinear relation between two image projections $\mathbf{x}_I \in \mathbb{R}^3$ and $\mathbf{x}_J \in \mathbb{R}^3$ of a scene point $\mathbf{X} \in \mathbb{R}^4$ in image planes I and J. The image projections are bound with an epipolar constraint [14, p. 245]:

$$\mathbf{x}_J^T F \mathbf{x}_I = 0, \tag{4.1}$$

where F is a fundamental matrix. This matrix can be interpreted as a transformation matrix that transforms a point $\mathbf{x}_I$ in the first image plane to a line $\mathbf{e}_J = F\mathbf{x}_I$ (epipolar line) in the second image plane. The rest of the constraint $\mathbf{x}_J^T\mathbf{e}_J = 0$ requires that the point $\mathbf{x}_J$ in the second image lies on the epipolar line $\mathbf{e}_J$.

In real scenes, the equation rarely holds even for the matching pairs of points due to discretization noise and errors in localisation of the local features. To estimate a reprojection error for a pair of points a *Sampson error* is used. Sampson error e^2 is a first order Taylor approximation of the reprojection error [14, p. 287, p. 314] defined as:

$$e^2(\mathbf{F}, \mathbf{x}_I, \mathbf{x}_J) = \frac{\epsilon^2}{\|\mathbf{J}\|^2}, \quad \epsilon = \mathbf{x}_J^T F \mathbf{x}_I, \quad \mathbf{J} = \left[\frac{\partial\epsilon}{\partial\mathbf{x}_I^{(1)}}, \frac{\partial\epsilon}{\partial\mathbf{x}_J^{(1)}}, \frac{\partial\epsilon}{\partial\mathbf{x}_I^{(2)}}, \frac{\partial\epsilon}{\partial\mathbf{x}_J^{(2)}}\right] \tag{4.2}$$

where $\mathbf{J}$ is Jacobian matrix of the algebraic error ϵ and $\mathbf{x} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, 1]^T$.

The advantage of the Sampson error is that the 3D coordinates $\mathbf{X}$ of the point in the scene do not have to be known or triangulated.

This error can be easily plugged into the detector and descriptor benchmarks described in sections 3.1 and 3.2 by using it as a new geometrical similarity measurement Sim. So instead of using the Overlap(a, b) we define a new frame similarity:

$$\text{SampsonErr}(a, b) = e^2(\mathbf{F}, \mathbf{x}_a, \mathbf{x}_b) \tag{4.3}$$

For computation of tentative correspondences Sampson error is computed for all pairs $(a, b) \in F(I) \times F(J)$.

This benchmark is used for measuring precision of frame spatial location as the frame scale or shape is ignored. It can be used both for evaluating detectors and descriptors, however in case of descriptors the size of the measurement region affects descriptor matching performance.

Please note that the epipolar constraint 4.1 does not guarantee that all the correspondences will be correct as any feature which is matched close enough to the epipolar line is considered correct even when it is not detected on the same image structure. For this purpose a method proposed in [30] would be more appropriate as it uses trifocal tensor constraint instead of the epipolar constrain tha allows to restrict the projection of a scene point $\mathbf{X}$ in image planes of three perspective cameras. However, most of the datasets are limited only to image pairs, thus we have used only two images holding in mind that the ground truth is not bulletproof.

4.3.1 Datasets and ground truth

For testing the feature extractors we have used several stereo image pairs, which has been collected by Lebeda et al. [19][5]. The image pairs in this dataset are shown in Figure 4.1. It contains scenes of planar objects but also highly structured scenes with repetitive patterns (`plant` and `leafs`).

The datasets does not contain the ground truth fundamental matrix. The ground truth was computed using the CMP WideBaseLine software, developed by Perďoch et al. [31][6]. We used all available affine detectors combined together (Hessian Affine, Harris Affine, MSER, DoG) in order to sufficient coverage of image by correspondences and support better estimate of the fundamental matrix. RANSAC algorithm with local optimisation [8] is then used to find the fundamental matrix. Figure 4.2 shows the comparison of the performance of three descriptors on a planar scene with an affine transformation and a scene with a significant amount of occlusion. It can be seen that in the LIOP descriptor [49] it is much better than SIFT on a planar scene, but worse on a structured scene.

4.4 DTU Robot 3D benchmark

This benchmark has been contributed to the VLBenchmarks project by Anders Boesen Lindbo Larsen and is based on article [1]. As it is not our work, we only describe the way how it is

[5] Available at `http://cmp.felk.cvut.cz/data/geometry2view/index.xhtml`

[6] `http://cmp.felk.cvut.cz/~wbsdemo/demo/`

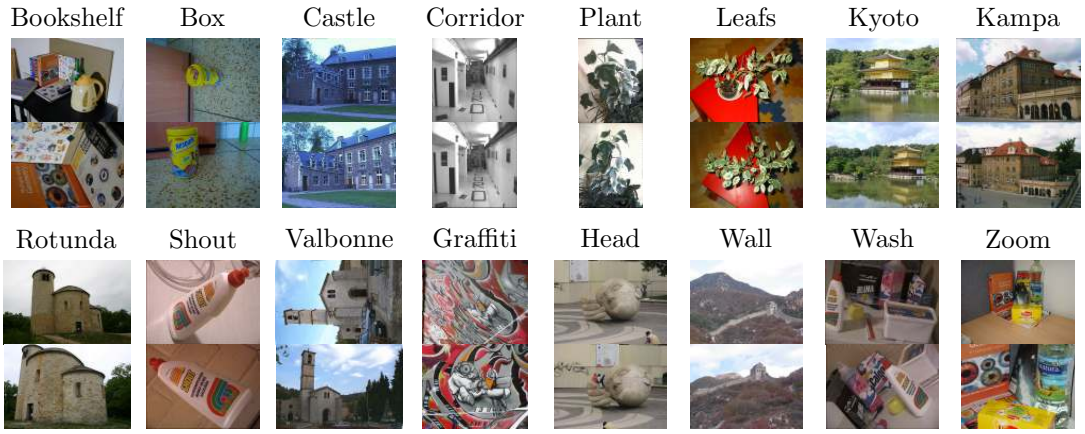
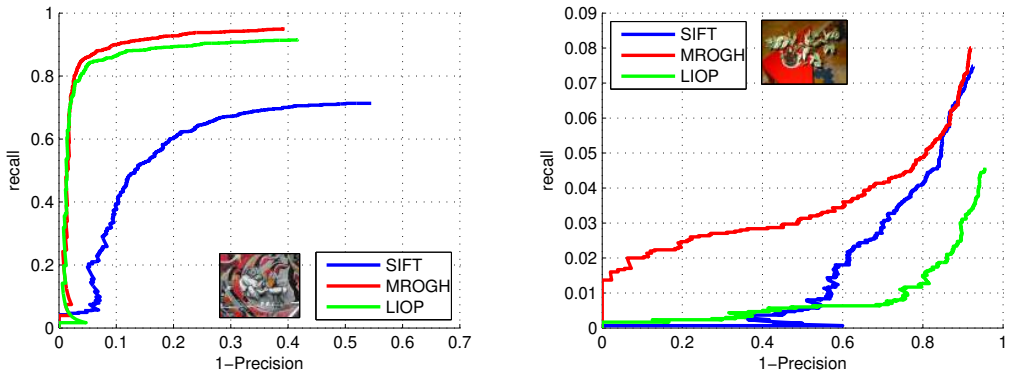


Figure 4.1: Image pairs in the CMP WBS dataset. It can be seen that some pairs are generally a homography, i.e. pictures of planar objects, whereas other scenes contain 3D objects with significant occlusions.



(a) Homography (*Graffiti*)

(b) Epipolar geometry (*Leafs*)

Figure 4.2: Comparison of descriptor performances using CMP Hessian-affine on different types of scene. In case of the Epipolar geometry LIOP descriptor does much worse than SIFT descriptor, even when on planar scene it is vice versa.

included into our framework. We found this benchmark useful for tuning parameters of the local feature detectors.

This benchmark is intended only to test the local feature detectors. On top of the epipolar geometry benchmark the corresponding image frames must also have consistent scales. Because this benchmark uses three conditions for a correspondence it is hard to find a similarity score which would be used for image frames matching. Therefore tentative correspondences are generated as follows:

$$\mathcal{T}_{DTU} = \mathrm{SM}(\mathcal{B} \cap \mathcal{C}, -\textsc{RepErr}) \tag{4.4}$$

meaning of the symbols used is described in section 5.2. The correspondences are then obtained as:

$$\mathcal{C}_{DTU} = \mathcal{T}_{DTU} \cap \mathcal{A} \tag{4.5}$$

repeatability, using this benchmark is the computed according to the original article as:

$$\textsc{DetRep}_{DTU} = \frac{|\mathcal{C}_{DTU}|}{|\mathrm{F}(I)|} \tag{4.6}$$

In the original article it is called recall, however in our framework, repeatability term fits better.

4.5 Descriptor evaluation

We have implemented the descriptor evaluation benchmark based on [28] closely following the procedure depicted in section 3.2. Contrary to the original implementation, we compute the precision and recall for each pair of image frames (a, b), not only for a limited sets of thresholds. he computation time is slightly better than in the original implementation as it benefits from faster way for computing frame overlaps. The most computationally expensive step is the sorting of the cross-distances in the case of threshold based matching, respectively computing the nearest neighbours in case of nearest neighbour matching (or second closest matching as well). To find the nearest neighbours, we use rather fast implementation from the Yael library[7]. Disadvantage of this library is that for the NN it supports only limited subset of distance functions (L1 and L2-Norm).

4.6 Retrieval benchmark

The repeatability and matching score are performance measures which are relatively easy to compute however they do not address some specific issues of retrieval systems. The available homography datasets are rather limited and often targeted to a particular geometric or photometric transformation.

Our image retrieval setup consists of a simple image retrieval system which is based on the framework proposed by [15]. The performance of the test is evaluated using a *mean average precision* (mAP), proposed in [33]. The benchmark may be used as a performance comparison of different detectors. The parts of the benchmark were developed with help of Relja Arandjelovic.

4.6.1 Retrieval system design

The task of retrieval system is to obtain instances of an object from a large dataset of images. The object sought is defined by the user as a bounding box in a query image. The process of searching for the object instances in the database proceeds as follows. In the first step, descriptors of all database images are computed and stored. Then descriptors from the query image are extracted and the closest descriptors for each query descriptor are found using the *K-Nearest Neighbours*, defined below. Based on the occurrence of the closest descriptors, each database image gets a vote based on a simple voting criterion which uses normalised descriptor distance.

Let the dataset of images consists of m images. Each image is described by a set of descriptors $n_j, j = 1...m$. All extracted descriptors are stored in a database as a collection of descriptors $\mathcal{Y} = y_1, ..., y_i, ...y_n$. Let us denote the image where the descriptor y_i has been observed is as $\mathrm{im}(y_i)$.

The descriptors detected in the query window are denoted as a set $\mathcal{D}(q)$. For each query descriptor $x \in \mathcal{D}(q)$ the K-Nearest descriptors are obtained as:

$$\mathcal{N}_k(x) = k - \underset{y_i \in \mathcal{Y}}{\arg\min}\, d(x, y_i) \tag{4.7}$$

where $d(.,.)$ is the Euclidean distance. The r-th descriptor is denoted as $N_r(x)$ therefore for each query we get ranked list of descriptors $\mathcal{N}_k(x) = N_1(x), ..., N_k(x)$.

Intuitively, the descriptors suggest the relevance of the sought object in the images where they were computed. Thus, each descriptor $y \in \mathcal{N}_k(x)$ cast a vote for its image (image where it has been detected). Then, all the votes for each image are added up and a the resulting list of database images is sorted in the order of descending number of votes. Jégou et al. [15, p. 6] observed that the performance of the system can be improved by weighting the votes of the descriptors according to their distance to the query descriptor x. This weighting cannot use the absolute distances $d(x, y)$ as they vary across a particular query descriptor. Instead, the vote of the descriptor is weighted by a value which expresses how much closer is the descriptor

[7] `https://gforge.inria.fr/projects/yael/`

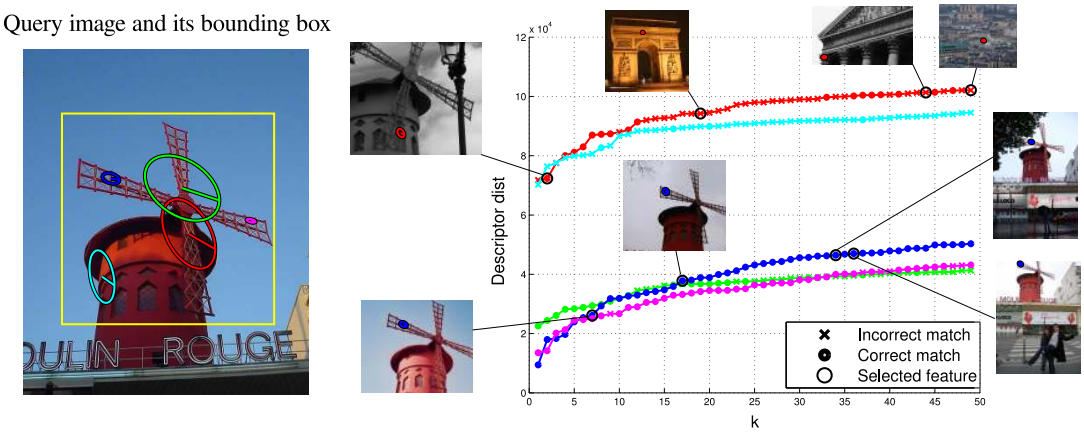


Figure 4.3: Example of 50-Nearest neighbours in Paris dataset for selected five local image features..

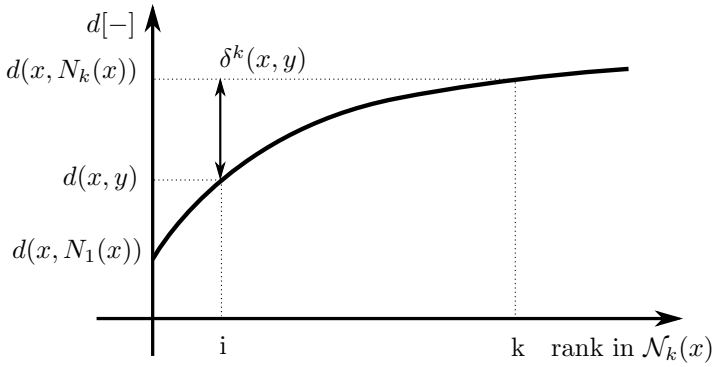


Figure 4.4: The weighting $\delta^k(x, y)$ of a vote of descriptor $y = N_i(x)$ which is i-th nearest neighbour of query descriptor x.

to the query descriptor than the most distant descriptor in $\mathcal{N}_k(x)$. Expressed mathematically, the vote strength $\delta^k(x, y)$ is defined as:

$$\delta^k(x, y) = \max(d(x, N_k(x)) - d(x, y), 0) \tag{4.8}$$

The weighting is visualised in 4.4. Jégou et al. [15, p. 8] had shown that this voting criterion is superior to other examined criteria as it models the distinctiveness of the descriptor w.r.t. the other descriptors in the database.

The final score of an image b for a query q is calculated and normalised using Square Root Normalisation (SRN) so that the score does not depend on the number of detected descriptors in the image b or in the query q:

$$s_a(q, b) = \frac{1}{\sqrt{n_q}\sqrt{n_b}} \sum_{x \in \mathcal{D}(q)} \sum_{y \in \mathcal{N}_k(x) \cap \mathcal{D}(b)} \delta^k(x, y) \tag{4.9}$$

where $\mathcal{D}(a)$ is a set of all descriptors in image a and $n_a = |\mathcal{D}(a)|$. The SRN leads to normalisation that assigns score $s_a(q, q) = 1$ to the query image, for which all the descriptors are matched to themselves. Based on this score a ranked list of database images is created.

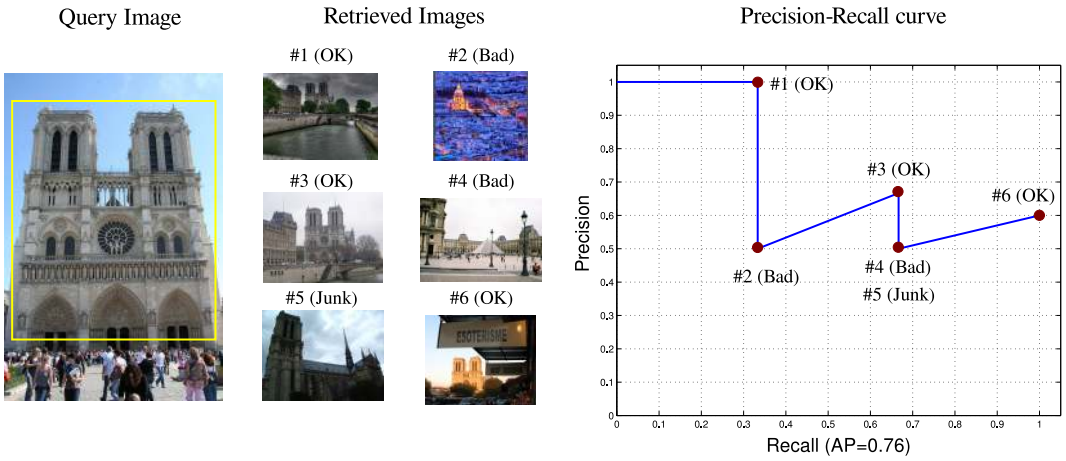


Figure 4.5: An example of mAP calculation for retrieved list of 6 images. The images labelled as *Junk* (e.g.te bottom left image in the middle column) are ignored and do not influence the precision nor the recall value.

4.6.2 The performance evaluation of the image retrieval system

To evaluate the performance of the image retrieval system, we use the *Oxford Buildings* [33] dataset (5K version) and *Paris Buildings* [32] dataset. The datasets contains images of the tourist landmarks in Oxford resp. Paris. The images were downloaded from Flickr using corresponding tags together with many other unrelated images accidentally having the same tag. The extensive ground truth is provided. For each landmark a three lists of images are provided: *Good* images, where the whole object is visible, *Ok* where at least 25% of the object is visible and *Junk* where only less than 25% is visible or there is some significant occlusion. All other images are for the given landmark considered as having label Bad. Besides that the database contains a set of *distractors* which does not contain any queried object.

For each query, set of positive samples is defined by the *Good* and *Ok* sets. Images in set *Junk* does not affect the computed score (are ignored). The set of images with label Bad are considered as negative examples. The *precision* of the retrieved list of images is calculated as the ratio between the number of retrieved images in the positive set and the number of all retrieved images. The *recall* is defined as a ratio between the number of retrieved images in the positive set and the number of all images in the positive set. The *average Precision* (AP) is defined as the area under the precision-recall curve. The ideal AP = 1, i.e. that the system puts all positive instances in the front of the list. An example of mAP calculation is shown in Figure 4.5. The *mean Average Precision* (mAP) is a mean of all average precisions over all queries available in the dataset.

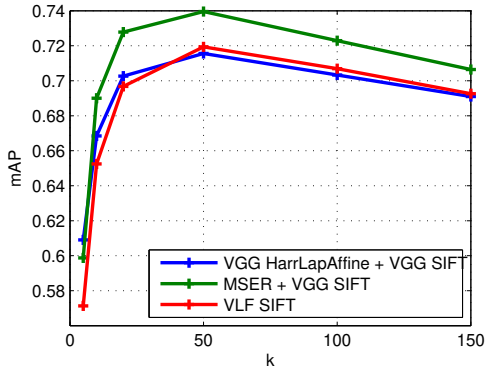
The above-mentioned image retrieval datasets contain thousands of image. To limit the computation time in our benchmark, only subset of the dataset is used that contains all images from the lists (*Good*, *Ok* and *Junk*) and a subset of images from the *Distractors* set of images. This helps to keep the number of descriptors in the database $\mathcal{Y}$ tractable and makes the evaluation faster. The subset of Distractors is generated by uniform random sampling, with the sampling seed set by user.

4.6.3 Parameters of the retrieval system

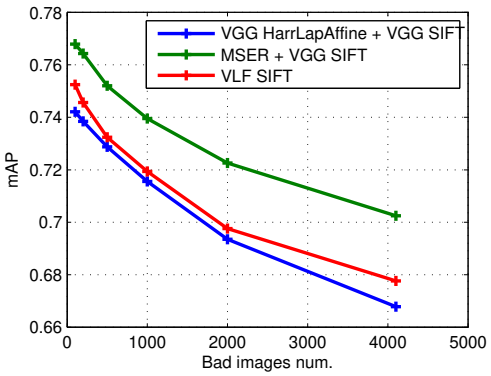
The crucial parameter of the retrieval benchmark is k, the number of nearest neighbours which is obtained for each query descriptor. Therefore, we have performed an experiment, where the measured mAP and different types of features in order to test, whether all feature detectors behave the same. The results are shown in Figure 4.6a. It can be seen that generally all the tested algorithms behave similarly and in all cases the best results are obtained with

$k = 50$, we use this value in all other experiments.

Another parameter which affects the results is the number of images from the *Distractors* set. These images are used mainly as a source of distraction as they do not contain the object instances. Intuitively, with less distractors, we obtain better score. To measure the influence of the subset size, we have performed an experiment, where the mAP is measured on the Oxford dataset, over 9 different seed values. The results of this experiment are shown in 4.6b. We can see that all interest region detectors behave the same and with increasing number of distractors, the score slowly decreases. Even in the tests repeated with different random seed, the standard deviation of the score was not higher than $\pm 0.6\%$.



(a) mAP based on k-value, mean over 9 measurements.



(b) mAP based on k-value.

Figure 4.6: Influence of the internal parameters of retrieval benchmark on the mAP score, measured on the Oxford 5K dataset with SIFT descriptors. Different SIFT descriptor implementations and different measurement regions were used therefore, the absolute values of mAP between detectors are not comparable. Results are averages of mAP over 10 random seeds.

