Image Correspondences for Camera Registration

Ánoq of the Sun, Hardcore Processing *

September 14, 2009

Quoted as e.g.: Ánoq of the Sun, Ánoq, o., Ánoq, o. t. S. or Ánoq, of the Sun. Not: Sun, Á.
Ánoq is considered the "family name", always written and pronounced first.

Contents

1 Reader's Guide

This section gives is a guide to the reader, particularly for those who might prefer to skip reading parts of the document. The document contains the following main sections:

- *Introduction*: An introduction to the setting for this project and to some of the terminology used in this report
- *Previous Work*: A review of some previous work in this area. This part can **safely be skipped**, for those familiar with the reviewed articles, but may be intensive reading for new-comers to the field. There are no important conclusions in this section
- *Conclusions of Previous Work*: This section is important, for those wishing to understand the reasons for the design. It is based on specific references and forms a basis for the analysis
- *Design Analysis*: The design analysis and design choices, mostly based on the experiences exposed by previous work
- *The Implemented Methods*: Describes the methods, which were chosen for implementation and which have been implemented. This section is generally not programming language specific
- *Suggested Implementation Improvements*: Describes possible improvements to the implemented methods
- *Evaluation of the Implemented Methods*: Documents the evaluation of the implemented methods
- *Future Work and Conclusions*: Don't miss it
- *Appendices*: The appendices contain some implementation details, which are either important or tricky, when implementing it, but not important for the overall understanding. This also includes design details specific to the implementation language, Standard ML

The reader is assumed to be familiar with computer algorithms and maths at a reasonable level, particularly including linear algebra using vectors and matrices and things like computing partial derivatives. It is an advantage to be familiar with image processing, computer graphics or computer vision techniques, but most descriptions should be detailed enough that, this is not a strict requirement.

2 Introduction

This section describes the setting and terminology for this report. It also gives a general overview of the relevant areas of technical methods and finally defines the focus of the report.

2.1 Overview of Wide-baseline Image Correspondences for Camera Registration

Finding reliable correspondences in two images of a scene is a difficult and critical step towards fully automatic reconstruction of 3D scenes. This is especially true when we allow the images to be taken from arbitrary viewpoints, viewed with possibly different cameras or camera settings and in different illumination conditions. There are different ways of establishing correspondences. Two main branches

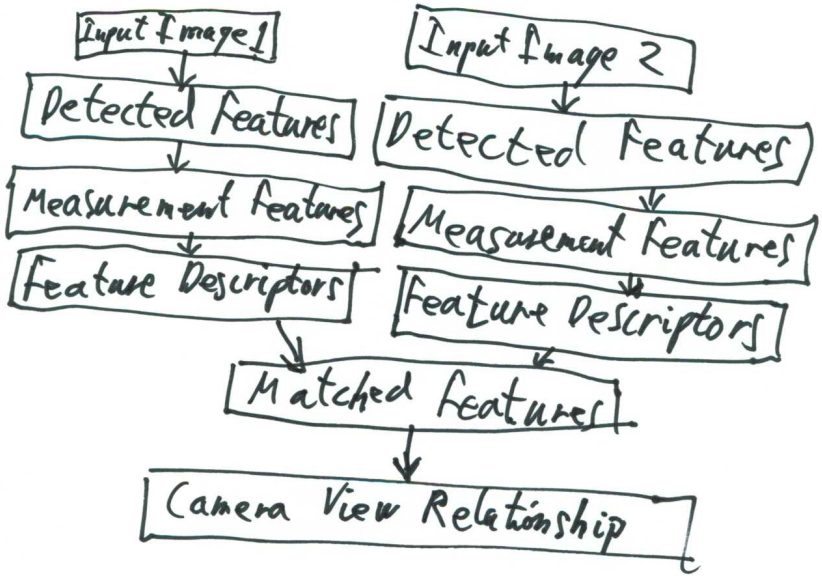

Figure 1: Data flow overview of the main steps for establishing a camera view relationship between two images by image correspondences. The steps for each image individually are feature detection, determining measurement areas and making feature descriptors. The steps involving both images are feature matching and estimation of the camera view relationship. These steps are symbolized by the arrows, while the result of each step are the boxes, starting from the input images.

of methods are the *direct* pixel based methods and the *feature-based* methods. This report considers only feature based methods, which, according to the litterature, are quite robust and precise.