Implementation

The computationally most expensive step of the benchmark - the K-nearest neighbours was implemented using the function of the Yael library[8]. We have considered the use of fast approximated nearest neighbours library (FLANN), however we have found that their implementation in the VLFeat library is in some cases slower than exact NN in the Yael library. Therefore we have decided to use precise NN and not blur our results with the FLANN approximations. After reducing the time for nearest neighbour search, the majority of evaluation time is taken by computation of descriptors.

The computation of scores has been parallelised using the Matlab Parallel toolbox, which uses multi-process paralellisation, in order to further speed up the computation, it is possible to limit the number of images involved in one NN search. The overhead is then only in the ordering of the retrieved nearest neighbours by their distances and keeping the closest k features. The speed-up is mainly due to smaller memory requirements so that the data does not have to be cached. This modification also allows the computation on computers with smaller memory.

4.7 Miscellaneous improvements to VLBenchmarks

Among other features we have also improved the former project with the following features:

- Added several new detectors and descriptors (see the following text).

[8] `https://gforge.inria.fr/projects/yael/`

- Added caching of the results and intermediate results (mainly detected local features). This speeds up the computations and allows simple and fast experiments with the parameters because only the results affected by the changed parameters are recalculated. The caching granularity may differ for the particular detector, descriptor or benchmark parameters, the modification date of the detector's binary or the input image.
- Rewritten installation system in order to support compilation of the required libraries (e.g. OpenCV, VLFeat).
- New logging framework that allows to set different verbosity for each module.
- Parallelisation support.

4.7.1 Feature detection algorithms included in the benchmark

The modular design of the software allows to easily create wrappers around several feature detection and description algorithms (and any combination of detector and descriptor). We have included wrappers for several feature detector and descriptor implementations. Each implementation supports different set of detector variants, level of invariance and available descriptors. In the software, the following implementations are included.

CMP Detectors implemented mainly by Michal Perdoch and other members of Centre for Machine Perception. The package is currently not publicly available and we use proprietary binaries. From scale-space detectors, it supports DoG, LoG, Hessian and Harris response functions and affine shape of the features can be estimated using Baumberg iteration. It does not have any hybrid detectors. It also contains MSER implementation [26]. From the descriptors it supports only the SIFT descriptor.

VGG Detectors used in evaluation by Mikolajczyk et al. [29][9]. It supports Hessian-Laplace and Harris-Laplace detector and their affine variants. On the website there is also the MSER detectors implementation available which is identical with the CMP implementation, however it is usually referred as VGG MSER. From the descriptors, it supports the SIFT descriptor but also many other mentioned in [28]. All algorithms are available only in binary form.

VLFeat Open source library[10]. From the scale space detectors, it supports DoG and Hessian responses and have also affine shape estimation procedures for detected features. From the hybrid feature detectors it supports Hessian-Laplace and Harris-Laplace but also Multiscale Hessian and Multiscale Harris where the non-maxima suppression is not performed over scales. During our internship in VGG in Oxford in April and May 2012 I have helped with their implementation. From the descriptors is currently supported is only the SIFT descriptor.

OpenCV Open source library[11] provides a vast variety of similarity invariant detectors. In VLBenchmarks we have prepared wrappers for their DoG, SURF and FAST detectors. From the descriptors, wrappers are available for the SIFT and SURF.

Additionally, we have created wrappers for other descriptors such as LIOP [47] and MROGH [10] were added together with the wrappers for the original implementation of SURF detector by Bay et al. [5] which is also used in our experiments.

All wrappers has an automatic installation and compilation process. This has been rather tricky task for the local image features algorithms which needs to link with the OpenCV library. The Matlab, used for implementation of the framework is distributed as a binary package, and dynamically linked to its own version of `libstdc++` libraries. Therefore, each binary which is invoked in Matlab has to be able to use the Matlab's versions of standard

[9] `http://www.robots.ox.ac.uk/~vgg/research/affine/`
[10] `http://vlfeat.org`
[11] `http://opencv.org`

library. The OpenCV wrappers compiled without taking this into account would use the different version of `libstdc++` library available in the operating system. This problem is solved by compilation of OpenCV in the Matlab environment. In future we plan to distribute compiled binaries in order to speed-up the installation process.

4.8 Possible future improvements

There are several ways how to improve the benchmarks implementation. Our code currently lacks support for other similarity measures as it relies on the Yael kNN algorithm. Also Yael library is supported only for Unix/Linux platforms whereas majority of the VLBenchmarks code is multi-platform. The evaluation time of the Retrieval benchmark,can speed up by utilization of approximated nearest neighbours.

There are several issues of the evaluation protocols that which has not been addressed so far. For example, protocols that measure only the local image feature geometry, with increasing number of features, the chance that a correspondence will be found by accident increases. Some detectors it also return multiple detections around a single local feature. Some sort of penalisation of this behaviour would be useful, however it is out of scope of this work.

Chapter 5

Experiments with detector parameters selection

Development of local image feature detectors has been traditionally connected with some form of tests which were used to both avoid code regression but also to set properly detector parameters. Harris-Laplace and Hessian-Laplace detectors, and their affine-invariant variants has been tested with benchmarks for repeatability and matching score presented in [29]. Similarly, when in [24] the DoG detector and SIFT descriptor has been tested, experiments has been performed by querying a database of descriptors built from 112 image which were artificially rotated, scaled and corrupted with white noise. Though, it seems that parameters of these algorithms has been set with tests based on planar data (data transformed by homography). In this chapter we use benchmarks presented in the previous chapter to revise some of selected detector parameters. Most of the parameters are bound to detectors which use pyramid scale-space. Goal of this section is not to thoroughly compare all feature detectors but in some cases we compare different implementations. Motivation in this is mainly avoid interpretations which would show to be bound to a particular implementation of a detector.

In the first section the nominal image scale which defines Gaussian blur in the input image, an almost forgotten parameter of the scale space is explored together with its properties. The second parameter investigated in this chapter is response function threshold. Then the behaviour of detectors with different initial scales is tested. The last part of this chapter is devoted to properties of measurement region which is bridging feature detectors and their descriptors.

For all the test presented in this section we have used SIFT descriptor as it had shown to be the most universal one for all tasks. We have tried tests with other descriptors as well but with similar results.

5.1 Nominal image blur and scale space detectors

Continuous scale space-based feature detectors which additionally to the feature point location detect also its scale (extent) usually examine evolution of local image feature response function over scale, or in other sense, over a range of spatial frequencies. Yet some of the highest spatial frequencies can be caused by noise.

Therefore the input image is filtered with a Gaussian kernel which corresponds to setting initial scale σ_0. Lowe [24, p. 9] experimentally measured that the best results are achieved by $\sigma_0 = 1.6$. The initial scale has two effects on the scale space. First, it limits the minimal scale on which features can be detected, pruning out small features. Second, higher values help to to fulfil the sampling theorem as it removes high spatial frequencies [1].

In most implementations of the scale space pyramid there is also hidden estimated nominal scale of the input image σ_n which is commonly set to $\sigma_n = 0.5$ and often is not even adjustable as user parameter. What if the image does not have the nominal scale of this value and is blurred even more? This is the issue we would like to solve in this section. In the first part, we show in detail how the pyramid scale-space is being built and our changes introduced to reflect any nominal image blur. With these changes we are able to achieve much higher repeatability as it is shown in Figure 5.5.

5.1.1 Analysis of the scale space pyramid building algorithm

The commonly used algorithm to build a scale space pyramid [24] is described in algorithm 3

Algorithm 3 Pyramid scale space building

Input: Input image I, nominal image scale σ_n, initial image scale σ_0, num. of octaves O, num. of octave layers S
Output: The scale space pyramid

Smooth the input image to initial scale $\hat{L}_{0,0}(\mathbf{x}) = G(\mathbf{x}, \sqrt{\sigma_0^2 - \sigma_n^2}) * I(\mathbf{x})$,
$\sigma_{0,0} \leftarrow \sigma_0$
for all $v \in 0, \ldots, O$ **do**
    $d_v \leftarrow 2^v$
    for all $s \in 1, \ldots, S+1$ **do**
        $\sigma_{s,v} \leftarrow \sigma_0 2^{v+\frac{s}{S}}$
        $\hat{L}_{s,v}(\mathbf{x}) = G\left(\mathbf{x}, \sqrt{\frac{\sigma_{s,v}^2}{d_v} - \frac{\sigma_{s-1,v}^2}{d_v}}\right) * \hat{L}_{s-1,v}(\mathbf{x})$,
    end for
    $\hat{L}_{0,v+1}(\mathbf{x}) = \hat{L}_{s+1,v}(2 \cdot \mathbf{x})$
end for
return Scale space pyramid $\hat{L}_{s,v}$.

If the pyramid is build in this way, the upper layer of an octave can be directly down-sampled to obtain first layer of the first octave. In practical implementations the layers in the octaves does overlap by more than one level in order to perform reasonable non-maxima suppression. In case when the image nominal scale σ_n' is higher than assumed $\sigma_n = 0.5$, the first layer of the scale space is overly blurred with error $\Delta_0 = \sqrt{\sigma_0'^2 - \sigma_0^2}$. As the image smoothing increases, the scale error Δ_s decreases exponentially as can be seen in Figure 5.1. This means that the error is relatively higher for smaller features.

[1] This is in a way a design decision as the frequency characteristics of the Gaussian filter do not have ideal low pass filter properties so with increasing σ_0 we increase the cutoff when the pyramid layers are down-sampled.

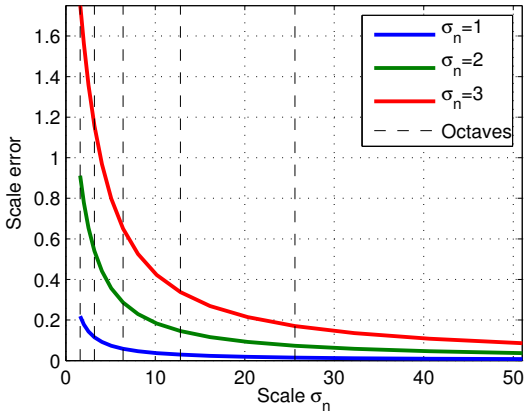
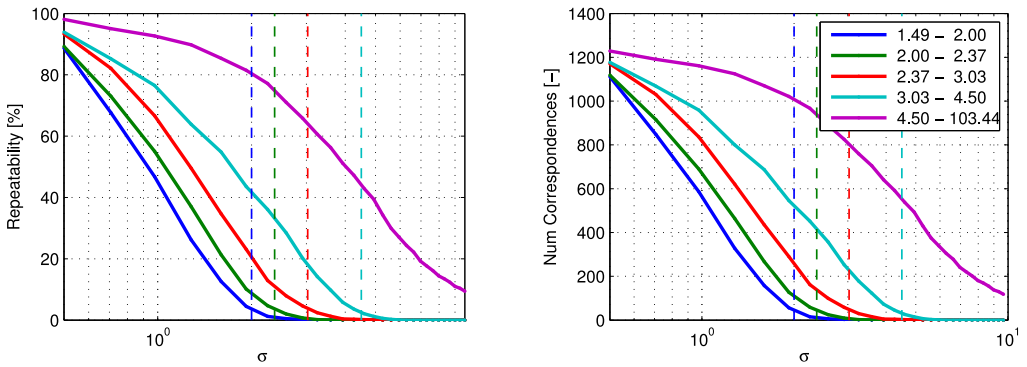


Figure 5.1: Scale errors for features detected in Guassian scale-space for different nominal image scale σ_n.

Overestimating nominal scale causes that scale of the detected features is bigger as it is shown in Figure 5.1. You can see that in the case of $\sigma_n = 3$, the error in scale for features from the first layer is of same magnitude as their size. This effects detector's performance as can be seen in Figure 5.2 where the increasing nominal image scale affects repeatability (Equation 3.1) of features of all sizes (even features bigger than the nominal scale).



(a) Repeatability

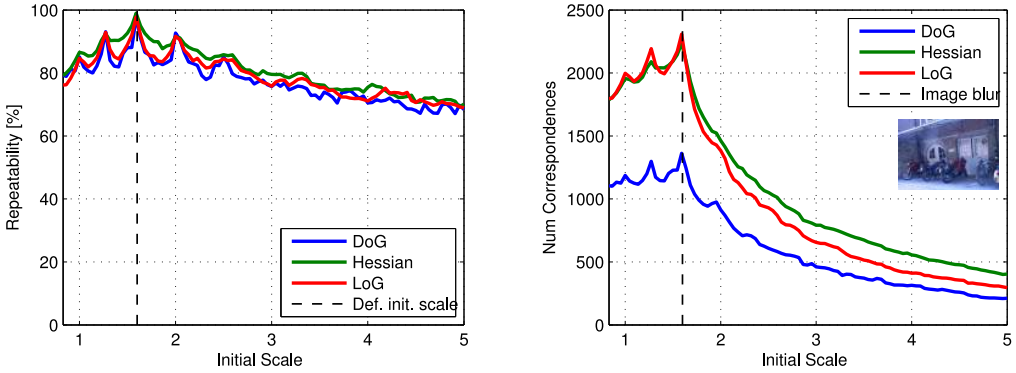
(b) Number of correspondences

Figure 5.2: Repeatability of original CMP Hessian detector per scale on increasingly blurred *boat 1* image [29]. Frames detected in the reference image are split into 5 intervals in their scales such that each interval contains the same number of frames. Then repeatability and number of correspondences per group is computed.

In order to tackle this problem we have improved Algorithm 3. The improvements are described in algorithm 4, which is able to build correct scale-space for any σ_n, i.e. also when $\sigma_n > \sigma_0$. The CMP detector implementations were modified to reflect the new algorithm. When $\sigma_n < \sigma_0$, the Algorithm 3 blurs the image correctly in order to obtain the requested initial scale. However in the cases when $\sigma_n > \sigma_0$, we have observed that adjusting the initial scale to the nominal scale (so that the original image can be used) does not lead to the same sampling of the scale space. To achieve the same sampling, pyramid layers must be located on the same scales independent on the nominal scale.

This can be seen in Figure 5.3 where the reference frames for repeatability experiment (without normalisation) are computed with $\sigma_0 = 1.6$ and tested against frames detected with the same detector but with different initial scale. Note that the repeatability and mainly

number of correspondences decreases. The most interesting part is the behaviour in interval $\sigma_0 \in (1, 1.6)$ where the features with higher spatial frequencies are not filtered out. We can see some small peaks which represent the layers of an octave $2^{1/S}$. We can conclude that the detector gains the best repeatability when the pyramid layers are located on the same scales. With a different sampling the repeatability, which should be theoretically 100% decreases to almost 80% in the tested image.



(a) Repeatability

(b) Number of correspondences

Figure 5.3: Repeatability between frames detected in image *boat* with $\sigma_0 = 1.6$ and frames detected with different values of initial scale σ_0 (x-axis).

Algorithm 4 Improved pyramid scale space building

Input: Input image I, nominal image scale σ_n, initial image scale σ_0, num. of octaves O, num. of octave layers S

Output: The scale space pyramid

if $\sigma_0 > \sigma_n$ **then**
  Smooth the input image to initial scale $\hat{L}_{0,0}(\mathbf{x}) = G(\mathbf{x}, \sqrt{\sigma_0^2 - \sigma_n^2}) * I(\mathbf{x})$,
  $v_0 \leftarrow 0$, $s_0 \leftarrow 0$
else
  $v_0 \leftarrow \lfloor \log_2(\sigma_n/\sigma_0) \rfloor$, $s_0 \leftarrow \lfloor \log_{2^{1/S}}(\sigma_n/(\sigma_0 2^{v_0})) \rfloor$
  $\hat{L}_{s_0,v_0}(\mathbf{x}) = I(\mathbf{x})$
end if
$\sigma_{s_0,v_0} \leftarrow \sigma_0$
for all $v \in v_0, \ldots, O$ **do**
  $d_v \leftarrow 2^v$
  for all $s \in s_0 + 1, \ldots, S + 1$ **do**
    $\sigma_{s,v} \leftarrow \sigma_0 2^{v + \frac{s}{S}}$
    $\hat{L}_{s,v}(\mathbf{x}) = G\left(\mathbf{x}, \sqrt{\frac{\sigma_{s,v}^2}{d_v} - \frac{\sigma_{s-1,v}^2}{d_v}}\right) * \hat{L}_{s-1,v}(\mathbf{x})$,
  end for
  $\hat{L}_{0,v+1}(\mathbf{x}) = \hat{L}_{s+1,v}(2 \cdot \mathbf{x})$
end for
return Scale space pyramid $\hat{L}_{s,v}$.

The algorithm 4 does not work straight forward for DoG detector, due to properties of the DoG response which approximates Laplace of Gaussian s.t. [24, p. 6]:

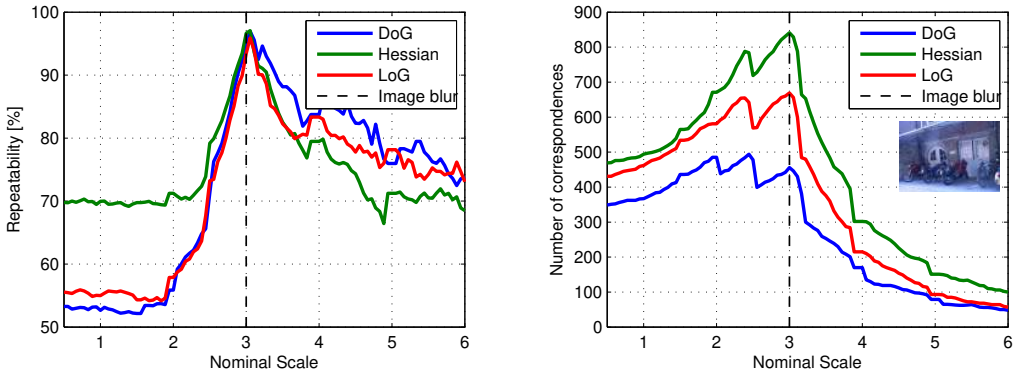
$$G(\mathbf{x}, k\sigma) = G(\mathbf{x}, \sigma) \approx (k-1)\sigma^2 \bigtriangledown^2 G \tag{5.1}$$

In pyramid built by Algorithm 3 k and σ are constants. In case of pyramid built by Algorithm 4 and the first two layers $\hat{L}_{s_0,o_0}$ and $\hat{L}_{s_0+1,o_0}$ the ratio between scales $\sigma_{s_0,v_0}/\sigma_{s_0+1,v_0}$

is not equal $2^{1/S}$ and therefore the DoG response is not comparable to other layers. Unfortunately this is not solvable by different feature response normalisation as in case of other feature response functions. To solve this problem when $\sigma_{s_0,v_0}/\sigma_{s_{0+1},v_0} < 2^{1/S}$, octave layer $\hat{L}_{s_0,o_0}$ is skipped.

Another issue of this algorithm is a fact that the standard deviation of the Gaussian kernel $\sigma_\Delta = \sqrt{\sigma_0^2 - \sigma_n^2})$ rather often gets smaller than 0.6 when the sampled discrete Gaussian kernel starts to resemble more a derivation kernel. In these cases the image is simply copied. However this causes that the repeatability and number of correspondences are rather uneven.

In Figure 5.4 you can see that for image blurred by Gaussian kernel with $\sigma = 3$, the peak in repeatability is achieved when the nominal scale is set to $\sigma_n = 3$ which verifies our algorithm.



(a) Repeatability (b) Number of correspondences

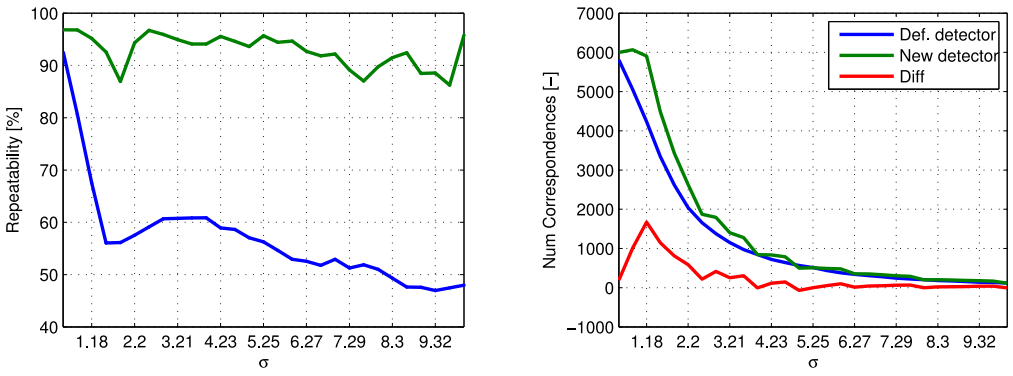
Figure 5.4: Repeatability between frames detected in an image *bikes 1* and its version blurred by Gaussian kernel with $\sigma = 3px$ for detectors with varying parameter σ_n. It can be seen that the maximum repeatability is achieved when $\sigma_n = \sigma$.

However the repeatability is not 100%. In Figure 5.5 it is shown how the repeatability behaves with increasing image blur σ when nominal scale is set to $\sigma_n = \sigma$. The repeatability is always around 90% which is much more than when the nominal scale of an image is assumed to be constant. The repeatability value varies also due to fact that we are not able to construct reliable Gaussian kernel with $\sigma < 0.6$. For $\sigma < \sigma_0 = 1.6$ the number of correspondences remains the same and then the number of correspondences decreases as more and more features are filtered out. However the remaining features are not detected with wrong scale so the repeatability remains high, plus we obtain little bit more correspondences in some cases.

To have more thorough understanding what is happening, in Figure 5.6 is shown how the repeatability behaves per scale intervals. When compared to Figure 5.2, for frames with scale higher than σ the repeatability remains around ideal 100%.

We have also tried several ways how to detect the nominal scale of an image, however we have not found yet any robust enough algorithm for doing so as the blur can differ in different parts of the images. We have also performed some tests with image blur caused by non-focused images where the blur kernel resembles more a disc. In those cases it also improved results but only to a certain extent.

Conclusions We have shown the significant influence of nominal input image blur to detector performance. The scale-space detectors gain the best performance when their scale-space is built from the same initial scale and with adapted nominal scale. For that we have proposed an improved algorithm for building a scale space which can with a prior knowledge of image blur significantly improve detector's repeatability.



(a) Repeatability

(b) Number of correspondences

Figure 5.5: Repeatability between the reference frames detected in image *bikes 1* and frames detected in *bikes 1* with different Gaussian image blur with std. dev σ. *Def. detector* is detector ignoring the σ_n parameter and *New detector* is detector with $\sigma_n = \sigma$. Computed with Hessian detector.

5.2 The response function threshold

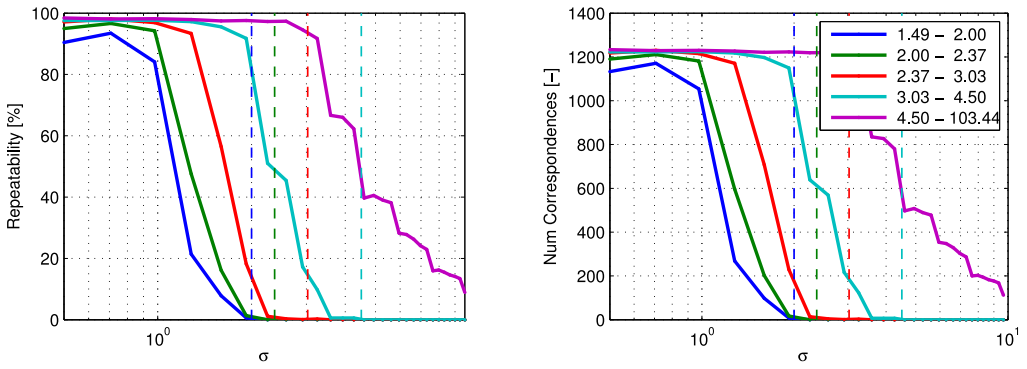
The response function threshold R_t is a minimal value of a response function (LoG, DoG, Hessian etc.) of a detected feature. Main purpose of the response function threshold is to filter out spurious local features which may arise from the stochastic processes in image acquisition rather than structures observed in the scene. It directly affects the number of detected features as it rejects features with low contrast.

Generally most of the authors which tried to measure feature detectors performance had struggled with the response function threshold and other parameters which affect the number of detected features. The number of detected features depends on the input image and also on the detector implementation. Therefore authors ([27],[2]) usually choose to use the default parameters of the detectors.

In this section we examine relation of the response function threshold to detector performance. DTU Robot 3D benchmark is used to measure the geometric precision of the detector and retrieval benchmark is used to measure the distinctiveness of the extracted features in image retrieval task. To obtain more general results we have performed the tests with several implementations of the local image feature detectors. Also when the implementations allow, we include both similarity and affine invariant alternatives of the detected features in order to see whether the behaviour is affected by affine iteration.

In CMP implementation of the detectors, the response function threshold is set directly as expected standard deviation of noise of image brightness values, with exception of the DoG approximation where it is set directly as the minimum difference of Gaussian value. In other implementations its value is directly used against computed values and it does not account different properties of the feature responses. In the case of the MSER detector [26], parameter Δ_{min}, minimal margin of intensity values for a stable region is varied.

Epipolar geometry In this experiment we test the geometric precision of detected local features. It is performed with the DTU Robot 3D benchmark (described in and). We test the detectors with varying response thresholds (or minimal margin for MSER detector) and measuring the repeatability on two selected camera viewpoints against the reference image. The reference image is captured with camera 0.5m away from the scene (*Key frame* in Figure). The first viewpoint, referred as *Linear path* is with camera 0.8, away from the scene with camera in the same bearing and is used to measure detector scale invariance. The second viewpoint, referred as *Arc 2* the camera is on a circular path 0.65m away from the scene and snaps the scene in angle of 25°. The particular camera viewpoints tested against



(a) Repeatability

(b) Number of correspondences

Figure 5.6: Repeatability of CMP Hessian detector with scale space built with Algorithm 4. Repeatability is plotted per scale on increasingly blurred *bikes 1* image. Local features detected in the reference image are divided into 5 intervals in its scale such that each interval contain the same number of features. Then repeatability per group is computed.

Figure 5.7: First 10 scenes used for repeatability calculation and visualisation of selected viewpoints.

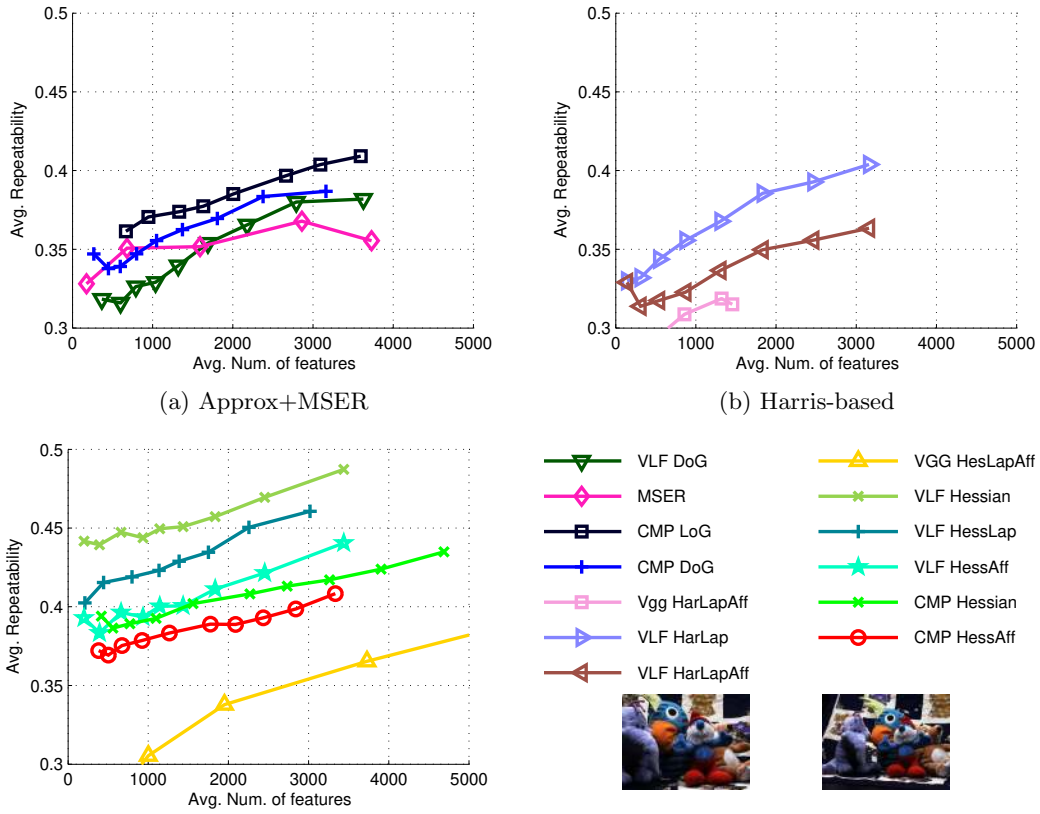
the reference frame are visualised in figure 5.7. The repeatability has been averaged over the first 10 scenes which include a variety of materials and structures. The scenes are shown in figure 5.7.

In order to compare the detectors fairly, the repeatability is drawn in the plots as a function of number of features which changes with response function threshold. For some algorithms, such as RANSAC in wide baseline stereo, the fraction of inliers is important for its time to converge as its stopping criterion depends only on that. Therefore, if we increase the number of features but the repeatability remains the same, it does not benefit RANSAC and increases processing time (but sometimes we want more absolute number of inliers as with that we can obtain more precise model).

For the *Arc 2* viewpoint the results are shown in Figure 5.8, measured values and detector thresholds are shown in Table C.2. For all detectors, with increasing number of features, the repeatability increases with exception of MSER detector where the repeatability remains the same. Comparing feature responses, the best results are obtained with Hessian based feature detectors.

In case of *linear path* viewpoint, the results are shown in 5.9, measured values and detector thresholds are shown in Table C.1. In this case the response function threshold has small effect on the detector performance, only in case of DoG and MSER features, with smaller response function threshold the repeatability decreases. In case of MSER, with small margin the stability function is more affected by noise which causes least stable estimation of region extent. With DoG features the repeatability decrease more than for LoG features which it approximates as with small thresholds the approximation error may have bigger influence.

Affine invariance usually make the results worse, which is similar to the results presented



(a) Approx+MSER (b) Harris-based

(c) Hessian-based (d) Legend

Figure 5.8: Detector repeatability in DTU Robot 3D benchmark as a function of average number of detected features per image when varying response function threshold (or min. margin for MSER). Repeatability was computed over first 10 scenes in *Arc 2* dataset for the viewpoint 25°. Detector thresholds are shown in Table C.2.

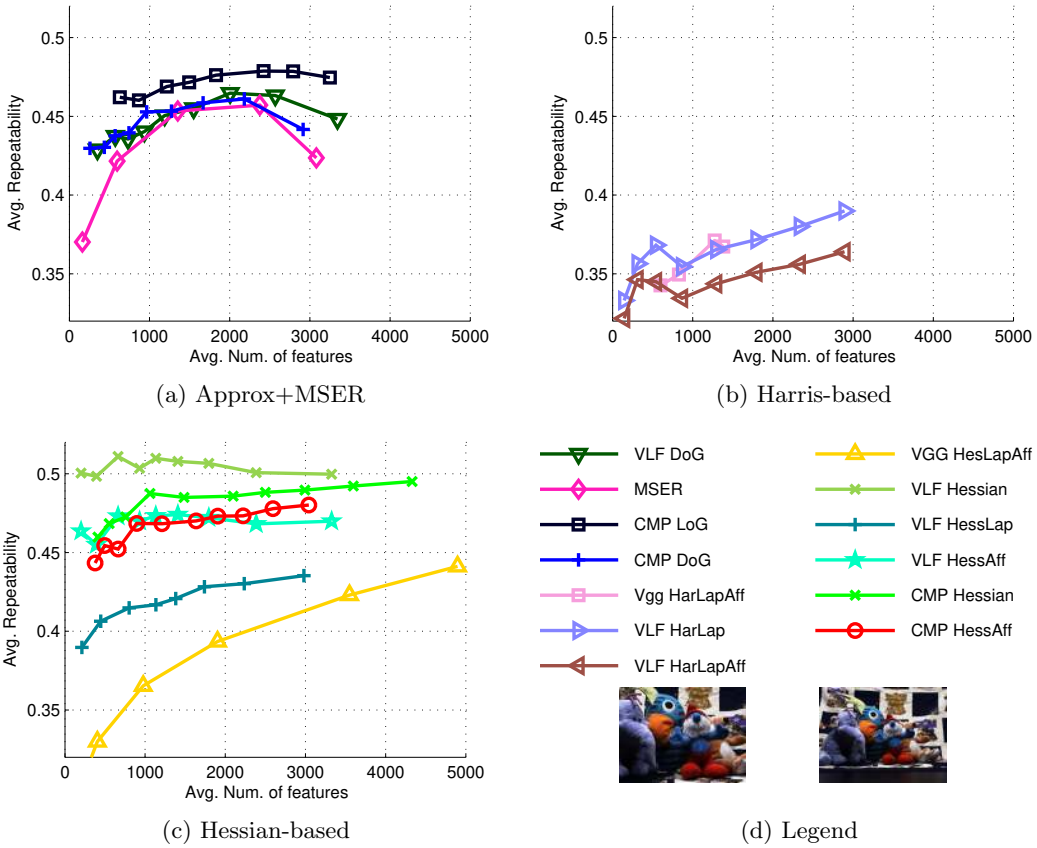
in [1] and is caused by rather small affine transformations between the tested images and additional degrees of freedom that need to be estimated by detector. From the implementation point of view, VGG detectors, used in detector comparison [1] obtained the worst performance.

Image retrieval The results of image retrieval experiment (mAP ,defined in 4.6) as a function of average number of features per image are shown in Figure 5.10, measured values and detector thresholds in Table . They are computed with the same detectors as in previous experiment but accompanied by SIFT descriptor. Descriptor algorithm was set to use measurement scale $\nu = 3$ and without orientation assignment. The tests were performed with the Oxford Buildings dataset [33].

From the results we can see that for all detectors there is a minimal number of features required to successfully cover object instances which explains small mAP for small number of features. When the number of features increases over 2000 the newly added features are not improving the results significantly, with exception of DoG features where with the lowest response function thresholds mAP decreases, similarly as in the epipolar geometry test.

From the invariance point of view it seems that the affine invariance improves the results slightly only for hybrid detectors. In case of Hessian detectors, affine shape adaptation seems only to decrease the number of features but does not the performance.

Contrary to epipolar geometry results, it seems that the VGG detectors give better results than other implementations.



(a) Approx+MSER (b) Harris-based

(c) Hessian-based (d) Legend

Figure 5.9: Detector repeatability in DTU Robot 3D benchmark as a function average number of detected features per image when varying response function threshold (or min. margin for MSER). Repeatability was computed over first 10 scenes in *linear-path* dataset where camera moves 0.3 metres away from the scene. Detector thresholds are shown in Table C.2.

Conclusions Our experiments show that for tasks, where the precise localisation of detected features is at stakes, detecting more features with lower threshold can increase detector performance. However, for DoG and MSER features there exists a limit where their scale invariance deteriorate. In image retrieval there is a limit after which the new regions does not improve retrieval system performance as the detected features become less distinctive.

5.3 Initial scale of scale space detectors

In this section we investigate performance of scale-space detectors dependent on initial scale σ_0 on the real scenes. This parameter sets the minimal scale of the features which can be detected and similarly as response function threshold it affects number of detected features.

In [24], this parameter has been studied in case of DoG feature detector. With our tests we broaden this study also to other feature responses and tested separately detector performance in DTU Robot 3D benchmark and in retrieval benchmark, configured in the same way as in previous section. Again, the detector performance is measured as a function number of detected features. Contrary to the precious section, experiments are performed only with CMP local feature detectors as they allow to set the value of σ_0 parameter.

The results for image retrieval (Figure 5.11c) suggest that the initial scale does not affect the performance. This is rather interesting observation as it means that features with small scales has only little impact on the image retrieval system performance. However this seems to be a consequence of the properties of the Oxford Buildings dataset where objects in query

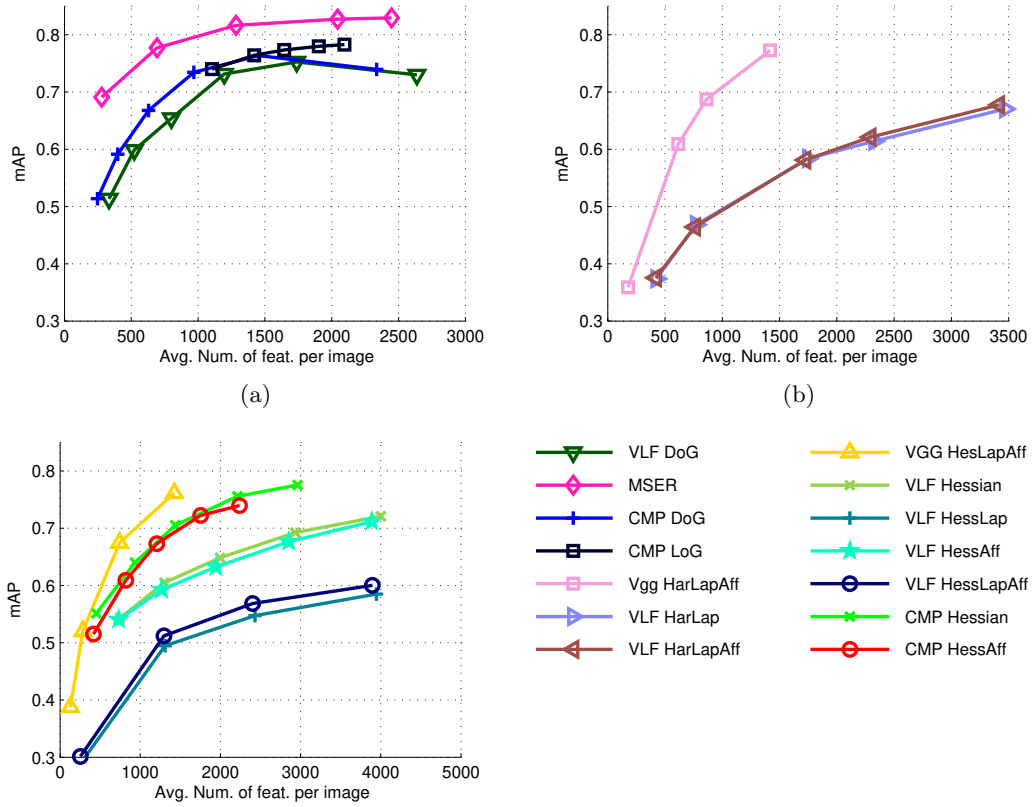


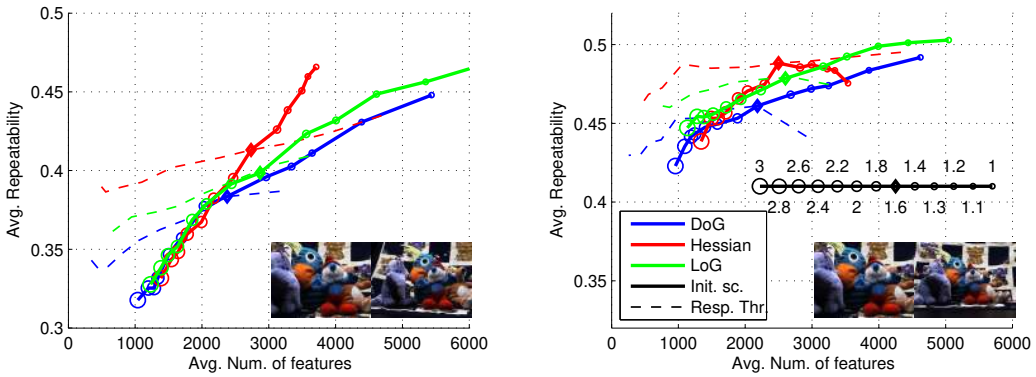
Figure 5.10: Detector mAP in Retrieval benchmark as a function average number of detected features per image when varying response function threshold (or min. margin for MSER). All descriptors has been computed using CMP SIFT implementation with measurement scale $\nu = 3$ and without orientation assignment. Results are computed on data from Oxford buildings dataset with a subset of 100 "Bad" images and $k = 50$. Detector thresholds in Table C.3.

and database images are usually of similar size. In cases where we do not demand robustness to big scale changes, the higher value of σ_0 can speed up the retrieval as removal of almost one half of the features decreased mAP only by few percents.

In the geometry precision experiment (DTU Robot 3D), the behaviour is different. In Figure 5.11a) and 5.11b) it can be seen that adding features with smaller scales improve the repeatability much more than including features with small feature response. This holds both for *Arc 2* and *Linear path* datasets with exception of Hessian detector where the scale invariance decreased performance of small features (may be caused by worse localisation of saddle points).

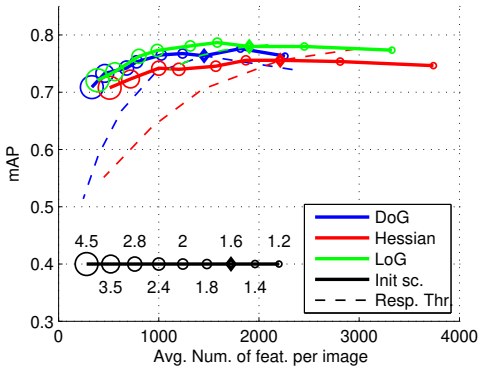
With higher values of the initial scale, the repeatability decreases much faster than for response function threshold. This is caused by fact, that all tested detectors localise blob-like features in a scale space pyramid, where with increasing scale the precision of localisation decreases.

Conclusions In some cases when the change of scale between two views is not overly extensive, it is worth to compute features of smaller scale than traditional $\sigma_n = 1.6$. They will not only improve repeatability but it may increase the precision as small blob features are better localised. In our experiments this increases detector performance more than adding features with lower response function threshold. However in case of retrieval, small features has little impact on detectors performance, especially in comparison to features added by



(a) DTU Arc 2

(b) DTU Linear path



(c) Retrieval

Figure 5.11: Detector performance in DTU Robot 3D benchmark and Retrieval benchmark plotted against average number of detected features per image when varying initial scale σ_0. In retrieval test, the values are computed with CMP SIFT descriptor, $\nu = 3$ and without orientation assignment. With the DTU robot dataset, repeatability is an average over 10 scenes. The size of the marker represents the initial scale and dashed lines are plotted results varying response function thresholds.

lowering response threshold.

5.4 Size and shape of the window function of measurement region

In this section we perform several experiments with the measurement region properties. Measurement region is a part of the image used for computation of a feature descriptor. We investigate how its size and shape influences the performance in case of SIFT descriptor.

Having detected features in the image, their neighbourhood is normalised into canonical coordinate system into a patch of constant size and used for computing descriptors. Usually the feature region scale is multiplied by a factor ν (measurement scale) in order to obtain feature's *measurement region*. This leads to more discriminative power [29, p. 24] as it includes more context around the feature location.

But setting the measurement region properly is a difficult task [50]. Setting it too small may cause problems for small image features as it covers only small parts of the image and is not sufficiently informative. Setting this region too big may cause problems with occlusions as the measurement region can overflow imaged object boundaries and descriptor is then calculated from values of its background. The background may significantly change with a different viewpoint. Apart from that, with bigger measurement region we lose the details as

it is always re-sampled into a patch of constant scale. Additionally, with bigger measurement region some regions get out of the image borders and also patch extraction takes more computational effort (in case of its normalisation).