When the camera viewpoint positions differ significantly between the input images, the reconstruction problem is referred to as having *wide-baseline*. This is as opposed to having a *narrow-baseline*, where the viewpoint positions are fairly close to each other. The baseline is the line between the projection centres of the camera viewpoints. In the narrow-baseline case, there are many properties of the input images, which can often be exploited. For example, local image deformations can often be realistically approximated by only translation or translation with rotation and pixel intensities can be assumed not to change very much between images. These assumptions open op many possibilities for doing image matching. This report aims to handle the wide-baseline case, where more generally robust models have to be considered. For example, geometric image deformations may have to be approximated by affine transformations, in order to get successful image matching.

The *feature-based* methods can be broken down into some main steps: *Detecting* features, computing a *descriptor* for each detected feature, *matching* the features by using their descriptors and estimating the relationship of the *camera views* between the input images from the matching feature pairs. In between the first two steps, it is useful to consider an intermediate step of determining a *measurement area* from the detected feature, where the descriptor is computed on the measurement area. This step is often closely related to the feature detector or the descriptor and can sometimes be hard to isolate. The sequence of these steps is illustrated as a data flow diagram in figure 1.

2.1.1 Feature Detection

Reliable extraction of a manageable number of potentially corresponding image elements is a necessary, but certainly not sufficient prerequisite, for successful wide-baseline matching. We refer to such extracted image elements as *detected features*. The features here could be points, lines, edges, pixel regions or conics, such as ellipses. Other kinds of features may even be possible as well. Lines or edges will not be considered in this report. A few good reasons are that, no fixed location exists along a line and, according to [Hart03], lines often arise from object occlusions in the depicted scene, in which case they are not reliable features. We shall mostly deal with acquiring points and the term *interest point* is sometimes used for this. Also, we will often be associating a conic, such as a circle or an ellipse, with the detected feature, so the methods considered also often result in a *detected conic*. We will also see the term *detected regions*, since some of the detection methods find pixel regions, where the desired point is obtained as the center of the detected region. The term *region* is generally used when detecting a certain area of pixels and such methods are known as *region based* detection methods. In general, it is usually a central feature point, which is sought, and it is often also somehow located located at a particular scale of the image. For certain kinds of features, the term *blob* can be used.

2.1.2 Measurement Areas

For each detected feature, we can determine a measurement area from it. This could be a circle, an ellipse, a box or something else, which defines the area, which we will use for computing a descriptor for. For affine methods, areas of a fixed Euclidean shape, like rectangles or circles, are not good enough, since their shape is not preserved under affine transformation. For scale and rotation invariant methods however, they are commonly used.

For some methods, the step of determining a measurement area can be somewhat artifical, since it may already be part of either the feature detection method or the feature descriptor method, possibly even both. However, they in some sense define the interface between the method for feature detection and the method for constructing a feature descriptor.

2.1.3 Feature Descriptors

Often, a large number, perhaps hundreds or thousands, of possibly overlapping features are obtained. A feature *descriptor*, which is typically a vector, is then associated with each measurement area. This descriptor is chosen so as to discriminate between the features. A descriptor is often an approximate description of the image contents of the measurement area. One reason for introducing it, is to make matching of features between images faster than comparing image contents directly. Another reason for its use is to allow for some inaccuracies in the measurement areas, such as location, shape and noise.

2.1.4 Feature Matching

Correspondences may be established between two images of the same scene, by detecting and representing features independently in both images and then matching the features based on their descriptors. By design, changes between features in the two images try to follow viewpoint changes, so by design, corresponding features in the two images will have similar (ideally identical) feature descriptors. Correspondences can then be more easily established. The fact that multiple features are matched helps in making the method is robust to partial occlusions.

The *matching* between feature descriptors can be done in various ways. The descriptor vectors can be brute-force pair-wise matched, using e.g. the Euclidean distance. Other distance measures or more

efficient, possibly approximating, ways of finding matches are also possible.

2.1.5 Camera View Relationship

The final step that we shall go through here is deriving the final relationship between the camera views. This is usually done by finding a *fundamental matrix*, a *homography* or even just an affine mapping, which describes a relationship between the images. A homography is a $3x3$ matrix, which maps points on a plane in one image into points on a plane in the other image. Thus, it is a relationship between both the camera views and a plane in the scene. A fundamental matrix is also a $3x3$ matrix, but it maps a point in one image into a *line* in another image, where the true corresponding point in the other image is somewhere on that line. Thus, the fundamental matrix only describes the relationship between the camera views, independently of the scene. In this project, we shall only use homographies between images, as a means to verify correctness of detected features. For this it is important to notice that, they only map points, which are *on a plane in the scene*.

The camera relationship is typically sought as being the relationship consistent with the largest number of matched correspondences. The *RANdom SAmple Consensus (RANSAC)* algorithm is commonly used here.

When considering more than two images, it is natural to start thinking of the 3D scene represented by the images, rather than the images themselves. In this setting, relating the camera view of an image to the rest of a 3D scene is known as *camera registration*. As part of determining the camera view relationship more precisely, methods like *bundle adjustment* are often used.

2.1.6 Iterations of Methods and Applications

Many of the above methods can be applied in more than one iteration and some may be guided or improved by incorporating other methods, in order to incease robustness.

The kind of methods introduced here, and the methods that we will focus on, are methods for finding correspondences between two images, or at least a small predetermined set of images. However, many of the same methods are also often used for other applications, such as finding matches within a larger database of pictures, object recognition or image stitching.

2.2 Detection Accuracy and Invariance

The requirement for the detected features is that they should correspond to the same 3D element, which gave rise to the detected feature, for different viewpoints. The shape of features from different viewpoints is thus not fixed. The detected features should therefore adapt in shape, based on the underlying image data, so that they are the projection of the same 3D surface patch. In particular, consider images from two viewpoints and the geometric transformation between the images induced by the viewpoint change. This image-to-image transformation can be applied to the detected features between the images. Features detected after the viewpoint change should ideally be the same, modulo noise, as those detected before the viewpoint change. Thus it should, ideally, be possible to transform the detected features in one image into the detected features of the other image. Doing such an image-to-image transformation of detected features in one image and comparing to the features detected in the other image, can be used as a measure for how *accurate* the feature detection is. One way of measuring this is by an *overlap test*, which we will see later, but not use for measuring the accuracy. The *position accuracy* or *location accuracy* of detected features is also very important, for the estimate of the camera viewpoint. This mostly relates to the position of the central feature point.

Due to these properties, detected features could be considered *covariant* between the images, since they change covariantly with the viewpoint transformation. They are often referred to as *invariant* in the litterature, referring to the fact that, the features are invariant to the viewpoint of the camera, which photographed the image, in which the features are detected.

There are varying degrees of invariance to viewpoint changes. Some methods try to be invariant to only translation, some also to scale, some also to rotation. When a method is invariant to translation, scale and rotation, it is said to be invariant up to a *similarity* change. Even more refined methods can be *affinely invariant*. These kinds of invariance are collectively called *geometric invariance*. In practice, the invariance is only in the image plane, not according to the actual viewpoints of the cameras, so the invariance here is far from ideal.

The image geometry is, however, not the only property of a feature detector, which may be invariant. A detector may also be invariant to illumination changes. This is usually considered to be a one-dimensional property, specifically changes in image intensity. The invariance may be only to the intensity level, which is only translation invariance of intensity. It may also be invariance to both the offset and scale of itensities, i.e. affine intensity invariance. The scale invariance of intensity is one kind of *contrast invariance*.

In practice, the estimated *fundamental matrix* or *homography* between images is the closest we can get to making an image-to-image transformation of detected features. We will use *homographies* for the practical evaluation of the detector accuracy, since we have these available in the selected test image sets. As already stated, this requires that the features, or at least most of them, lie on a plane in the scene, which could even be the image plane itself.