In the following, we explore the information entropy in the measurement region for different type of features. Then, we examine properties of some existing detectors as even the scale of detected features differ by detector implementations. Besides the extent of the measurement region, for SIFT descriptor the extracted patch is also weighted by a Gaussian and we explore what effect does this shape have on its performance.

5.4.1 Image patch entropy

How much information we need to describe the measurement region? To answer this question we have computed information entropy of image intensity values for each pixel of a set of extracted patches. This quantity represents the expected number of bits needed to unambiguously encode the information in each pixels. We would like to see how this value evolves depending on the distance to the local image feature centre.

For set of patches $\mathcal{P}$ and a domain of possible brightness intensities $\mathcal{I} = \{0 \dots 255\}$, probability of a pixel on position $\mathbf{x}$ having an intensity $i \in \mathcal{I}$ is defined as:

$$\mathrm{pr}(\mathbf{x}, i) = \frac{\sum_{P \in \mathcal{P}} [P(\mathbf{x}) = i]}{|\mathcal{P}|} \tag{5.2}$$

Where $[\cdot]$ stands for the indicator function. Entropy of a pixel at position $\mathbf{x}$ is computed as:

$$H(\mathbf{x}) = \sum_{i \in \mathcal{I}} -\log_2(\mathrm{pr}(\mathbf{x}, i)) \cdot \mathrm{pr}(\mathbf{x}, i) \tag{5.3}$$

For computation of the entropy we have used similarity invariant patches extracted from 1000 images from *Distraction* subset of the Oxford Buildings retrieval dataset. Because image brightness intensities differ with the type of the interest point (LoG+ has usually brighter centre than LoG-) we have computed the entropies splitting the features by their type. Then, as we perform all experiments with the SIFT descriptor, we also compute pixel-wise entropies of image gradient magnitudes and angles where their values are normalised into 255 bins evenly covering their domain. Entropies of patches of some selected feature types are given in table 5.1 where the *dark* versions of local image features are selected (entropies of bright features look similar, only rotated by 180°). Measurement region of these patches is extracted for $\nu = 5$ and the patch size is set to 69 in order to obtain the same discrete derivatives as for standard SIFT measurement scale $\nu = 3$ and patch size 41.

In Table 5.1 it can be seen that for all local image features area with the lowest entropy is the detected interest region itself. This is because for the dark features the frame centre is usually dark and with a positive gradient in direction outward of the frame centre. This justifies why to set $\nu > 1$. But the centre is still quite interesting as it defines whether it is a dark of bright feature and it would theoretically add one more bit of entropy to its values.

As the regions size increases, image data has rather constant image entropy. In a way this does not give us any upper bound on measurement scale and generally shows that the context of the image region contains the majority of the information which makes the descriptor distinctive.

5.4.2 Scale of image frames

During our first tests with measurement region we have observed that different feature detector implementations detect different scales of the same image structures. This generally has two causes. At first, it is the response function which is used for detecting the feature scale (usually LoG, DoG or Hessian). But nevertheless it also differs by the particular implementation (e.g. their Gaussian kernel approximations etc.).

That is why we have decided to investigate detector behaviour more closely. As a testing feature we have chosen 2D Gaussian function (Equation 2.4) and as an etalon of its scale we

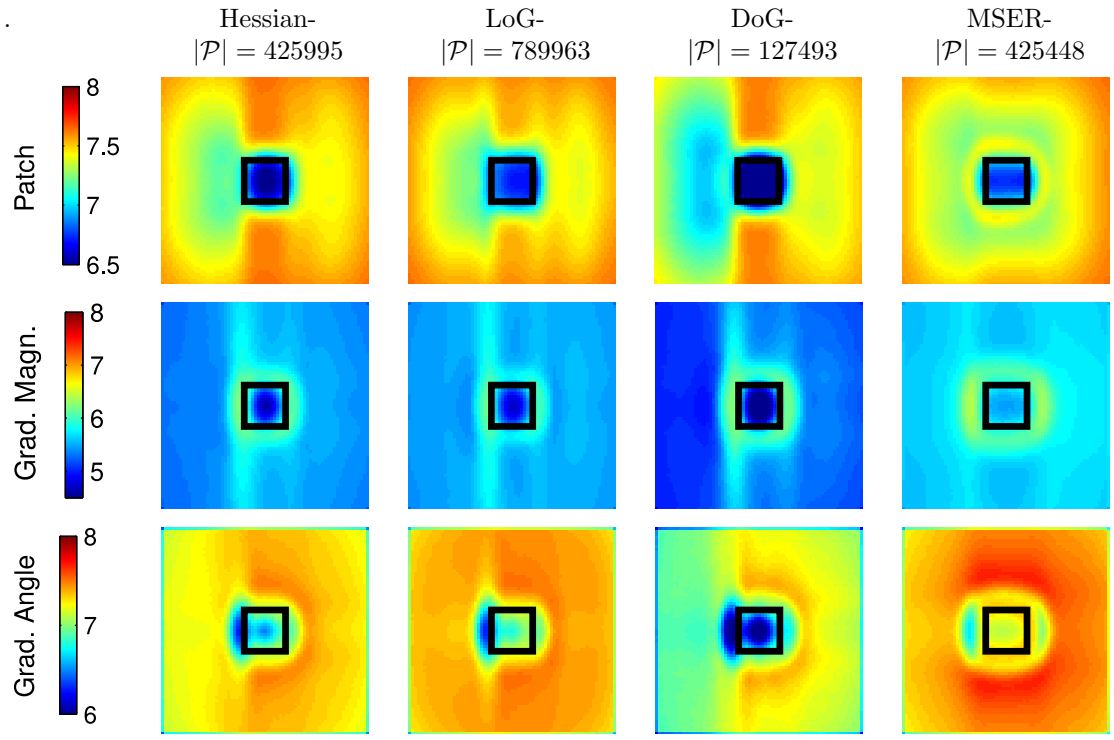
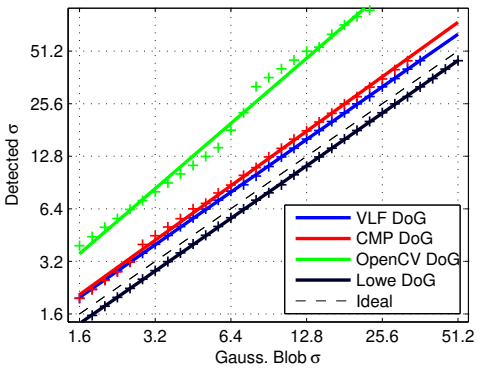


Table 5.1: Pixel-wise entropy of neighbourhood of selected local image features. Orientation of the patch has been normalised using the maxima in gradient orientation histogram (see 2.2.5 for details, this is a reason for entropy asymmetry). The visualised regions correspond to measurement region $\nu = 5$ and size 91×91 pixels. The black box inside a patch correspond to measurement region $\nu = 1$.

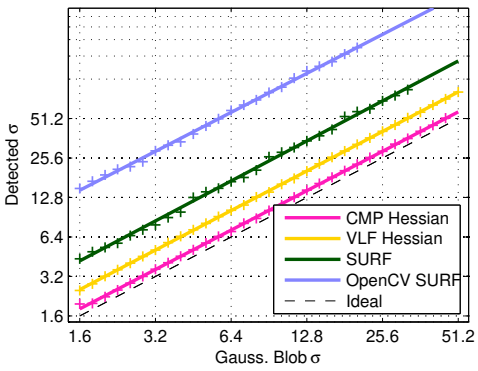
took its standard deviation σ. Than for each detector's detected features we pick the feature with a centre closest to the Gaussian blob centre and we take detected scale σ_{det} of this feature. We have computed these values for a set of standard deviations $\Sigma = \{\sigma | \sigma = 1.6 \cdot 2^{\frac{i}{6}}, i \in 1, .., 28\}$ which correspond to Lowe's pyramid with 6 layers per octave. Images with the Gaussian function has been generated with cut-off= 8 to prevent any border effects. Then in a log space we have fitted lines through the measured values using RANSAC and linear least squares line fitting in order to obtain more robust results. Measured values and fitted lines for selected detectors are shown in Figure 5.12.

From Figure 5.12 it can be seen that the detected scales differ significantly and there is small correlation between detectors using same quantities for scale selection. However the detectors which differ the most are OpenCV implementations where for their SURF detector for unknown reason the frame scale is 8-times bigger than the Gaussian standard deviation. Also in some cases the fitted line did not go through zero which means that there is some *offset* in the detected scale. This may cause problems with detector scale invariancy. Numerical values are given in table 5.2, offset is expressed as an intersection of the fitted line with x-axis.

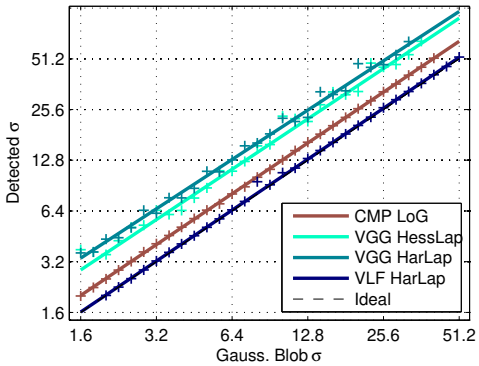
This shows that a fair detector comparison is rather difficult as the scale of detected local image features differ by implementation. As it will be seen in the further experiments, the measurement region influence detector performance in most of descriptor tests. This is a reason why we have decided to measure the ideal measurement scale quantitatively using proposed benchmarks mainly for CMP implementation of detectors which seem to share similar properties.



(a) DoG-based scale estimation



(b) Hessian-based scale estimation



(c) LoG-based scale estimation

Figure 5.12: Results of scale calibration of selected detectors. Cross markers represents separate measurements and the and solid line is line fitted using RANSAC and least squares optimisation on inliers using line to point distance in logarithmic coordinates.

5.4.3 Shape of the window function of measurement region

Besides the extent of measurement region, in case of SIFT descriptor the extracted patch is usually weighted by a Gaussian with σ equal to one half the width of the patch window to assign a weight to the magnitude of each sample point [24, p. 15]. Lowe claims that purpose of this weighting is to avoid sudden changes in the descriptor with small changes of a position of image feature centre and to give less weight to gradients which are further away. This weighting function is visualised in Figure 5.3a.

From the source code of the publicly available CMP HessianAffine detector[2] we have observed that also another weighting window is used. It is basically the same Gaussian function, but weight of pixels farther away from the patch centre than is the patch radius are set to zero, probably to increase rotation invariance. This weighting is visualised in Figure 5.3b and will be further referred as *circular weighting*. The third weighting which we use in our experiments is no weighting, i.e. to use directly the computed gradient magnitudes.

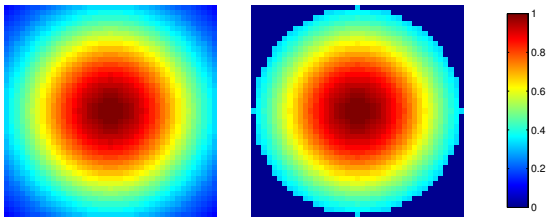
Additionally to the extent of measurement region we would like to test the influence of patch weighting to the descriptor performance. For this purpose we have extended the CMP SIFT descriptor implementation with an option to set any patch weighting (shape). In the following text we investigate influence of measurement region properties to epipolar geometry tasks with epipolar geometry benchmark, then to retrieval tasks with the retrieval benchmark.

Epipolar geometry This test is carried out with the proposed epipolar benchmark (section 4.3) as it allows to see behaviour of a detector and descriptor on a particular wide baseline

[2] `https://github.com/perdoch/hesaff`

Det.	σ_{det}/σ	offset	Det.	σ_{det}/σ	offset
VLF DoG	1.25	0	SURF	2.75	0.07
CMP DoG	1.47	0.16	OpenCV SURF	8.68	0.07
OpenCV DoG	5.18	0.9	CMP LoG	1.27	0.01
Lowe DoG	0.88	0	VGG HessLap	1.74	0.04
CMP Hessian	1.12	0	VGG HarLap	1.9	0.15
VLF Hessian	1.6	0.01	VLF HarLap	1.02	0.02

Table 5.2: Ascertained relations between 2D Guassian standard deviation and detected scales by selected detectors. These values has been computed by fitting a line and offset is the intersection of fitted line with x-axis.



Gaussian Circular

Table 5.3: Examined patch weighting for descriptor calculation.

scene. For all cases we have computed frame orientations and the measured quantity is the maximum precision of the nearest-neighbour classifier introduced in chapter 3, section 3.2, i.e. ratio of correct nearest neighbour matches to the number of detected features in the reference image. Results are shown in Figure 5.14.

In the results we can see one prevailing pattern which holds for all tested detectors. When a detector performance rises with increasing ν, computing descriptor without weighting seems to give the best results. Even in the case of significant scene rotation (Box, Valbonne), difference between the weightings is minimal. But increasing ν the number of correct matches decreases, the best weighting seems to be the circular window as it removes influence of image values more distant from the local feature centre.

From the perspective of measurement region scale it seems that for blob-like features, bigger ν rarely decreases the performance and if so, then only slightly in case of scenes with significant occlusions (Leafs). In all cases the precision is monotone for $\sigma < 5$. Exceptions are Harris and MSER features where the performance for non-planar scenes usually decreases for $\nu > 5$. It can be caused by a fact that MSER features are usually already of twice their usual scale (we maintained the same setting as was used in [29]).

Image retrieval Second test with measurement scale is image retrieval benchmark. Again we have varied the measurement scale and weighting. The results are shown in Figure 5.13.

Results shown in Figure 5.13 comply with the results in epipolar geometry. It seems that for all blob detectors, increasing measurement region increases also descriptor performance. Also as ν increases, circular weighting saturates later than no weighting variant. However it has to be said that this dataset contains little rotations and generally buildings has planar surfaces.

Again, Harris and MSER detectors are behaving in a different way. In case of Harris detector its maxima is gained with $\nu = 3$ but in case of MSER detector, smaller ν seems generally better. Again we can see that cropping the patch lobes with circular weighting can improve the results when the measurement region is too big.

The results presented in Figure 5.13 are computed assigning frame orientations. We have computed the same graphs also with upright features but the shape of the curves remained

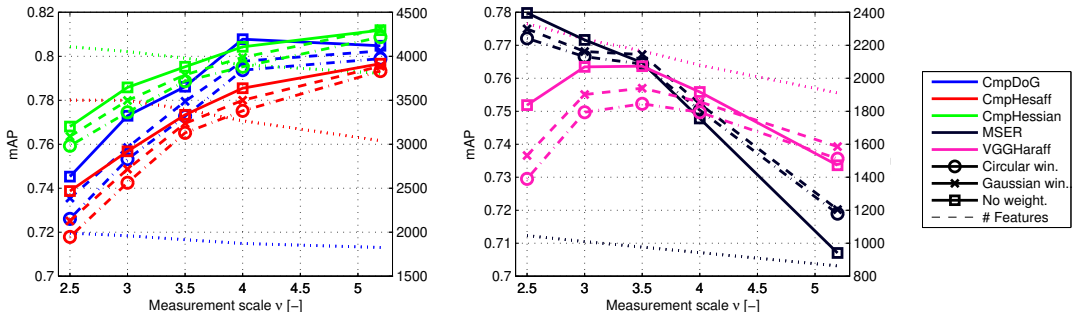
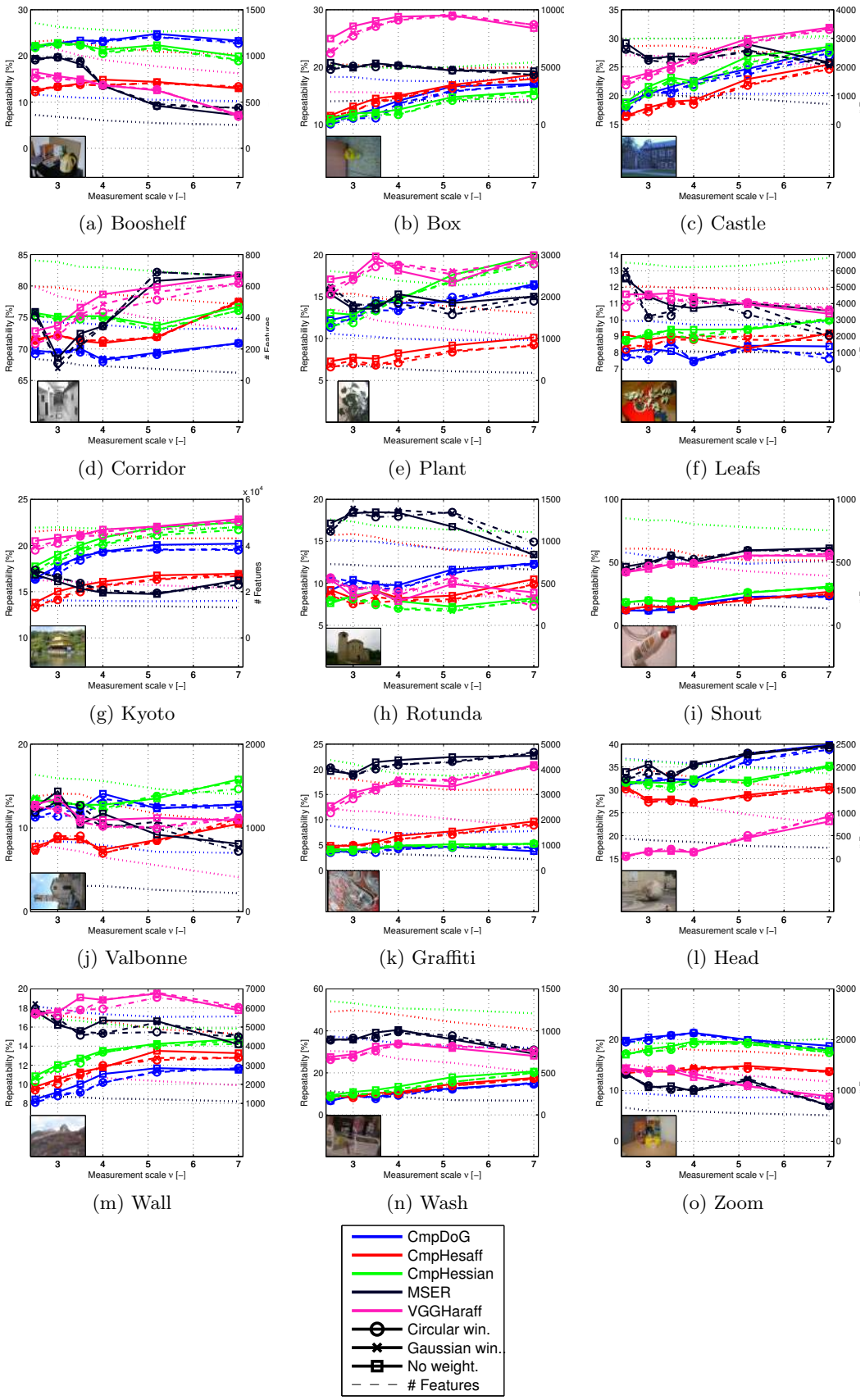


Figure 5.13: Results on retrieval benchmark using different measurement regions and different weighting functions. Right y-axis represent average number of extracted features per image (visualised as dashed line per detector). Values are computed using CMP SIFT descriptor implementation. Orientation of the patch is assigned with method presented by [24], max. 4 orientations per feature.

the same, only mAP has increased slightly.

Conclusions We have shown that from the information theory point of view, the least information is contained inside the detected local image feature for all examined local feature types. Therefore it is important to include sufficient context by increasing its scale to obtain distinctive measurement region. We have shown that various detector implementations differ in the detected scale for a Gaussian blob and therefore it is a difficult task to fairly compare them. With the experiments we have shown that for blob-like local features, increasing the measurement region size generally improve its performance and when no rotation of image data is expected weighting of the measurement region is not necessary. For Harris and MSER detector the size of the measurement region is more limited and the best results are usually obtained with measurement scale $\nu = 3$.



(a) Booshelf (b) Box (c) Castle

(d) Corridor (e) Plant (f) Leafs

(g) Kyoto (h) Rotunda (i) Shout

(j) Valbonne (k) Graffiti (l) Head

(m) Wall (n) Wash (o) Zoom

Figure 5.14: Influence of measurement scale ν and measurement region weighting to the number of inliers in epipolar geometry on selected scenes. The Sampson error threshold used was set to the same value which was used for RANSAC inlier threshold when ground truth was computed.

Chapter 6

Improvements to emulated detectors

In this chapter experiments with emulated feature detectors are discussed. The emulated feature detectors has been described in section 2.5. In Šochman and Matas [38] two emulators are examined - Hessian-Laplace and Kadir saliency detector emulators. In our experiments we perform experiments with the emulated Hessian-Laplace and other blob-like image feature detectors, leaving the emulated Kadir-saliency detector aside. All experiments has been performed with the source code obtained from Šochman [36]. Training part of the classification framework is implemented in Matlab and the classification itself in C++. With this code we have been able, with minor differences, to reproduce results from [38].