2.3 Scale-Space for Feature Detectors

Several feature detection methods deal with *scale-space* in one way or another. This relates to the notion that, a detected feature has a certain scale in an image. The scale-space can be considered as an extra dimension of the image, which in some way defines the image scaled to a certain resolution. Also, scale-space handles differences in input image resolutions and makes it possible to get the advantages of having a higher resolution in one input image than other input images.

Many methods implement this as a pyramid of successively scaled versions of an input image, which is known as an *image pyramid*. Apart from performing actual scaling of the image, the image data may also be filtered, often with some kind of a Guassian filter, where adjusting the filter size can give a more fine-grained set of scales, without having to explicitly represent all image resolutions in memory. Often, scale is divided into *octaves*, and *scales within each octave*. In the case of using an image pyramid, octaves could be the images being explicitly resized, whereas the scales within each octaves could be handled by filter-width ajustment, when sampling image locations. The octaves can be used for making an initial rough search for features at roughly estimated scales. The scales in between can be used for further determining at which scale a feature is most naturally represented. Determining that a feature has a certain scale is known as *scale selection*. [Cyga09] gives a good overview of scale-space, including source code, where e.g. section 6.7.5 explains scale subdivision.

Some methods, however, handle scale-space in different ways. This can lead to less memory usage. It may also lead to either improved performance or worse performance, depending on the nature of the method. Trying to find image correspondences, without somehow handling scale-space, will probably not work for realistic wide-baseline cases.

The way that some feature detectors, and actually also some feature descriptors, work, is by doing some kind of image sampling in a local area of the image. These samples often perform some kind of differentiation of the image data. This could be done in a simple way, e.g. by subtracting neighbouring

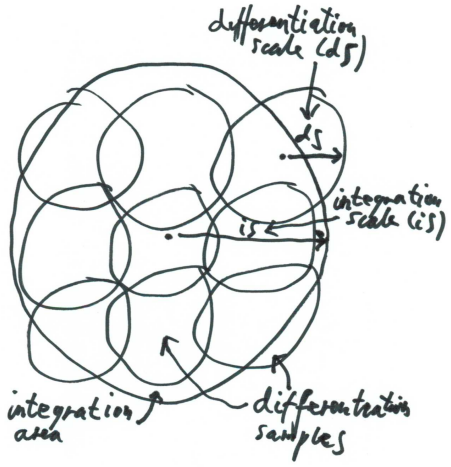

Figure 2: Differentiation scale and integration scale are used by some methods. Differentiation scale is illustrated by the radii of the small circles, integration scale by the radius of the large circle

image intensities of an image, where this image may be an input image scaled down to a certain resolution, e.g. by using one of the images of an image pyramid. It could also be done by more refined methods, such as Laplacian filtering, which is differentiation of image intensities with the *second order partial derivatives* in the x and y directions of the image. Often, several methods are combined, where combining Gaussian and Laplacian filteres would be one example. No matter which method is used, the scale at which the differentiation is done, can be referred to as the *differentiation scale*. Thus, this generally has to do with the combined filter size and scaled image resolution, at which the differentiation is performed.

Several differentiation samples over a local area can be combined, when trying to detect a feature or make a feature descriptor. The size or scale of this local area can be referred to as the *integration scale*. This is the scale at which the area of differentiation samples are integrated into one feature sample or feature vector. The differentiation scale and integration scale are illustrated in figure 2.

Often, methods have the differentation scale and integration scale tightly coupled. When the *scale of a feature* is determined, it is often the integration scale, which is used.

2.4 The Gaussian Filter and Discrete Image Convolution

For completeness of this presentation, we shall breifly present the Gaussian filter and discrete convolution.

The Gaussian function is the function of a statistical normal distribution, where σ is its standard deviation. The one-dimensional Gaussian function, usually considered as being parameterized by t for some given σ, is defined by $G(t,\sigma) = e^{-\frac{t^2}{2\sigma^2}}$. The two-dimensional Gaussian function is given by $G((x,y),\sigma) = e^{-\frac{x^2+y^2}{2\sigma^2}}$. These formulas can be normalized, such that their integrated area over t, respectively volume over (x,y), is one. Normalization of the one-dimensional Gaussian is done by scaling by $\frac{1}{\sqrt{2\pi}\ \sigma}$. Normalization of the two-dimensional Gaussian is done by scaling by $\frac{1}{2\pi\sigma^2}$.