Contributions presented in this work are:

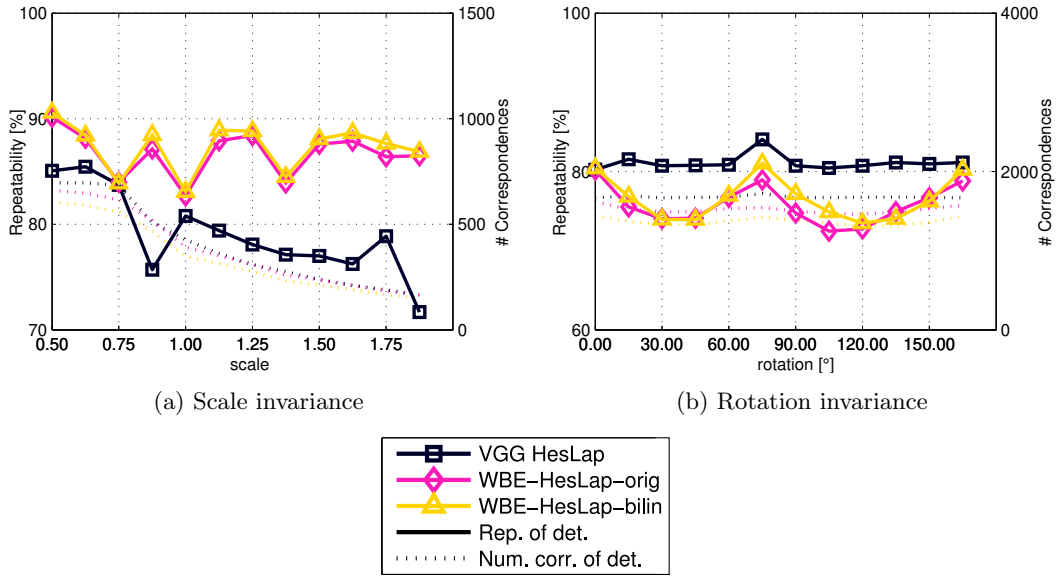
1. Improved the classification speed optimizing linearised code and non-maxima suppression
2. Trained emulators of different feature types (DoG and Hessian local features).
3. Measured performance of emulators in image retrieval and epipolar geometry tasks.
4. Improved Hessian-Laplace emulator performance in homography transformation based benchmarks.

First, some properties of the original Hessian-Laplace emulator are discussed and then we show some improvements to the training process. Second, we train emulators of different local image features and we show different properties of those emulators. Then we describe improvements which lead to faster classification. In the last part, performance of the emulators is evaluated in image retrieval and their geometric precision is measured with DTU Robot 3D benchmark.

6.1 Improvements of Hessian-Laplace emulated detector

Emulator is obtained by training the WaldBoost classifier with patches extracted by some local feature extraction algorithm, further referred as a teacher. The emulator presented in [38] was trained with features detected by VGG Hessian-Laplace detector implementation.

Training samples are small image patches of constant size extracted from the image where the local features has been detected. Region used for the patch is larger by a factor ν_{WB} (measurement scale) than the original feature scale in order to include some context of the detected feature. These regions are then transformed into a sample of constant size. In the original implementation, nearest-neighbour interpolation has been used. We have replaced it with bilinear interpolation which improved the results in homography based benchmarks. The improvements are shown in Figure 6.1 and are compared to original Hessian-Laplace emulator and the teacher in case of rotation and scale invariance. More results are shown in Appendix D in Figure D.2 for repeatability and Figure D.2 for matching score.



(a) Scale invariance

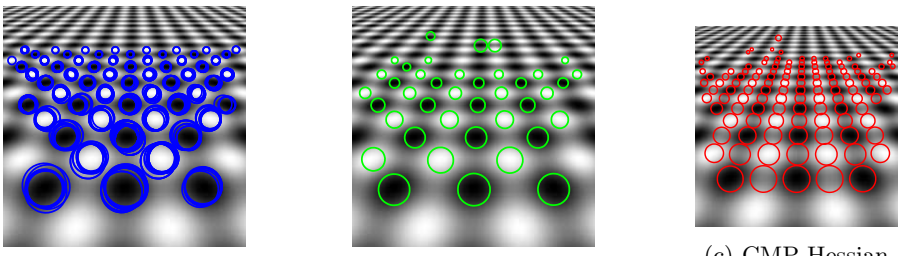
(b) Rotation invariance

Figure 6.1: Repeatability and number of correspondences of the original Hessian-Laplace emulator and Hessian-Laplace emulator with improved patch extraction using bilinear interpolation. Compared to the performance of the teacher on artificially rotated and scaled images. Repeatability is computed as an average over 7 images.

6.2 Emulation of different features

In this section we describe our experiments with emulators of different image features, such as DoG and Hessian. First, we notice some properties of the local image features used for training of the Hessian-Laplace detector from [38]. Then we explore possibilities to emulate detectors of different feature types.

In the original article [38], the Hessian-Laplace emulated detector is trained with features detected with the VGG implementation (introduced in Section 4.7.1). We have observed that this implementation does not detect saddle points which are expected for the Hessian feature response function (Equation 2.10). In Figure 6.2, detections of VGG Hessian-Laplace, CMP DoG and CMP Hessian are shown. It can be seen that VGG Hessian-Laplace detects local features in similar locations as CMP LoG and misses all saddle points detected by CMP Hessian detector. Please note that VGG Hessian-Laplace also returns multiple detections per maxima in the image brightness function.



(a) VGG Hessian-Laplace

(b) CMP LoG

(c) CMP Hessian

Figure 6.2: Detected frames by VGG Hessian-Laplace, CMP LoG and CMP Hessian detectors. It can be seen that VGG Hessian-Laplace misses all saddle points.

We have included support for training with any local image features detected by the CMP detector implementations. We have decided to test the framework with Hessian features including saddle points and with DoG features. In figure 6.2, it can be seen that features

detected on a simple image brightness structures are different. For both the emulators we extracted $2 \cdot 10^5$ features from *Distractor* images of Paris buildings retrieval dataset. Each weak classifier has been trained with $|\mathcal{T}| = 10,000$. In the original detector, emulator is learnt with patches extracted with measurement scale $\nu_{WB} = 2$. We have set the measurement scale for CMP detections to $\nu_{WB} = 3.5$, coarsely based on the observations made in Section 5.4.2. DoG features used for training has been detected with CMP DoG detector response threshold 10, Hessian features with threshold 20. These values has been selected empirically, based on the repeatability score, however we did not have enough computational resources to perform extensive experiments with other thresholds (training of one classifier takes around 12 hours).

In table 6.1 are shown the first Haar features of trained emulators. These features usually give us some intuition how the detected feature looks like as the first feature should reject the most of false samples. It is interesting to see, that for the Hessian emulated detector, the centre surrounded Haar feature was not picked as the first feature. From the experiments we have found that this is caused by saddle points. However this has big impact on both the detector speed (on average Hessian emulator evaluates 2.12 features per window whereas Blob emulators evaluates around 1.3 features per window).

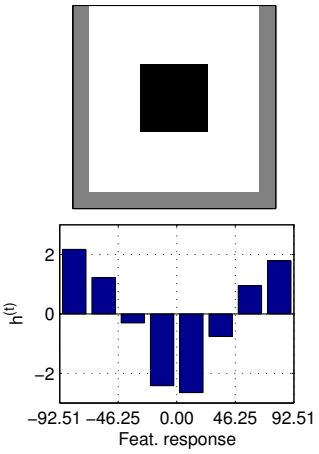
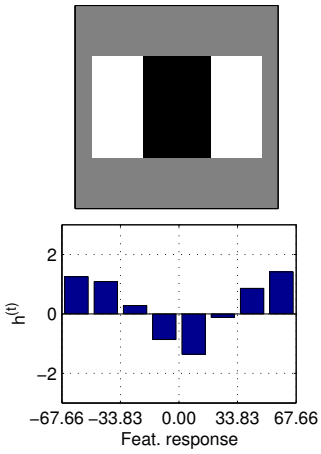
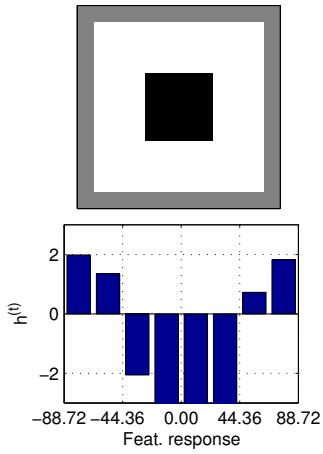
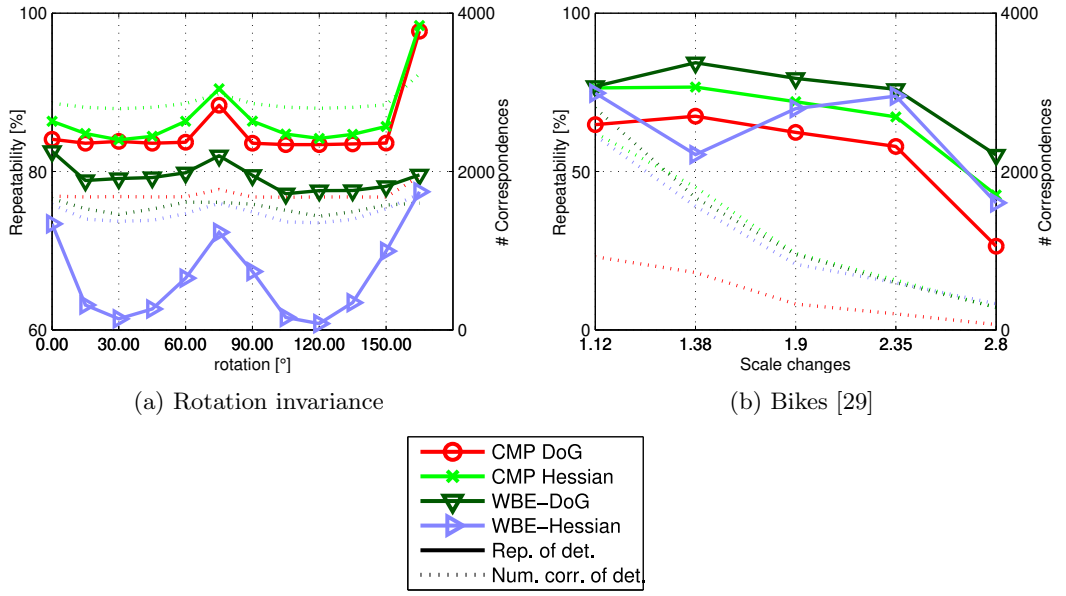
Orig. VGG HesLap 1.388 #F/Win	CMP Hessian 2.12 #F/Win	CMP DoG 1.234 #F/Win
		

Table 6.1: First weak classifiers of trained emulators with the average number of evaluated features per window. The average number of evaluated features is computed on a 21MPx image.

In table 6.1, it can be seen that the first feature of DoG emulator is located more in the patch centre. Also the average number of evaluated features per image decreased slightly. During development we have tested the detector performance with classical repeatability test. The results of rotation invariance dataset and *boat* dataset from [29] are shown in Figure 6.3. In can be seen that the rotation invariance of the Hessian emulated detector is much worse compare to the emulated DoG detector. Repeatability and matching score tested on other datasets are shown in Appendix D in Figure D.3 and Figure D.4.

6.3 Speeding up the classification

In the section we describe modifications which lead to shorter classification time. The original implementation is able to generate linearised code of the classifier. It leads to shorter classification time because it misses the overhead of complicated class structure of the classifier. Besides that, the linearised code is easier to optimise by the compiler as it exposes all mathematical operations performed by the classifier and therefore its memory operations are more transparent to the compiler.



(a) Rotation invariance

(b) Bikes [29]

Figure 6.3: Repeatability and number of correspondences of the new DoG and Hessian detector emulators. Compared to the performance of their teachers. Rotations invariance is computed as with artificially rotated images as an average over 7 images.

Profiling the code, we have improved the classification speed even further. The most time consuming parts were response calculation in the linear code and non-maxima suppression. The changes we have made are:

- Linearised code of the classifiers computes in single precision instead of double precision.
- Improved memory allocation in non-maxima-suppression so that the needed data are allocated ahead, not on demand. This lead to fewer calls of memory allocator.
- Because the standard deviation is not used for normalisation of Haar responses (as we are not interested in low-contrast regions) we have removed it completely from the code as it brings overhead in computation of another integral image.
- Non-maxima suppression is performed in single precision.
- Changes to the function for two circles overlap approximation - when the circle centres are farther away than the sum of their radii, the circles cannot overlap.

Speed is compared with SURF detector [5] (in particular, OpenSURF[1] open source implementation) and VLFeat DoG detector in a configuration that response is sampled with step of 2 pixels, which is a default setting for the original SURF detector. For DoG detector it means that the responses are computed from the second octave. VLFeat DoG implementation has been chosen as it allows to vary this parameter. In all cases the detectors has been configured to detect similar number of features and time values are averaged over 16 images of various scenes. The speed improvements are summarised in table 6.2.

For the 21 Mpx images, the improved emulators are almost 3-times faster then the original version and they are more than 2-times faster than SURF detector. In addition, the speed of the improved emulators varies less for different images, which is closer to the behaviour of DoG and SURF detectors. This has been achieved thanks to improvements in non-maxima suppression where the computation time depends on the number of detected features.

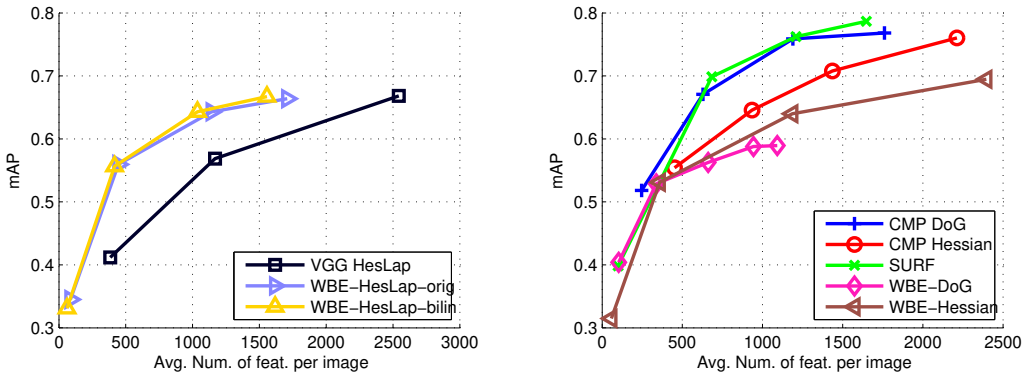
[1] `http://www.chrisevansdev.com/computer-vision-opensurf.html`

Detector	Image 640 × 480 (0.3MPx)		Image 3779 × 5669 (21MPx)	
	Avg. time [ms]	Avg. # feat	Avg. time [s]	Avg. # feat
OpenSURF	35 ± 1.5	120 ± 111	2.7 ± 0.11	3848 ± 4698
VLFeat DoG	42 ± 2	107 ± 64	4.5 ± 0.2	3367 ± 3028
WBE HesLap orig.	19 ± 12	91 ± 59	3.7 ± 2	3111 ± 3093
WBE HesLap fast	16 ± 4.4	91 ± 59	1.3 ± 0.31	3111 ± 3093
WBE DoG	15 ± 3.2	88 ± 54	1.3 ± 0.33	2748 ± 2631
WBE Hessian	20 ± 6.9	90 ± 59	1.4 ± 0.49	3002 ± 3006

Table 6.2: Processing speed of SURF, DoG, original emulator and the improved emulators with faster classification time. Processing time is averaged over 100 measurements and is shown together with the number of features. Detectors are configured to process only each second pixel position and with thresholds to obtain similar number of features. Results computed as average over 16 images with different scenes.

6.4 Emulator in the wild

Main design advantage of the emulators is their fast feature detection. However with an disadvantage that it only emulates the feature detectors and makes a decision on rather crude features. But how much these approximations affect the performance of the detector in real-world tasks? This question we would like to answer using previously introduces benchmarks, Retrieval and DTU Robot 3D benchmark (see Chapter 4).



(a) Hessian-Laplace emulators

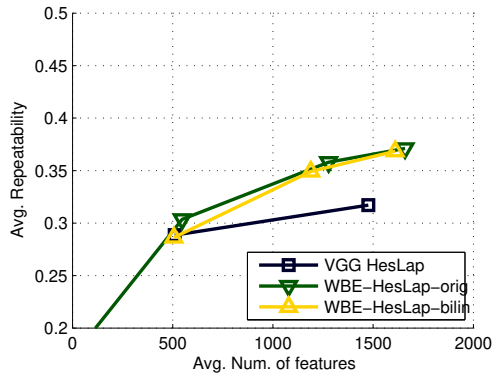
(b) DoG and Hessian emulators

Figure 6.4: Performances of the emulated detectors and their teachers varying their response function threshold or minimal confidence for emulated detectors (WBE), mAP measured on Oxford buildings dataset. The values are computed with CMP SIFT descriptor, $\nu = 3$ and without orientation assignment.

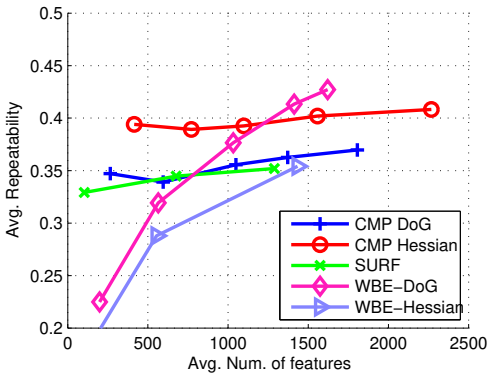
Because setting a response function threshold (or minimal confidence for the emulators) is not a simple task, we have performed similar tests as in Section 5.2, which deals with response function thresholds of common feature detectors.

In the Retrieval benchmark (Figure 6.4), newly trained Hessian-Laplace emulator has generally the same performance as the original Hessian-Laplace emulator. It is interesting to see that the emulators over-perform their teacher, VGG Hessian Laplace. However, emulated Hessian and DoG feature detectors perform much worse in this task compare to their teachers.

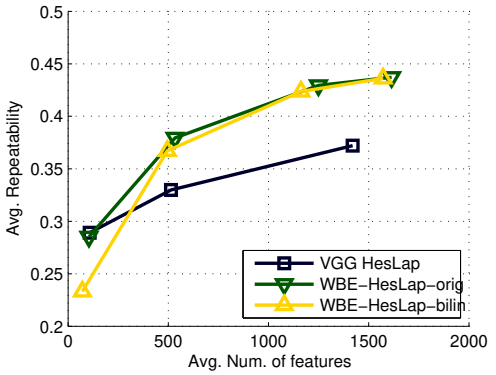
Next set of tests with the minimal confidence and feature response thresholds has been made with DTU Robot 3D benchmark which tests precision of feature localisation. Results are shown in Figure 6.5. Again, Hessian-Laplace emulators over-performs their teacher and newly-trained Hessian-Laplace emulator has got similar results as the original emulator, contrary to its improved performance on planar scenes.



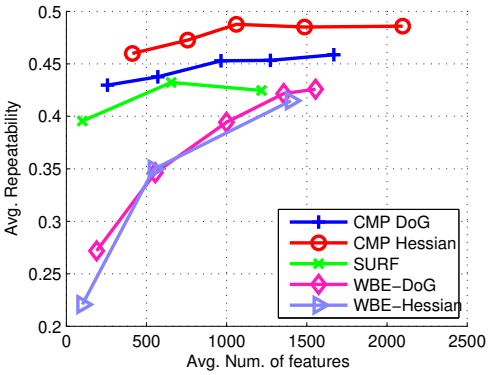
(a) Hessian-Laplace emulators, DTU *Arc 2*



(b) DoG and Hessian emulators, DTU *Arc 2*



(c) Hessian-Laplace emulators, DTU *Linear Path*



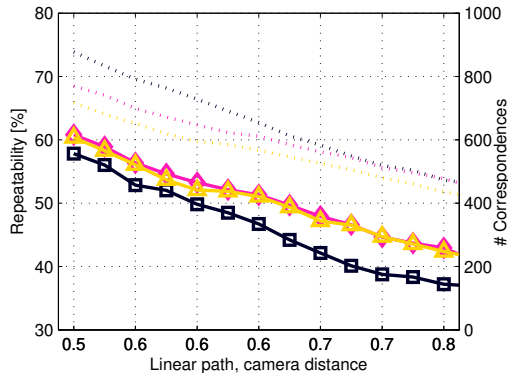
(d) DoG and Hessian emulators, DTU *Linear Path*

Figure 6.5: Performances of the emulated detectors and their teachers varying their response function threshold or minimal confidence for emulated detectors (WBE). Repeatability computed with the DTU Robot 3D dataset, repeatability is an average over 10 scenes. Tested camera viewpoint for the *Arc 2* path is 25° and 0.8m for the *Linear path*.

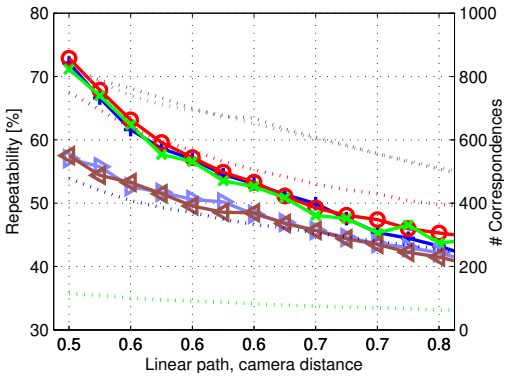
For the DoG and Hessian emulators, their teachers have better performance mainly for the *Linear path*, showing that the scale invariance of the emulators is worse than their teachers. However, for the DoG emulator with the lowest confidence threshold and *Arc 2*, it has better repeatability than any tested algorithm in this chapter. In order to investigate these results more, we have measured detector repeatability for all viewpoints in the DTU Robot 3D benchmark. The results are shown in Figure 6.6. From the results measured on *Linear path* it seems that the new emulators has worse scale invariance than their teachers and the Hessian-Laplace emulators. But in the *Arc 2* viewpoint, DoG emulator still outperforms other detectors for viewpoints more distant from the reference frame. This is even more interesting considering that the emulator does not have any sub-pixel localisation. We have not found what is the reason of this behaviour so this remains an open question.

When comparing the emulated detectors with SURF detector, SURF has got generally better scale invariance and has similar performance as CMP DoG detector in both geometric precision and in image retrieval task. Especially in the image retrieval it clearly outperforms the emulated detectors. The worse scale invariance of the emulators may be caused by missing sub-scale localisation.

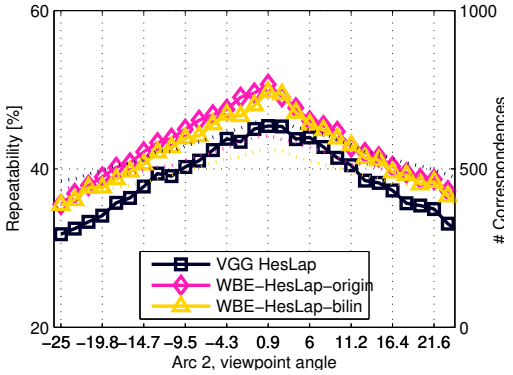
.



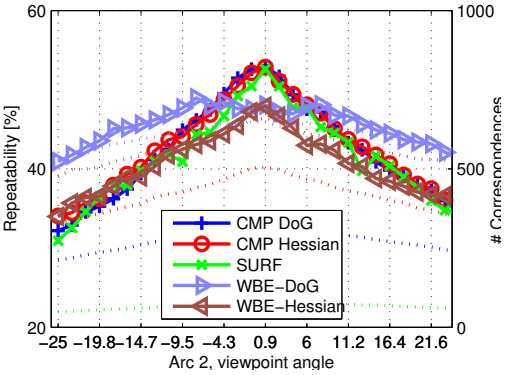
(a) Hessian-Laplace emulators, DTU Linear Path



(b) DoG and Hessian emulators, DTU Linear Path



(c) Hessian-Laplace emulators, DTU Arc 2



(d) DoG and Hessian emulators, DTU Arc 2

Figure 6.6: Performances of the emulators their teachers in DTU Robot benchmark. Repeatability is an average over first 10 scenes. Dashed lines are number of correspondences.

6.5 Dead ends

We have explored several ways which promised to give some improvements either in speed or in performance of the emulated detectors. However many of them did not lead to any significant improvements. Among these experiments were:

- Use of SURF-like integer scale pyramid [5] with its $3 \times 3 \times 3$-neighbourhood non-maxima suppression using confidence as a feature response. Motivation for this was mainly a computational complexity as the non-maxima suppression used in the current implementation is rather high. However using this pyramid leads to rapid decrease in the repeatability even when the same scale-sampling has been used.
- Artificial rotation of image patches. This again leads to decrease in repeatability, both in rotation and scale invariance of the emulator.We have found out that the extracted patches itself are enough for sufficient rotation invariance.
- Randomly moving the patches in space or scale similarly as in Kálal [17] in order to bridge the spatial and scale-space sampling lead also to worse emulator performance.

6.6 Future work

The boosting process can be improved using some more robust optimisation techniques than the used in Real-Boost. Among them are techniques which emerged from the view on Boosting as a Gradient descent optimisation in function space [25] (AnyBoost). This observation allows

to optimise over different cost functions than just the negative exponential, e.g. standard logistic function which allows to use Boosting in multiple instance learning as in [45]. However we have not found solution to a problem how to extend the AnyBoost framework to the RealBoost way of domain partitioning, which are used, and are rather important part of the algorithm. We suppose that mainly the multiple-instance learning can bring some significant improvements as it also picks the best context of the features.

In the current state the training stage with $200 \cdot 10^3$ training samples and $5 \cdot 10^3$ training samples per weak classifier and the training process takes whole 12 hours. The training time makes any experiments rather difficult and improving this, for examples using similar improvements as in Kálal et al. [18] would be an useful addition to the emulators framework.

In this work we have not performed experiments with different response function thresholds of the teacher detectors. This may offer further improvements to the performance of DoG and Hessian emulators.

We have also been able to train emulator of Kadir saliency features. However we have not noticed any significant improvements to the emulator presented in [38]. Also mAP of the emulated Kadir saliency detector in the Retrieval benchmark was lower compared to other emulators. But these features may be complementary to the blob features which can be useful in some classification tasks.

Conclusions We have been able to speed up the classification time of the emulators so that the emulated detectors are faster than OpenSURF detector. Then we have trained emulators with new types of local image features, such as DoG and Hessian images features, and we have shown that the rotation invariance of the decreases when trained with anisotropic features such as saddle points. We have shown that the emulated feature detectors gain worse scale invariance than other scale space based image feature detectors and also has worse results in the image retrieval tasks. But surprisingly, in tasks where the scale of the scene does not change, emulated detectors gain comparable performance.

Chapter 7

Conclusions

The reliable and fair performance evaluation of the local feature detectors and descriptors is a difficult task that needs to take into account many applications and desired properties of the local features.The prospected framework started as VLBenchmarks project which intended to gather image feature detector benchmarks. We have finalised implementation of the detectors previously included in the project and included other important evaluation protocols traditionally used for feature detector and descriptor testing. Additionally we have proposed a new image retrieval benchmark which evaluates detectors in this particular application of local features on standard retrieval datasets. To asses the geometric precision we propose a simple epipolar geometry criterion based on Sampson's reprojection error which allows to test detectors in wide baseline stereo problems and extends the homography based benchmarks. Other improvements were merely technical such as inclusion of other homography datasets or including support for results caching, parallel processing and automatic installation process for several image feature detector and descriptor implementations. This project, with the improvements presented in this work, has also been presented in ECCV 2012 tutorial to the computer vision community.

We have proposed new algorithm for building a scale space pyramid for images with any nominal Gaussian blur. It shows, that when apriori known nominal image blur is compensated with nominal scale, it significantly improves repeatability of the scale-space based detectors. With an algorithm for image blur estimation this technique can be used to improve feature detection in blurred images.

A popular way how to tune detector performance is to adjust minimal threshold on the feature response function value as it has direct impact on the number of detected features. We have found that when the scale-space detector sought property is the geometric precision, better results can be gained by including smaller features. However, in image retrieval tasks, features with smaller response threshold are having higher influence on the performance. It has shown that when geometric precision is at stakes, CMP and VLFeat implementation gain better results and Kristian Mikolajczyk's detectors are more capable in image retrieval tasks.

We also measured the feature detector and descriptor performance w.r.t. to the size and shape of the window function of the feature's measurement region. We proposed a computation of the pixel-wise entropy of a patch, which revealed that the most informative part is the immediate neighbourhood of the feature and the feature centre is less informative. The comparison of detector implementations has shown to be a difficult task as there are several inconsistencies in feature scale detection. We have proposed a robust method to measure the relation between Gaussian blob standard deviation and the detected feature scale and have shown that for some implementations the reported scale is in some cases more than 8-times higher than the standard deviation. By testing several detector implementations we found that in tasks where the rotation invariance is not important, weighting the measurement region by a window function is not necessary. Ideal measurement region size differs for blob, corner and MSER-like features. For blob-like features, increasing the measurement region

improves the detector performance both in matching and image retrieval tasks but after certain size the performance saturates. On the other side, MSER and Harris detectors have a clear limit for the measurement region size.

In the last part of this work, we have presented several improvements to the emulated detectors. We have successfully trained emulators of different local feature detectors and improved the training process. In the case of the Difference of Gaussian feature emulator, it gains better geometric precision than the original Hessian Laplace emulated detector. Additionally, we have significantly reduced the classification time, and thus improved the main advantage of the emulators. With these improvements emulators became faster than the OpenSURF hand-crafted detector implementation.

Bibliography

[1] Henrik Aanæs, Anders Lindbjerg Dahl, and Kim Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97(1):18–35, 2012.

[2] Henrik Aanæs, Anders Dahl, and Kim Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97:18–35, 2012.

[3] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *IEEE Computer Vision and Pattern Recognition*, 2012.

[4] A. Baumberg. Reliable feature matching across widely separated views. In *IEEE Computer Vision and Pattern Recognition*, volume 1, pages 774–781, 2000.

[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[6] Matthew Brown and David Lowe. Invariant features from interest point groups. In *British Machine Vision Conference*, pages 656–665, 2002.

[7] Michael Calonder, Vincent Lepetit, Mustafa Ozuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. BRIEF: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:1281–1298, 2012.

[8] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized ransac. In *Pattern Recognition*, pages 236–243. Springer, 2003.

[9] Kai Cordes, Bodo Rosenhahn, and Jörn Ostermann. Increasing the accuracy of feature evaluation benchmarks using differential evolution. In *IEEE Symposium on Differential Evolution*, pages 1–8. IEEE, 2011.

[10] Bin Fan, Fuchao Wu, and Zhanyi Hu. Aggregating gradient distributions into intensity orders: A novel local image descriptor. In *IEEE Computer Vision and Pattern Recognition*, pages 2377–2384, 2011.

[11] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002. ISBN 0130851981.

[12] Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal of Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[13] Jonas Gårding and Tony Lindeberg. Direct computation of shape cues using scale-adapted spatial derivative operators. *International Journal of Computer Vision*, 17: 163–191, 1996.

[14] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge University Press, 2000.

[15] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Exploiting descriptor distances for precise image search. Research report, INRIA, June 2011.

[16] Timor Kadir and Michael Brady. Saliency, scale and image description. *International Journal of Computer Vision*, 45(2):83–105, 2001.

[17] Zdeněk Kálal. Face detection with Waldboost algorithm. Master's thesis, FEL CTU Prague, Prague, Czech Republic, 2007.

[18] Zdeněk Kálal, Jiří Matas, and Krystian Mikolajczyk. Weighted sampling for large-scale boosting. In *British Machine Vision Conference*, 2008.

[19] Karel Lebeda, Jiri Matas, and Ondrej Chum. Fixing the locally optimized RANSAC. In *British Machine Vision Conference*, 2012.

[20] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. BRISK: Binary robust invariant scalable keypoints. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2548–2555. IEEE, 2011.

[21] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers/Springer, 1994.

[22] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30:79–116, 1998.

[23] Tony Lindeberg. Generalized gaussian scale-space axiomatics comprising linear scale-space, affine scale-space and spatio-temporal scale-space. 2008. URL `http://www.csc.kth.se/~tony/abstracts/Lin10-GenGaussScSp.html`.

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

[25] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. Neural Information Processing Systems, 1999.

[26] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, pages 384–393, 2002.

[27] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *Proceedings of the 7th European Conference on Computer Vision*, pages 128–142, 2002.

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

[29] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1/2):43–72, 2005.

[30] Pierre Moreels and Pietro Perona. Evaluation of features detectors and descriptors based on 3D objects. *International Journal of Computer Vision*, 73(3):263–284, 2007.

[31] M. Perďoch, O. Chum, and J. Matas. Efficient representation of local geometry for large scale object retrieval. In *IEEE Computer Vision and Pattern Recognition*, pages 9–16. IEEE, 2009.

[32] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *IEEE Computer Vision and Pattern Recognitions*, pages 1–8, 2008.

[33] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[34] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443. May 2006.

[35] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[36] Jan Šochman. *Learning for Sequential Classification*. PhD thesis, FEL CTU Prague, Prague, Czech Republic, 2009.

[37] Jan Šochman and Jiří Matas. WaldBoost - learning for time constrained sequential detection. In *IEEE Computer Vision and Pattern Recognition*, volume 2, pages 150–157, 2005.

[38] Jan Šochman and Jiří Matas. Learning fast emulators of binary decision processes. *International Journal of Computer Vision*, 83(2):149–163, 2009.

[39] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[40] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2010.

[41] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

[42] A. Vedaldi, H. Ling, and S. Soatto. Knowing a good feature when you see it: Ground truth and methodology to evaluate local features for recognition. In *Computer Vision: Detection, Recognition and Reconstruction*, volume 285, pages 27–49. Springer, 2010.

[43] Andrea Vedaldi and Brian Fulkerson. VLFeat: an open and portable library of computer vision algorithms. In *Proceedings of the international conference on Multimedia*, pages 1469–1472, 2010.

[44] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

[45] Paul Viola, John Platt, Cha Zhang, et al. Multiple instance boosting for object detection. *Advances in neural information processing systems*, 18:1417–1425, 2006.

[46] Abraham Wald. *Sequential analysis*. Courier Dover Publications, 1947.

[47] Zhenhua Wang, Bin Fan, and Fuchao Wu. Local intensity order pattern for feature description. In *IEEE International Conference on Computer Vision*, pages 603–610, 2011.

[48] Simon Winder, Gang Hua, and Matthew Brown. Picking the best daisy. In *IEEE Computer Vision and Pattern Recognition*, pages 178–185. IEEE, 2009.

[49] Bin Fan Zhenhua Wang and Fuchao Wu. Local intensity order pattern for feature description. In *IEEE International Conference on Computer Vision*, pages 603–610, 2011.

[50] Jan Čech, Jiří Matas, and Michal Perdoch. Efficient sequential correspondence selection by cosegmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1568–1581, 2010.

Appendix A

Contents of Enclosed CD

Enclosed CD holds source code of VLBenchmarks project in its last public version and in version used for computing the presented results (as some of the features has not been yet released) together with recipes of the experiments performed in this work. Then it contains source code of feature emulators, source code of adapted CMP detector and source code of this thesis.

In the following table the directory structure of the CD is summarised:

Folder/File	Description
`src_vlb/`	Source code of VLBenchmarks
`src_emul/`	Source code of detector emulators
`src_cmp_det/`	Source code of CMP detector with improved pyramid building
`doc/`	Source code of this thesis
`thesis.pdf`	Text of this thesis

Table A.1: Directory structure of enclosed CD.

Appendix B

VLBenchmarks tutorials

In this appendix we present tutorials which were created for VLBenchmarks project[1]. Main purpose of these tutorial is to introduce the application programming interface of the project.

VLBechmarks is a MATLAB framework to evaluate feature detector and descriptors (further referred as feature extractors) automatically. Benchmarking your own features is as simple as writing a single wrapper class. Then *VLBenchmarks* takes care of downloading the required benchmarking data from the Internet and running the evaluation(s). The framework ships with wrappers for a number of publicly available features to enable comparing to them easily. *VLBenchmarks* has a number of functionalities, such as caching of intermediate results, that allow running benchmarks efficiently.

Repeatability benchmark

The repeatability benchmark reimplements the protocol introduced by Mikolajczyk et al. [29]. It defines two feature extractor tests. The first test measures the feature extractor repeatability. The repeatability measures to what extent do detected regions overlap exactly the same scene region by comparing detected features in two images of the same scene. It is based only on the feature geometry.

The second test computes the matching score that includes also the local features descriptors. The second test helps to asses detected regions distinctiveness in planar scenes.

In this tutorial, it is shown how to perform both tests together with visualisation of the computed scores.

Image features detection

VLBenchmarks contains a few built-in image feature extractors such as VLFeat SIFT, VLFeat MSER, VLFeat Covdet and a random features generator. Each feature extractor is represented by a MATLAB object which unifies the feature detection. All these feature extractors are implemented in a Matlab package `localFeatures`. For example, an instance of a class of VLFeat SIFT feature extractor can be obtained by:

```
sift = localFeatures.VlFeatSift() ;
```

The feature extractor object manages the values of feature extractor parameters. Default values are set in the constructor of the object however any parameter can be changed. For example we can create VLF-SIFT feature extractor with different peak threshold parameter.

```
thrSift = localFeatures.VlFeatSift('PeakThresh',11);
```

Let's generate a test image.

[1] `http://www.vlfeat.org/benchmarks`

```
ellBlobs = datasets.helpers.genEllipticBlobs();
ellBlobsPath = fullfile('data','ellBlobs.png');
imwrite(ellBlobs,ellBlobsPath);
```

To extract features from an image, each feature extractor implements method `extractFeatures`.

```
siftFrames = sift.extractFeatures(ellBlobsPath);
bigScaleSiftFrames = bigScaleSift.extractFeatures(ellBlobsPath);
```

The detected features can be visualises with their regions using `vl_plotframe` function.

```
imshow(ellBlobs);
sfH = vl_plotframe(siftFrames,'g');
bssfH = vl_plotframe(bigScaleSiftFrames,'r');
legend([sfH bssfH],'SIFT','Big Scale SIFT');
```

The detected frames are visualised in Figure B.1.

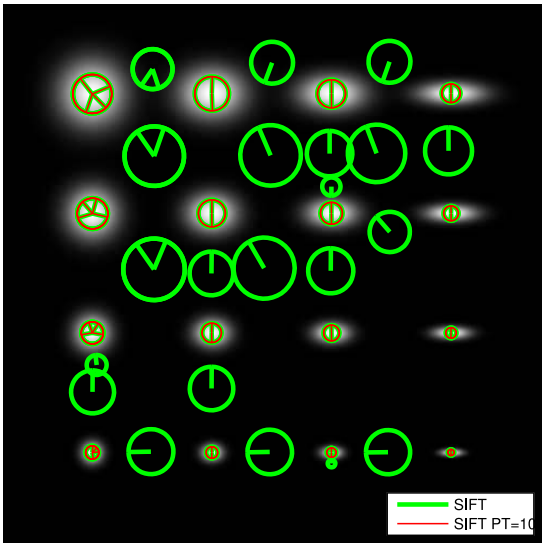


Figure B.1: SIFT frames detected on a test image using different peak threshold.

The feature extractors cache the detected features in a cache, thus when you run `extractFeatures` again, features are loaded from the cache. You can disable caching by calling feature extractor method `obj.disableCaching()`.

Repeatability test

The feature extractor repeatability is calculated for two sets of feature frames FRAMESA and FRAMESB detected in a reference image IMAGEA and a second image IMAGEB. The two images are assumed to be related by a known homography H, mapping pixels in the domain of IMAGEA to pixels in the domain of IMAGEB. The test assumes static camera, no parallax, or moving camera looking at a flat scene.

A perfect co-variant feature extractor would detect the same features in both images regardless of a change in viewpoint (for the features that are visible in both cases). A good feature extractor will also be robust to noise and other distortion. The repeatability is the percentage of detected features that survive a viewpoint change or some other transformation or disturbance in going from IMAGEA to IMAGEB and is calculated only based on the frames overlap. For detail about this test see [29].

For measuring feature extractors repeatability there is a class `RepeatabilityBenchmarks()`. To measure the repeatability as it is defined in [29] the benchmark object needs the following configuration:

```
import benchmarks.*;
repBenchmark = RepeatabilityBenchmark('Mode','Repeatability');
```

To test a feature extractor, benchmark object has a method —testFeatureExtractor—. The remaining parameters can be obtained from the VGG Affine dataset class which contains sets of six images with known homographies. Let's take the graffiti scene:

```
dataset = datasets.VggAffineDataset('Category','graf');
```

Now we define set of feature extractors which we want to test.

```
mser = localFeatures.VlFeatMser();
featureExtractors = {sift, thrSift, mser};
```

And finally loop over the feature extractors and selected images.

```
imageAPath = dataset.getImagePath(1);
for detIdx = 1:numel(featureExtractors)
  featExtractor = featureExtractors{detIdx};
  for imgIdx = 2:dataset.numImages
    imageBPath = dataset.getImagePath(imgIdx);
    tf = dataset.getTransformation(imgIdx);
    [rep(detIdx,imgIdx) numCorr(detIdx,imgIdx)] = ...
      repBenchmark.testFeatureExtractor(featExtractor, tf, ...
        imageAPath,imageBPath);
  end
end
```

This loop can be easily executed in parallel using parfor. Computed results are usually plotted in a graph showing together repeatability and number of correspondences.

```
detNames = {'SIFT','SIFT PT=10','MSER'};
plot(rep'.*100,'LineWidth',2); legend(detNames);
...
plot(numCorr','LineWidth',2); legend(detNames);
...
```

The matching score and number of correspondences is shown in Figure B.2.

Displaying the correspondences

It is useful to see the feature frames correspondences. Let's see what correspondences have been found between the features detected by VLF-SIFT feature extractor in the first and the third image. We can get the cropped and reprojected features and the correspondences itself by:

```
imgBIdx = 3;
imageBPath = dataset.getImagePath(imgBIdx);
tf = dataset.getTransformation(imgBIdx);
[r nc siftCorresps siftReprojFrames] = ...
  repBenchmark.testFeatureExtractor(sift, tf, imageAPath,imageBPath);
```

The repeatability results are also cached, thus the data are loaded from cache and nothing is recalculated for successive calls. To visualise the correspondences call:

```
imshow(imread(imageBPath));
benchmarks.helpers.plotFrameMatches(siftCorresps,siftReprojFrames,...
  'IsReferenceImage',false,'PlotMatchLine',false);
```

The correspondences are drawn in Figure B.3.

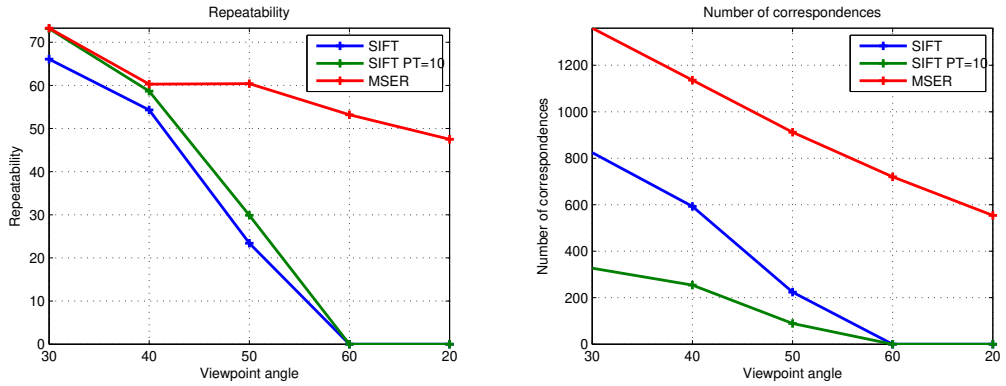


Figure B.2: Feature extractors Repeatability and number of correspondences for the graffiti dataset.