Convolution is normally done discretely, i.e. at invdividual points pixel by pixel, in a window of limited filter size, e.g. a window of the Gaussian filter. In case of discrete convolution, as presented here, it is the process of, for each pixel in the image P, placing the window of the filter, such that it is centered over the pixel P of the image. Then each pixel of the filter window is multiplied by the pixel of the image at the position where that filter pixel is placed. These pixel products are summed

and yield the resulting convolution value for pixel P. This convolution can be done for all pixels in an image and stored in an output image of the same size as the input image, or it can be done on the fly for each needed convoluted pixel. Strictly speaking, the image of the filter should be mirrored, due to the way that convolution is defined, but since most filters are symmetric, this is normally not needed. The Haar-wavelet filter, which we will consider later, is not symmetric, but for what we use it for, the mirroring is not important, as long as we are consistent about it. We use the operator $\otimes$ to denote convolution. Convolution can also be done at sub-pixel precision, but this will not be presented here.

2.5 Our Focus

To limit the extent of this report, we will focus on the methods for *feature detection*, *feature descriptors* and determining the *measurement areas* in between. Matching features and how to incorporate this into a full pipeline will be considered to a limited extent.

Establishing camera view relationships will not be described, except for its use in evaluating the implemented methods. A follow-up report on this part is planned to be made. The experiments are mainly for evaluating feature detectors and feature descriptors. In the implementation and most of the descriptions, we will only consider two input images.

3 Previous Work

This section describes some relevant previous work, divided into subsections of feature *detectors*, feature *descriptors*, *matching* strategies, *comparisons* of methods and general tutorials.

3.1 Feature Detectors

3.1.1 The Harris Corner and Edge Detector

The famous article [Harr88] introduces the Harris point and edge detector, which is an image filter. We will denote two of its basic quantities as I_x and I_y, which denotes the gradients of a pixel P in the x and y directions. I_y is calculated by taking the intensity of the pixel below P and subtracting the intensity of the pixel above P. I_x is similar, the pixel to the right of P minus the one to the left of P. This approximates the first order partial derivatives $\partial I/\partial x$ and $\partial I/\partial y$ in the x and y directions, respectively. From these quantities, the Harris matrix is constructed as:

$$M_H = \begin{bmatrix} I_x^2 \otimes G(\sigma) & I_x I_y \otimes G(\sigma) \\ I_x I_y \otimes G(\sigma) & I_y^2 \otimes G(\sigma) \end{bmatrix} \tag{1}$$

Here, the notation $\otimes G(\sigma)$ denotes convolution with a Gaussian filter with standard deviation σ. What is used from this matrix is particularly the trace

$$Tr(M_H) = I_x^2 \otimes G(\sigma) + I_y^2 \otimes G(\sigma) \tag{2}$$

and the determinant

$$Det(M_H) = (I_x^2 \otimes G(\sigma))\ (I_y^2 \otimes G(\sigma)) - (I_x I_y \otimes G(\sigma))^2 \tag{3}$$

The corner response function is given as

$$Det(M_H) - kTr(M_H)^2 \tag{4}$$

where k should be 0.04, according to [Poll00]. This function takes on positive values at corners, negative values at edges and is close to zero elsewhere. For most methods that we shall consider, we are only interested in corners, not edges, i.e. where this function is positive.

3.1.2 Scale Adapted Harris Point Detector: Harris-Laplace

[Miko04] presents a scale-adapted Harris detector, which comes from previous work by the same authors. This is based on the Harris detector, but where the partial derivatives are computed on various scales of the image, by using Guassian filtered intensities. This detector thus uses scale-space, as was described in the introduction in section 2.3. The partial derivatives on Gaussian filtered intensities can be denoted $I_x((x,y),\sigma)$ and $I_y((x,y),\sigma)$. The size of these Gaussian filters is the *differentiation scale*, σ_D, of this detector. The derivatives are averaged in a neighbourhood of a point, which is done by Gaussian filtering at the *integration scale*. This response is computed over the entire image and for several resolutions. They select the *characteristic scale*, for each point of the image, by finding maxima of the response over the different resolutions at the same point.