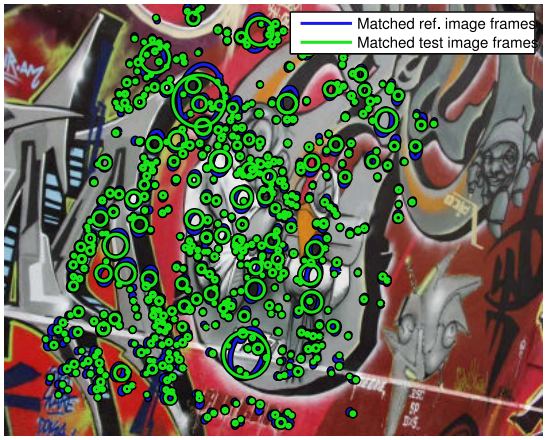


Figure B.3: Frame correspondences between first and third image from the graffiti dataset using the SIFT feature extractor.

Matching score

The computation of the matching score differs from the repeatability score. The one-to-one correspondences are not only based on the feature frames geometry (overlaps) but also the distance in the descriptor space. Therefore the feature extractor must be able to extract feature descriptors. This is not the case of MSER feature extractor, so it has to be coupled with a feature extractor which supports descriptor calculation. Unfortunately none of the built-in descriptors is affine invariant so only similarity invariant SIFTs is used.

```
mserWithSift = localFeatures.DescriptorAdapter(mser, sift);
featureExtractors = {sift, thrSift, mserWithSift};
```

The matching benchmark object can be constructed.

```
matchingBenchmark = RepeatabilityBenchmark('Mode','MatchingScore');
```

The rest is the same as for repeatability.

```
matching = zeros(numel(featureExtractorss),dataset.numImages);
numMatches = zeros(numel(featureExtractorss),dataset.numImages);

for detIdx = 1:numel(featureExtractorss)
```

```
  featExtractor = featureExtractorss{detIdx};
  for imgIdx = 2:dataset.numImages
    imageBPath = dataset.getImagePath(imgIdx);
    tf = dataset.getTransformation(imgIdx);
    [matching(detIdx,imgIdx) numMatches(detIdx,imgIdx)] = ...
      matchingBenchmark.testFeatureExtractor(featExtractor, ...
        tf, imageAPath,imageBPath);
  end
end

detNames = {'SIFT','SIFT PT=10','MSER with SIFT'};

plot(matching'.*100,'LineWidth',2); legend(detNames);
...
plot(numMatches','LineWidth',2); legend(detNames);
...
```

The matching score and number of correspondences is shown in Figure B.4.

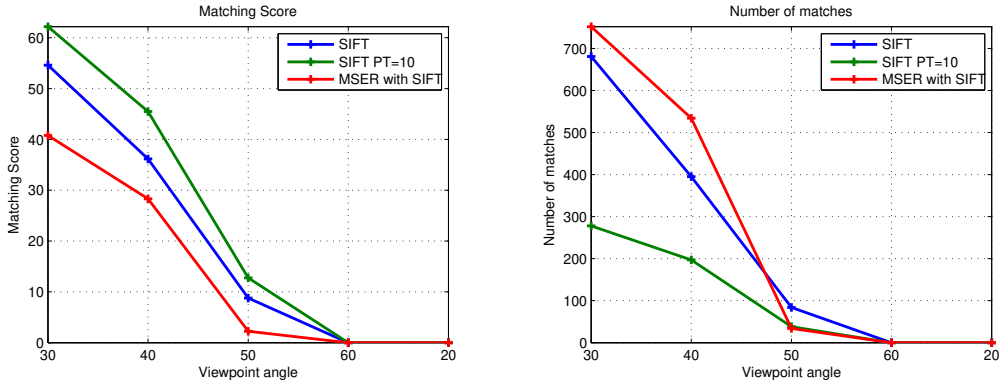


Figure B.4: Feature extractors Matching score and number of correct matches for the graffiti dataset.

Image retrieval benchmark tutorial

The image retrieval benchmark tests feature extractors in a simple image retrieval system. The retrieval benchmark closely follows the work Jégou et al. [15]. First a set of local features is detected by selected feature extractor, and described using selected descriptor. To find most similar features it employs a K-Nearest neighbours search over descriptors from the all dataset images. Finally, a simple voting criterion based on K-nearest descriptors distances is used to sort the images (for the details c.f. [15]).

The dataset used in the evaluation consists of a set of images and a set of queries. Set of ground truth images for each query is split into three classes 'Good', 'Ok', 'Junk'. For each query, the average precision (area under the precision-recall curve) is calculated and averaged over all queries to get mean Average Precision (mAP) of the feature extractor.

Feature extractors algorithms comparison

The main purpose of this benchmark is to compare the feature extraction algorithms. In this tutorial we have selected feature extractors, which are part of the VLFeat library:

```
featExtractors{1} = VlFeatCovdet('method', 'hessianlaplace', ...
                                 'estimateaffineshape', true, ...
                                 'estimateorientation', false, ...
                                 'peakthreshold',0.0035,...
                                 'doubleImage', false);
featExtractors{2} = VlFeatCovdet('method', 'harrislaplace', ...
                                 'estimateaffineshape', true, ...
                                 'estimateorientation', false, ...
                                 'peakthreshold',0.0000004,...
                                 'doubleImage', false);
featExtractors{3} = VlFeatSift('PeakThresh',4);
```

The first two image feature extractors are affine covariant whereas the third one is just similarity invariant and is closely similar to Lowe's original SIFT feature extractor (DoG detector, in fact). All local features are described using by SIFT descriptor.

To perform the image retrieval benchmark we defined a subset of the original 'The Oxford Buildings' dataset to compute the results in a reasonable time.

```
dataset = VggRetrievalDataset('Category','oxbuild',...
                              'OkImagesNum',inf,...
                              'JunkImagesNum',100,...
                              'BadImagesNum',100);
```

The subset of Oxford buildings contains only 748 images as only a part of the 'Junk' and 'Bad' images is included. 'Bad' are images which does not contain anything from the queries. The original dataset consist of 5062 images.

Now an instance of a benchmark class is created. The benchmark computes the nearest neighbours over all images. This can be too memory consuming, however the search can be split into several parts and the results merged. The parameter 'MaxNumImagesPerSearch' sets how many images are processed at one KNN search. For the call:

```
retBenchmark = RetrievalBenchmark('MaxNumImagesPerSearch',100);
```

The estimated memory consumption is approximately:

$100 \times 2000 \times 128 \times 4 \sim 100$MB of memory

Given each image contains around 2000 features on average, each feature is described by 128 bytes long descriptor and the fact that the used KNN algorithm works only with single (4 Byte) values.

Finally we can run the benchmark:

```
for d=1:numel(featExtractors)
  [mAP(d) info(d)] =...
    retBenchmark.testFeatureExtractor(featExtractors{d}, dataset);
end
```

Even having a subset of the Oxford buildings dataset, it takes a while to evaluate the benchmark for selected feature extractors. The feature extraction for a single image takes several seconds so overall the feature extraction takes approximately:

$3 \times 748 \times 3 = 6732\text{s} \sim 2\text{h}$

Giving you a plenty of time for a coffee or even a lunch. Fortunately if you have setup Matlab Parallel Computing Toolbox running this benchmark with open matlabpool can run feature extraction and KNN computation in parallel.

Both the features and partial KNN search results are stored in the cache so the computation can be interrupted and resumed at any time.

Average precisions

The results of the benchmark can be viewed at several levels of detail. The most general result is the mean Average Precision (mAP), a single value per a feature extractor. The mAPs can be visualises in a bar graph:

```
detNames = {'VLF-heslap', 'VLF-harlap', 'VLFeat-SIFT'};
bar(mAP);
```

The mean average precision is shown in Figure B.5 (a).

```
    VLF-heslap  VLF-harlap    VLF-SIFT
mAP      0.721       0.772       0.758
```

Note, that these values are computed only over small part of the Oxford Buildings dataset, and therefore are not directly comparable to the other state-of-the-art image retrieval systems run on the Oxford Buildings dataset. On the other hand this benchmark does not include any quantisation or spatial verification and thus is more focused on comparison of image features extractors.

An important measure in the features extractors comparison is a number of descriptors computed for each image, or average number of features per image. The average numbers of features can be easily obtained using:

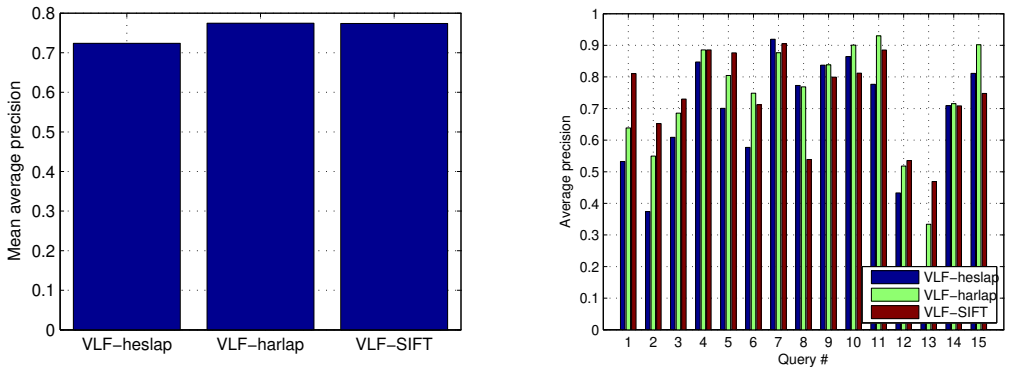
```
numDescriptors = cat(1,info(:).numDescriptors);
numQueryDescriptors = cat(1,info(:).numQueryDescriptors);
avgDescsNum(1,:) = mean(numDescriptors,2);
avgDescsNum(2,:) = mean(numQueryDescriptors,2);
```

It can be seen that the selected set of feature extractors produce similar number of features with the selected settings:

```
                  VLF-heslap  VLF-harlap    VLF-SIFT
      Avg. Descs.    1803.822    1678.195    1843.202
Avg. Query Descs.     892.582     869.255     853.582
```

To get better insight where the extractors differ, we can plot the APs per each query. These values are also contained in the info structure. For example the APs for the first 15 queries can be visualised by:

```
queriesAp = cat(1,info(:).queriesAp); % Values from struct to single array
selectedQAps = queriesAp(:,1:15);     % Pick only first 15 queries
bar(selectedQAps');
```



(a) Mean average precision of the selected feature extractors.

(b) Average precisions of the feature extractors over the first 15 queries.

Figure B.5: Average precision of the tested feature extractors.

And the results are shown in Figure B.5 (b).

As you can see there are big differences between the queries. For example query number 8 as it is a query where the SIFT feature extractor gets much worse than other algorithms. Let's investigate the query number 8 in more detail. In the first step, we can show the precision recall curves.

Precision recall curves

The precision-recall curves are not part of the results but they can be easily calculated using the `rankedListAp` static method.

```
queryNum = 8;
query = dataset.getQuery(queryNum);
for d=1:numel(featExtractors)
  % AP is calculated only based on the ranked list of retrieved images
  rankedList = rankedLists{d}(:,queryNum);
  [ap recall(:,d) precision(:,d)] = ...
    retBenchmark.rankedListAp(query, rankedList);
end
% Plot the curves
plot(recall, precision,'LineWidth',2);
```

The precision recall curves are shown in Figure B.6 (a).

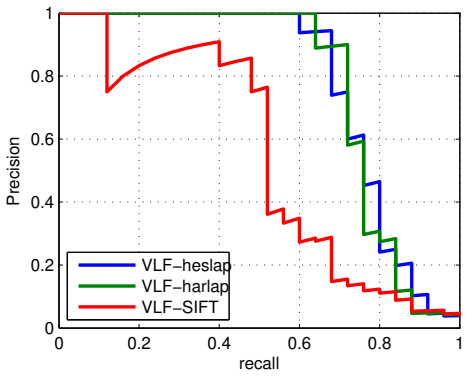
In this graph it can be seen that a VLF-SIFT (DoG + SIFT descriptor) achieved lower AP score because one of the first ranked images is wrong, therefore the area under the precision recall curve shrinks significantly.

We can check it by showing the retrieved images.

Retrieved images

The indices of the retrieved images are stored in `info.rankedList` field. It contains indices of all dataset images sorted by the voting score of each image. The details of the voting scheme can be found in the help string of the retrieval benchmark class or in [15]. To asses the performance of the feature extractor, let's inspect the query number 8.

```
image(imread(dataset.getImagePath(query.imageId)));
% Convert query rectangle [xmin ymin xmax ymax] to [x y w h]
box = [query.box(1:2);query.box(3:4) - query.box(1:2)];
rectangle('Position',box,'LineWidth',2,'EdgeColor','y');
```



(a) 8th query precision-recall curves of the tested feature extractors.

(b) Query image of the 8th query with the query bounding box.

Figure B.6: Precision recall curves and the query image of the 8th query.

The query image is shown in Figure B.6.

Having the ranked list we can show the retrieved images for all feature extractors.

```
rankedLists = {info(:).rankedList}; % Ranked list of the retrieved images
numViewedImages = 20;
for ri = 1:numViewedImages
  % The first image is the query image itself
  imgId = rankedList(ri+1);
  imgPath = dataset.getImagePath(imgId);
  subplot(5,numViewedImages/5,ri);
  subimage(imread(imgPath));
end
```

Images retrieved by the VLFeat Hessian Laplace are shown in Figure B.7.

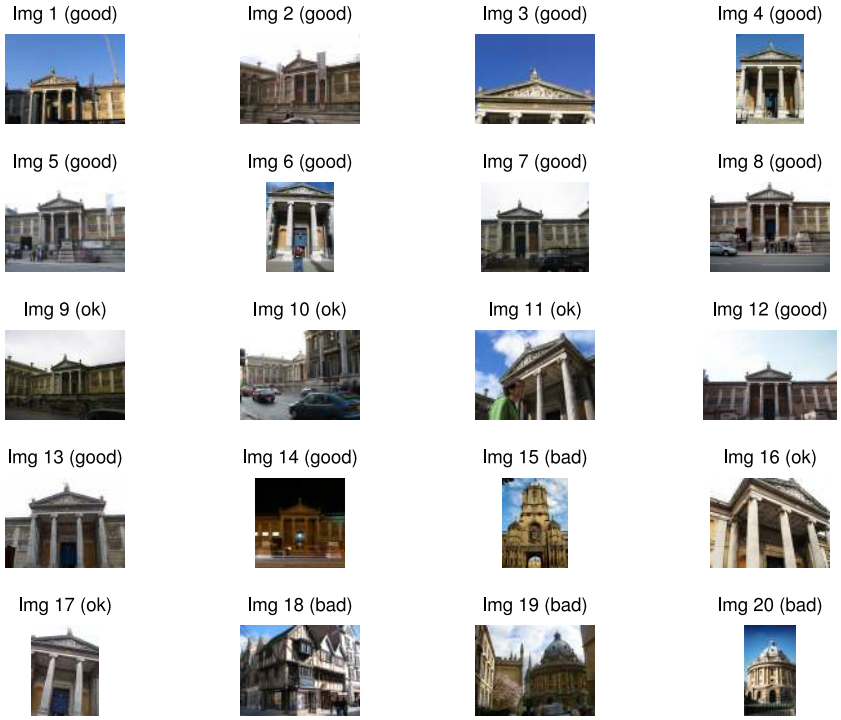


Figure B.7: First 20 retrieved images by VLFeat Hessian Laplace feature extractor.

Appendix C

All results for response function threshold experiments

In this appendix, results of experiments varying response threshold, discussed in Section 5.2, are given. In the enclosed tables it is possible to see that the definition of response function value differ significantly per local image feature detector implementation.

In Table C.1 and C.2 are results of DTU Robot 3D experiment which tested detector geometric precision. These values are plot in Figure 5.9 and 5.8.

In Table are results with Retrieval benchmark. These values are plot in Figure 5.10.

VLF DoG thr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**	
Avg. num feat.	3343	2569	2006	1551	1190	937	733	573	351	
Repeatability [%]	0.45	0.46	0.46	0.46	0.45	0.44	0.44	0.44	0.43	
MSER min. marg.	**1**	**2**	**5**	**10**	**20**					
Avg. num feat.	3081	2376	1353	597	162					
Repeatability [%]	0.42	0.46	0.45	0.42	0.37					
CMP LoG thr.	**6**	**8**	**10**	**14**	**17**	**20**	**25**	**30**		
Avg. num feat.	3249	2792	2423	1829	1495	1215	868	629		
Repeatability [%]	0.47	0.48	0.48	0.48	0.47	0.47	0.46	0.46		
CMP DoG thr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**	
Avg. num feat.	2917	2182	1668	1273	965	743	571	435	256	
Repeatability [%]	0.44	0.46	0.46	0.45	0.45	0.44	0.44	0.43	0.43	
Vgg HarLapAff thr.	**1e+02**	**1e+03**	**5e+03**	**1e+04**						
Avg. num feat.	1380	1274	828	597						
Repeatability [%]	0.37	0.37	0.35	0.34						
VLF HarLap thr.	**2.5**	**5**	**1**	**2**	**4**	**8**	**1.6**	**3.2**		
Avg. num feat.	2894	2335	1795	1292	866	541	311	144		
Repeatability [%]	0.39	0.38	0.37	0.37	0.35	0.37	0.36	0.33		
VLF HarLapAff thr.	**2.5**	**5**	**1**	**2**	**4**	**8**	**1.6**	**3.2**		
Avg. num feat.	2894	2335	1795	1292	866	541	311	144		
Repeatability [%]	0.36	0.36	0.35	0.34	0.33	0.34	0.35	0.32		
VGG HesLapAff thr.	**1e+02**	**2e+02**	**5e+02**	**1e+03**	**2e+03**	**3e+03**				
Avg. num feat.	4898	3550	1904	977	405	200				
Repeatability [%]	0.44	0.42	0.39	0.37	0.33	0.3				
VLF Hessian thr.	**0.0015**	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	3327	2385	1795	1408	1133	931	662	394	201	
Repeatability [%]	0.5	0.5	0.51	0.51	0.51	0.5	0.51	0.5	0.5	
VLF HessLap thr.	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	**0.02**	
Avg. num feat.	2982	2236	1740	1383	1134	801	447	211	NaN	
Repeatability [%]	0.44	0.43	0.43	0.42	0.42	0.41	0.41	0.39	0.41	
VLF HessAff thr.	**0.0015**	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	3327	2385	1795	1408	1133	931	662	394	201	
Repeatability [%]	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.45	0.46	
CMP Hessian thr.	**6**	**7**	**8**	**9**	**10**	**12**	**14**	**16**	**18**	**20**
Avg. num feat.	4331	3597	2993	2500	2098	1486	1061	757	552	412
Repeatability [%]	0.5	0.49	0.49	0.49	0.49	0.48	0.49	0.47	0.47	0.46
CMP HessAff thr.	**6**	**7**	**8**	**9**	**10**	**12**	**14**	**16**	**18**	**20**
Avg. num feat.	3042	2595	2222	1907	1637	1210	897	664	495	377
Repeatability [%]	0.48	0.48	0.47	0.47	0.47	0.47	0.47	0.45	0.45	0.44

Table C.1: Detector repeatability in DTU Robot 3D benchmark as a function of average number of detected features per image when varying response function threshold (or min. margin for MSER). Repeatability was computed over first 10 scenes in *Linear path* dataset for the viewpoint 0.5m distant from the reference camera position.

VLF DoG thr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**	
Avg. num feat.	3629	2791	2180	1689	1318	1036	791	600	366	
Repeatability [%]	0.38	0.38	0.37	0.35	0.34	0.33	0.33	0.32	0.32	
MSER min. margin	**1**	**2**	**5**	**10**	**20**					
Avg. num feat.	3731	2862	1591	680	175					
Repeatability [%]	0.36	0.37	0.35	0.35	0.33					
CMP LoG thr.	**6**	**8**	**10**	**14**	**17**	**20**	**25**	**30**		
Avg. num feat.	3595	3092	2664	2005	1631	1330	947	665		
Repeatability [%]	0.41	0.4	0.4	0.39	0.38	0.37	0.37	0.36		
CMP DoG thr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**10**	
Avg. num feat.	3161	2377	1806	1372	1049	799	595	448	265	
Repeatability [%]	0.39	0.38	0.37	0.36	0.36	0.35	0.34	0.34	0.35	
Vgg HarLapAff thr.	**1e+02**	**1e+03**	**5e+03**	**1e+04**						
Avg. num feat.	1454	1321	857	623						
Repeatability [%]	0.32	0.32	0.31	0.3						
VLF HarLap thr.	**2.5**	**5**	**1**	**2**	**4**	**8**	**1.6**	**3.2**		
Avg. num feat.	3159	2464	1834	1306	870	541	309	142		
Repeatability [%]	0.4	0.39	0.39	0.37	0.36	0.34	0.33	0.33		
VLF HarLapAff thr.	**2.5**	**5**	**1**	**2**	**4**	**8**	**1.6**	**3.2**		
Avg. num feat.	3159	2464	1834	1306	870	541	309	142		
Repeatability [%]	0.36	0.36	0.35	0.34	0.32	0.32	0.31	0.33		
VGG HesLapAff thr.	**1e+02**	**2e+02**	**5e+02**	**1e+03**	**2e+03**	**3e+03**				
Avg. num feat.	5217	3728	1951	1002	407	199				
Repeatability [%]	0.38	0.37	0.34	0.31	0.27	0.22				
VLF Hessian thr.	**0.0015**	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	3439	2453	1834	1435	1143	934	662	392	199	
Repeatability [%]	0.49	0.47	0.46	0.45	0.45	0.44	0.45	0.44	0.44	
VLF HessLap thr.	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	**0.02**	
Avg. num feat.	3017	2256	1751	1386	1134	798	444	210	NaN	
Repeatability [%]	0.46	0.45	0.43	0.43	0.42	0.42	0.42	0.4	0.36	
VLF HessAff thr.	**0.0015**	**0.002**	**0.0025**	**0.003**	**0.0035**	**0.004**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	3439	2453	1834	1435	1143	934	662	392	199	
Repeatability [%]	0.44	0.42	0.41	0.4	0.4	0.39	0.4	0.38	0.39	
CMP Hessian thr.	**6**	**7**	**8**	**9**	**10**	**12**	**14**	**16**	**18**	**20**
Avg. num feat.	4688	3903	3261	2734	2268	1558	1098	771	559	415
Repeatability [%]	0.43	0.42	0.42	0.41	0.41	0.4	0.39	0.39	0.39	0.39
CMP HessAff thr.	**6**	**7**	**8**	**9**	**10**	**12**	**14**	**16**	**18**	**20**
Avg. num feat.	3334	2841	2435	2089	1776	1264	924	674	502	382
Repeatability [%]	0.41	0.4	0.39	0.39	0.39	0.38	0.38	0.38	0.37	0.37

Table C.2: Detector repeatability in DTU Robot 3D benchmark as a function of average number of detected features per image when varying response function threshold (or min. margin for MSER). Repeatability was computed over first 10 scenes in *Arc 2* dataset for the viewpoint 25°.

VLF DoG thr.	**0**	**2**	**4**	**6**	**8**	**10**
Avg. num feat.	2637	1738	1196	800	522	333
mAP	0.73	0.75	0.73	0.65	0.6	0.51
MSER min. margin	**1**	**2**	**5**	**10**	**20**	
Avg. num feat.	2448	2045	1284	693	279	
mAP	0.83	0.83	0.82	0.78	0.69	
CMP DoG thr.	**0**	**2**	**4**	**6**	**8**	**10**
Avg. num feat.	2336	1452	968	629	399	246
mAP	0.74	0.76	0.73	0.67	0.59	0.51
CMP LoG thr.	**7**	**9**	**12**	**15**	**20**	
Avg. num feat.	2095	1903	1645	1420	1104	
mAP	0.78	0.78	0.77	0.76	0.74	
Vgg HarLapAff thr.	**1e+02**	**5e+03**	**1e+04**	**5e+04**		
Avg. num feat.	1416	864	617	177		
mAP	0.77	0.69	0.61	0.36		
VLF HarLap thr.	**5**	**2**	**4**	**1.6**	**3.2**	
Avg. num feat.	3470	2329	1750	771	425	
mAP	0.67	0.61	0.58	0.47	0.37	
VLF HarLapAff thr.	**5**	**2**	**4**	**1.6**	**3.2**	
Avg. num feat.	3416	2293	1724	761	420	
mAP	0.68	0.62	0.58	0.46	0.38	
VGG HesLapAff thr.	**5e+02**	**1e+03**	**2e+03**	**3e+03**		
Avg. num feat.	1427	741	283	129		
mAP	0.76	0.67	0.52	0.39		
VLF Hessian thr.	**0.0025**	**0.0035**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	4000	2933	1997	1301	748	
mAP	0.72	0.69	0.65	0.6	0.54	
VLF HessLap thr.	**0.0035**	**0.005**	**0.007**	**0.01**	**0.02**	
Avg. num feat.	6098	3947	2432	1311	256	
mAP	0.62	0.58	0.55	0.49	0.29	
VLF HessAff thr.	**0.0025**	**0.0035**	**0.005**	**0.007**	**0.01**	
Avg. num feat.	3889	2853	1944	1268	731	
mAP	0.71	0.68	0.63	0.59	0.54	
VLF HessLapAff thr.	**0.0035**	**0.005**	**0.007**	**0.01**	**0.02**	
Avg. num feat.	6013	3894	2402	1296	254	
mAP	0.64	0.6	0.57	0.51	0.3	
CMP Hessian thr.	**7**	**9**	**12**	**15**	**20**	
Avg. num feat.	2962	2212	1436	935	452	
mAP	0.78	0.76	0.71	0.64	0.55	
CMP HessAff thr.	**7**	**9**	**12**	**15**	**20**	
Avg. num feat.	2238	1754	1207	820	416	
mAP	0.74	0.72	0.67	0.61	0.52	

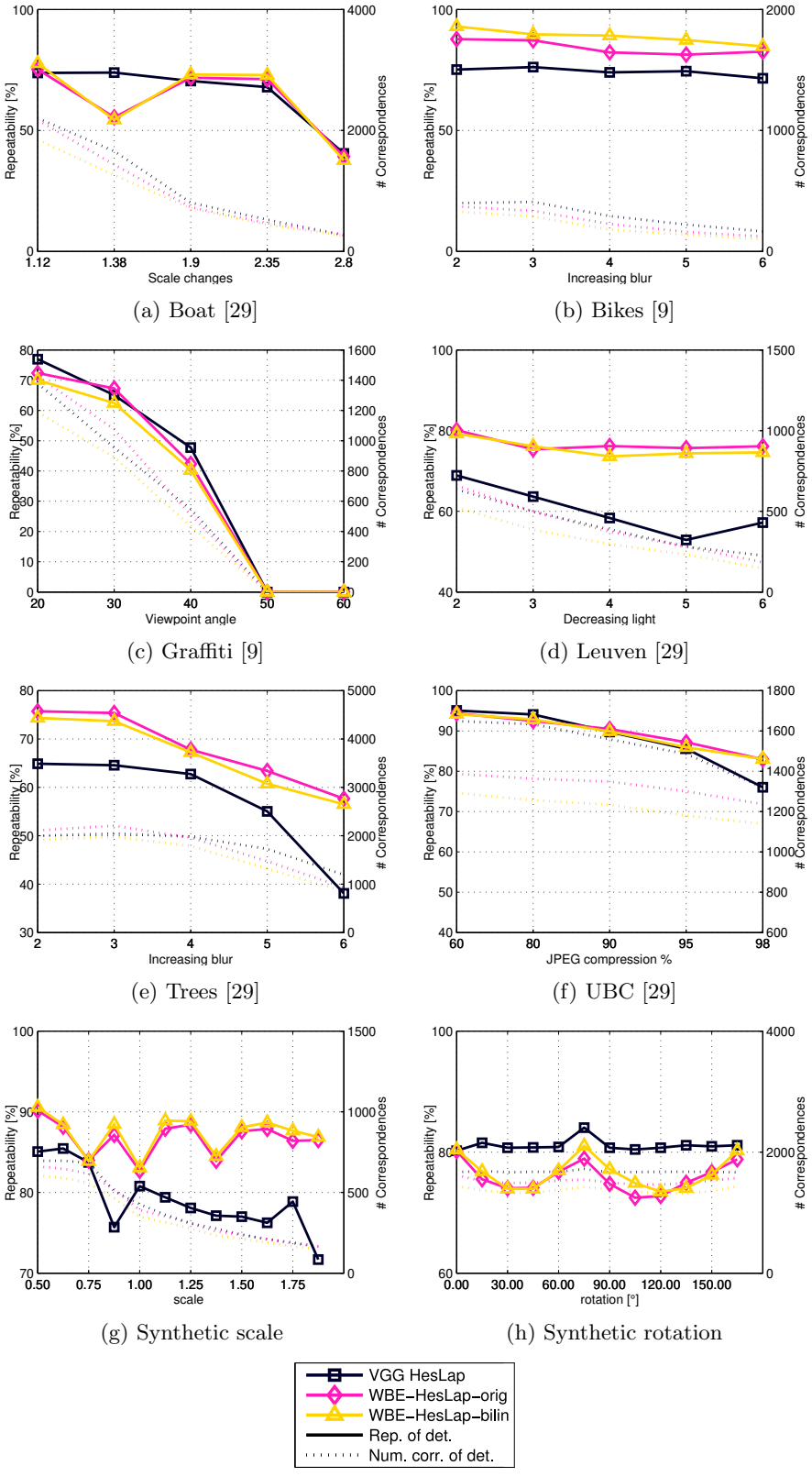
Table C.3: Detector mAP in Retrieval benchmark as a function average number of detected features per image when varying response function threshold (or min. margin for MSER). All descriptors has been computed using CMP SIFT implementation with measurement scale $\nu = 3$ and without orientation assignment. Results are computed on data from Oxford buildings dataset with a subset of 100 "Bad" images and $k = 50$.

Appendix D

Evaluation of emulated detectors with homography based benchmarks

In this appendix, we give complete results of the emulated detectors, presented in Chapter 6, evaluated with most of the datasets introduced in Mikolajczyk et al. [29]. Some of the Mikolajczyk's datasets has been replaced with their variants from [9], which offer more precise ground truth homographies. The repeatability score (see 3.1.2) and matching score (see 3.1.3) has been evaluated. Matching score has been computed matching only descriptors (see Equation 3.8) same as it is done in Šochman and Matas [38].

Results for Hessian-Laplace based emulators are given in Figure D.2 and Figure D.2, results for DoG and Hessian based emulators in Figure D.3 and Figure D.4.



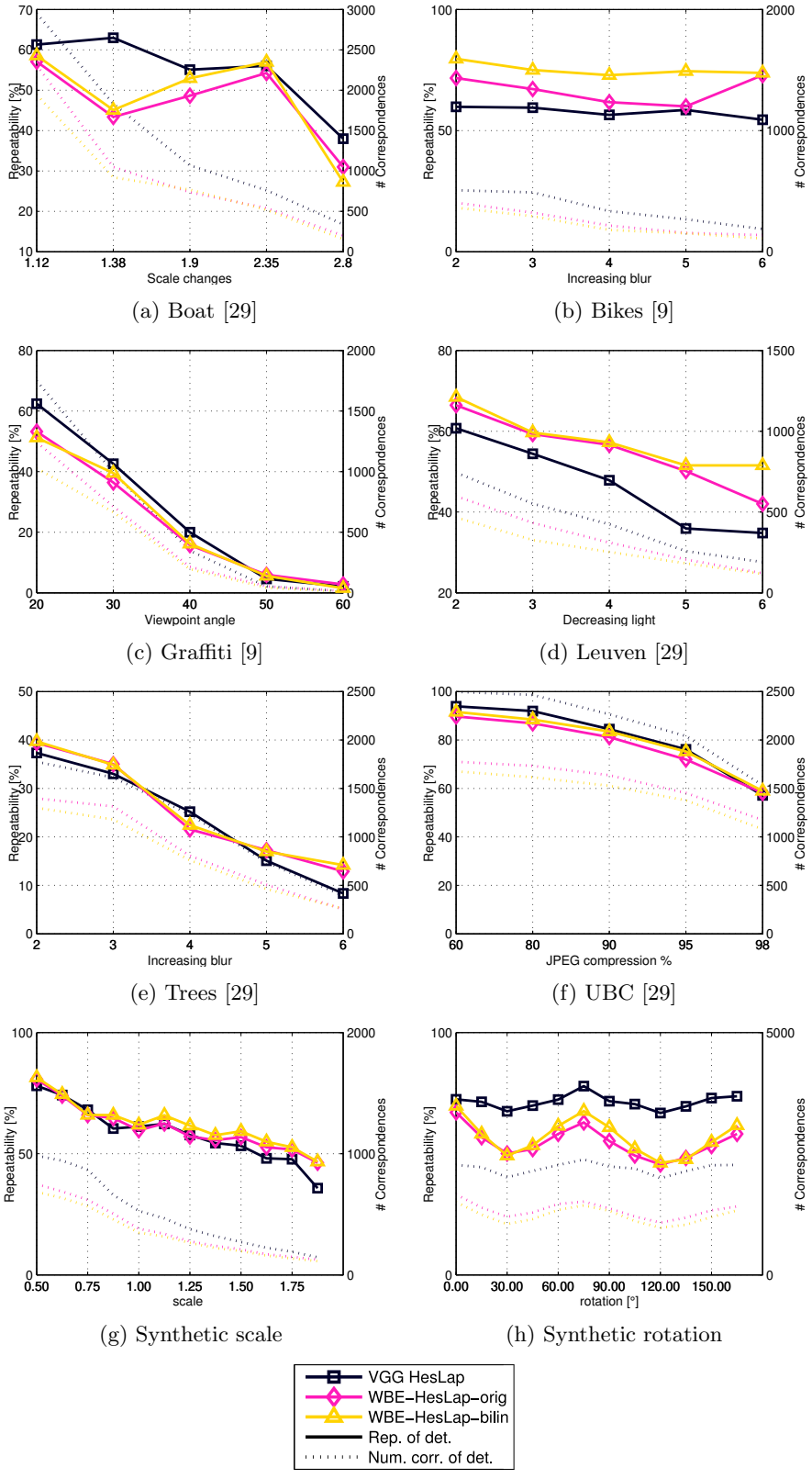
(a) Boat [29] (b) Bikes [9]

(c) Graffiti [9] (d) Leuven [29]

(e) Trees [29] (f) UBC [29]

(g) Synthetic scale (h) Synthetic rotation

Figure D.1: Repeatability and number of correspondences (see Section 3.1.2) of the original Hessian-Laplace emulator and Hessian-Laplace emulator with improved patch extraction using bilinear interpolation. Compared to the performance of the teacher VGG Hessian-Laplace. Results with the synthetic datasets (see Section 4.2.3) computed as an average over 7 images.



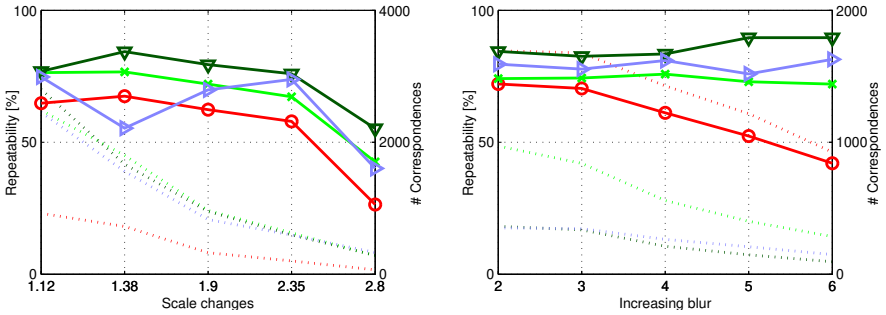
(a) Boat [29] (b) Bikes [9]

(c) Graffiti [9] (d) Leuven [29]

(e) Trees [29] (f) UBC [29]

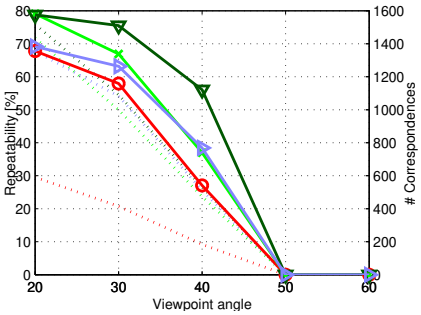
(g) Synthetic scale (h) Synthetic rotation

Figure D.2: Matching score and number of correct matches (see Section 3.1.3) of the original Hessian-Laplace emulator and Hessian-Laplace emulator with improved patch extraction using bilinear interpolation. Compared to the performance of the teacher VGG Hessian-Laplace. Results with the synthetic datasets (see Section 4.2.3) computed as an average over 7 images.

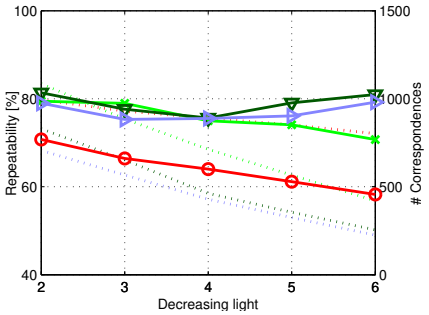


(a) Boat [29]

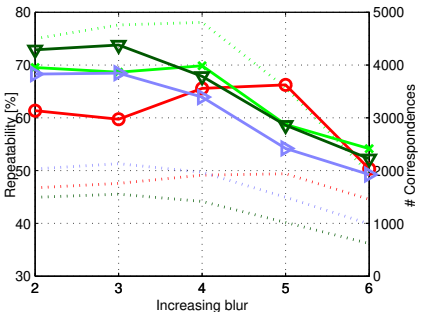
(b) Bikes [9]



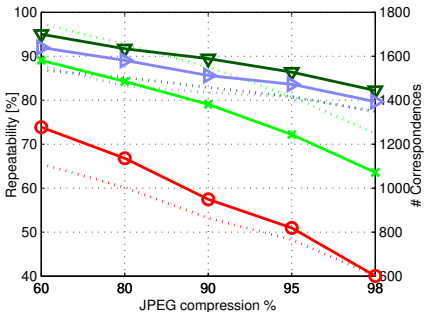
(c) Graffiti [9]



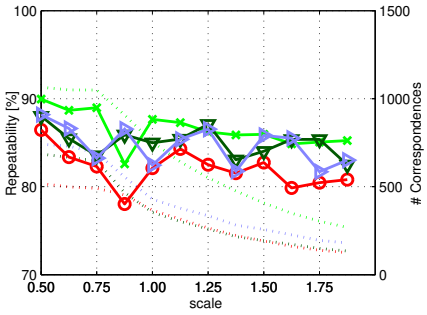
(d) Leuven [29]



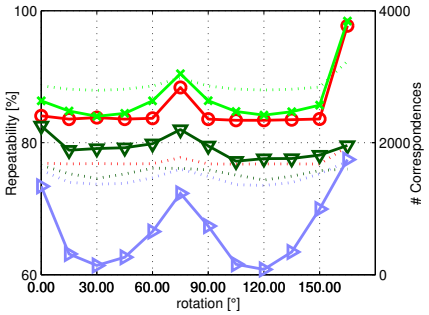
(e) Trees [29]



(f) UBC [29]



(g) Synthetic scale



(h) Synthetic rotation

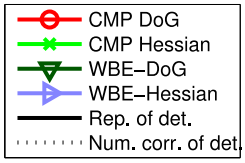
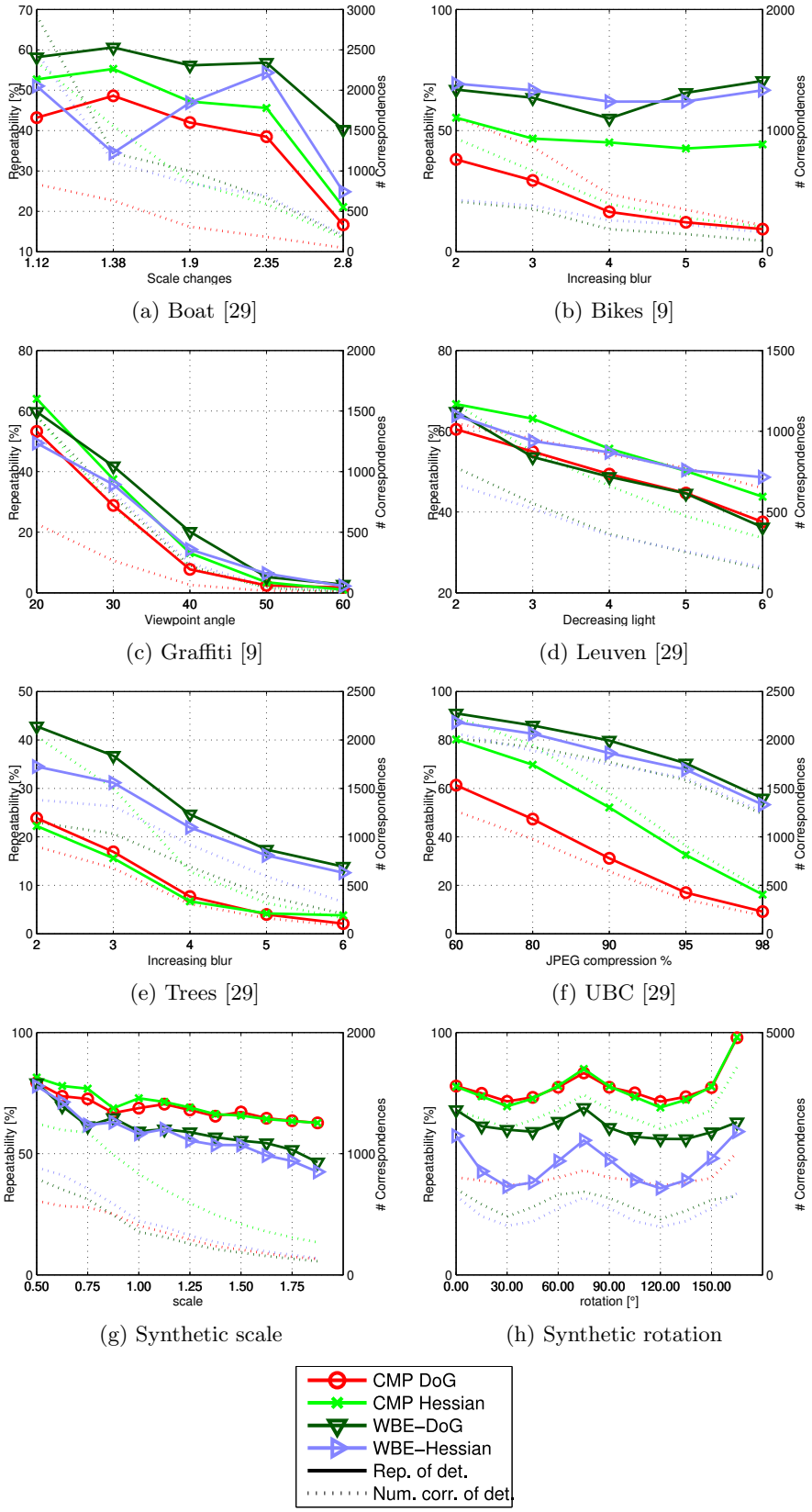


Figure D.3: Repeatability and number of correspondences (see Section 3.1.2) of the new DoG and Hessian detector emulators. Compared to the performance of their teachers. Results with the synthetic datasets (see Section 4.2.3) computed as an average over 7 images.



(a) Boat [29] (b) Bikes [9]

(c) Graffiti [9] (d) Leuven [29]

(e) Trees [29] (f) UBC [29]

(g) Synthetic scale (h) Synthetic rotation

Figure D.4: Matching score and number of correct matches (see Section 3.1.3) of the new DoG and Hessian detector emulators. Compared to the performance of their teachers. Results with the synthetic datasets (see Section 4.2.3) computed as an average over 7 images.