They argue that, a better way of selecting scale is by using the Laplacian of Gaussians, often referred to as *LoG*. The Laplacian of Gaussians (LoG) is computed as the second order partial derivatives, often denoted I_{xx}, I_{yy} and I_{xy}, of Gaussian filtered image intensities. When the image intensities are Guassian filtered, the second order partial derivatives can more explicitly be denoted $I_{xx}((x,y),\sigma)$, $I_{yy}((x,y),\sigma)$ and $I_{xy}((x,y),\sigma)$, where σ specifices the size of the Gaussian filter. The Laplacian of Gaussians at scale σ_n is then defined by:

$$|LoG((x,y),\sigma_n)| \quad = \quad \sigma_n^2 |L_{xx}((x,y),\sigma_n) + L_{yy}((x,y),\sigma_n)| \tag{5}$$

From the starting point of the previous scale-adapted Harris detector, they describe their combined Harris-Laplace detector. In the Harris-Laplace detector, Harris-points are initially detected at coarse scale changes, a factor of 1.4 between each scale. Then the scale is selected iteratively as maxima over the Laplacian of Gaussian (LoG) over finer scale changes (factor 1.1 between each scale) until convergence. They present a simpler algorithm for this too, which is faster to compute, but less accurate. In both algorithms, both Harris-points and Laplacian of Gaussian (LoG) points are rejected, if the response values are below certain thresholds. They have good illustrations of features detected by their methods.

3.1.3 Affine Adapted Harris Point Detector: Harris-Affine

In [Miko04], they present the discrepancy between views of detected features obtained with the Harris-Laplace detector, which does not have affine invariance. They also present the general second moment matrix for affine scale-space. They derive the affine relationship between the second moment matrix of detected feature neighbourhoods and show that, this relationship transforms corresponding feature neighbourhoods into each other, except for an orthogonal transformation, i.e. a rotation or a mirror transform. They show how to determine the anisotropy by the eigenvalues of the second moment matrix and how this relates to differentiation and integration scales, as previously described in the introduction in section 2.3.

They describe their Harris-Affine interest point detector. They describe all elements of their algorithm very well. It is an iterative algorithm, which starts be detecting initial feature points by the multi-scale Harris detector. The detected feature points are then iteratively refined, by iterating the following five steps: 1) warp the local image window, according to the shape-adaptation matrix, 2) select integration scale as the local maximum of the normalized Laplacian, 3) select differentiation scale from the integration scale and the ratio between the smallest and largest eigenvalues, 4) detect spatial localization, by the maximum of the Harris measure and 5) compute the second moment matrix for updating the shape-adaptation matrix. They have good illustrations of this method. They afterwards detect if several of the initial feature points have converged to the same feature point and use only one of these final points.

3.1.4 Scale Adapted Harris Points and Adaptive Non-Maximum Suppression (ANMS) for the Multi-Scale Oriented Patches (MOPS) Method for Image Stitching; a Method Resulting in a Very Long Section Headline

The article [Brow04] use Harris points and a Gaussian image pyramid with a subsampling rate $s = 2$ and pyramid smoothing width $\sigma_p = 1.0$. Features are extracted at each level of the pyramid. They use $\sigma_i = 1.5$ as integration scale and $\sigma_d = 1.0$ as differentiation scale. Their features are are defined where the corner strength function has a local maximum in a $3x3$ neighbourhood and has a value above the threshold $t = 10.0$. They use 2D quadratic fitting to the corner strength function in a $3x3$ neighbourhood for subpixel accuracy.

They introduce an adaptive non-maximum suppression algorithm (ANMS), which extracts a fixed number of feature points, which are spatially well-distributed over the image. This is conceptually done by only returning maxima within a neighbourhood of radius r, where r starts being 0 and is increased, until the desired number of features is obtained. In practice, they do this in a more efficient way, since the feature points obtained in this way form an ordered list. They also make this even more robust by by adding criteria for when neighbours are significantly stronger than their neigbours.

They show experimentally that this adaptive non-maximum suppression (ANMS) improves the spatial distribution of features and that, on a large database of features, this gives fewer dropped image matches for panoramic images, which is important for image stitching applications.

They also evaluate the repeatability of their detector, showing that their Harris points are only accurate up to $1 - 3$ pixels distance. A few remarks, which the authors did not make, is that, this is for a global image homography, which does not take perspective into account. This is important, if the images are not taken from the same camera location, and the images don't appear to all be taken from the same location, which could explain this inaccuracy.

3.1.5 Hessian-Laplace and Hessian-Affine

[Miko05] uses the Harris-Laplace and Harris-Affine detectors already described. They also use a Hessian-Laplace detector, which uses maxima of the Hessian determinant to locate the position of points, rather than by the Harris detector, as Harris-Laplace does. Both Hessian-Laplace and Harris-Laplace select scale by using the Laplacian of Gaussians (LoG). They also use a Hessian-Affine detector, which is also similar to Harris-Affine, except for, once again, using the Hessian determinant to locate the position of points, rather than Harris points. The basic Hessian matrix is given by:

$$H = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \tag{6}$$

I_{xx}, I_{yy} and I_{xy} are the second order derivatives, as ealier. The same considerations of convolution with a Gaussian and differentiation scale, integration scale and scale-space also apply here, as for the previously described cases of the Harris-based detectors. The trace and determinant of the Hessian, when not considering scale-space, are thus given by:

$$Tr(H) = I_{xx} + I_{yy} \quad (7)$$
$$Det(H) = I_{xx}I_{yy} - (I_{xy})^2 \quad (8)$$

Notice here that, the trace of the Hessian corresponds to the Laplacian.

3.1.6 Fast Hessian Detector

[BayH06] introduces a new "Fast-Hessian" feature *detector* and gives a good overview of some detectors that it resembles. The Fast Hessian detector uses an approximated determinant of the Hessian for both scale and position detection. Their approximation is made by special 9x9 filters (containing a configuration of two negative boxes and two positive boxes), which mimic the Hessian with scale $\sigma = 1.2$. They show how they can support scale directly, by scaling their filters, without having to iteratively filter images with Gaussians, as done in a scale-space image pyramid. They make sure that, the method that they use for this is automatically scale-normalized. They detect their features, which they call interest points, by a non-maximum suppression algorithm (see e.g. [Neub06]) in a 3x3x3 neighbourhood over the image and in scale. They interpolate the detected maxima over the image and in scale as in [Brow02]. This is particularly important for this method, since there are large differences in scale between particularly the first layers of every octave in their scale-space.

3.1.7 Maximally Stable Extremal Regions: MSER

[Mata02] gives a good overview of the image correspondence problem. They introduce a region *detection* method for finding Maximally Stable Extremal Regions (MSER). Their method has near linear time-complexity and is fast in practice. The regions can be described as upper or lower level sets, whose rate of change of the boundary, as a function of the level threshold, has a local minimum between neighbouring levels. The regions are stable, are detected without image smoothing and detect both fine and large structure. They are invariant to affine transformation of image intensities, such as contrast changes, and are covariant (between images) to adjacency preserving (continuous) transformation of the image coordinates.

The method works by sorting pixels by intensity, which can be done in linear time, by using binsort, if the range of image intensity values is small, such as in an 8-bit intensity image. Then, two passes of inserting pixels into the image are done; in decreasing, respectively increasing, order of image intensity. During these two passes, a list of connected components and their areas is maintained, which can be done very efficiently with a union-find algorithm. This results in two data structures, each consisting of the areas of the connected components as a function of image intensity. The intensity levels, which are local minima of the rate of change of the area function, are selected as thresholds, for producing the maximally stable extremal regions.

3.1.8 Fast Level Set Transform: FLST

[Oshe06] is a general text book on level set methods, particularly for computer vision, medical imaging and graphics. Chapter 7, particularly section 7.3.1, with theoretical background and proofs in the

sections preceding it, presents a method called the Fast Level Set Transform (FLST), about which more information can be found in [Mona00] and, more recently, in the preprint of the book [Case08], particularly chapters 6 and 7. Chapter 15 of [Oshe06] explains image matching using the Fast Level Set Transform (FLST) with moments up to second order as region descriptors, where a similar matching algorithm is described in [Mona99]. It should be noted that, the above four references have some discrepancies as to how upper versus lower level sets are defined, which we will sort out later in this report, in section 6.2. Some of the above references also describe a related method, the Fast Level Lines Transform (FLLT).

The Fast Level Set Transform (FLST) works by segmenting the image into a hierarchical set of regions, called shapes, which gives a complete and non-redundant representation of the image. This is quite similar to the Maximally Stable Extremal Regions (MSER) method, except that, MSER tries to extract only stable regions, whereas FLST extracts regions for the entire image and forms a hierarchy of shapes. Indeed, chapter 8, particularly section 8.4, of the preprint [Case08] shows how to extract various kinds of Maximally Stable Extremal Regions from the FLST representation, by using different criteria, as well as methods for finding the most meaningful edges.

The FLST algorithm has many details, but works by growing regions (shapes) in a bottom-up fashion and maintaining counts of holes in the regions. The hole counts are maintained by Euler characteristics, calculated by tracking changes in region boundary configurations, and are used for merging detected shape subtrees into one final tree in the end. The algorithm can be implemented both by handling images as pixels of discrete intensities and by handling images as a continuous surface of intensities. In the latter case, shapes are extracted, as a final step, with any desired intensity discretization threshold, e.g. using 100 intensity levels.

Apart from being useful as a region detector, FLST also gives a contrast invariant representation of the entire image and chapter 7 of the book [Oshe06] gives a recipe for doing edge-preserving removal of image noise by Total Variation Minimization.

3.1.9 Edge Based Regions: EBR

[Tuyt99] presents a feature *detector*, which works by first finding corner points, then following two edges from such a corner point and finally fitting a parallelogram to the corner and the edges, in order to get a local affinely invariant image region. Their corner points are found with the Harris detector. They have a method for tracing edges from such a point, which includes a criterion for when to stop tracing the edges. This method is referred to as Edge Based Regions (EBR) in [Miko06], where it is compared to other methods. Their application in this article is database content retrival.

3.1.10 Intensity Based Regions: IBR

[Tuyt00] presents a feature *detector*, which they suggest to combine with the Edge Based Regions (EBR) method from [Tuyt99]. Their application here is wide-baseline stereo image matching. This method is somewhat similar to the Maximally Stable Extremal Region (MSER) detector, but predates it. They detect their regions by first finding intensity maxima with a non-maximum suppression algorithm. [Neub06] describes efficient ways of implementing this. From these intensity maxima, they trace out lines in a circle from that point, to establish a region. They have a criterion for when to stop tracing each line, based on maxima of image intensity changes. They fit an ellipse onto the traced area and double the size of it, for use as their measurement area. This method is referred to as Intensity Based Regions (IBR) in [Miko06], where it is compared to other methods. The kinds of features detected in this article (IBR) are thus quite different from the kinds of features detected in [Tuyt99] (EBR). They suggest that,

combining several such feature detectors is a good idea, since it yields more correspondences and since it can enable a system to detect features in a wider range of images.

3.1.11 Other Detectors

[Lowe04] describes how to compute scale-space with Gaussian kernels and how to compute Differences of Gaussians (DoG) for octaves of scale-space by image subtraction. It is argued how Differences of Guassians (DoG) approximate the scale-normalized Laplacian of Gaussians (LoG). Methods like these are also thoroughly described with source code examples in chapter 5 of [Cyga09]. For feature detection, they use Differences of Gaussians. Detailed location, scale and ratio of principal curvatures of each feature is fitted to the nearby image data with a 3D quadratic function. This fitting gives a substantial improvement to matching and stability and can be used to discard unstable features with either low contrast or a high ratio between the principal curvatures.

3.2 Feature Descriptors

3.2.1 Scale-Invariant Feature Transform: SIFT

[Lowe04] describes a method known as the Scale Invariant Feature Transform (SIFT), where they use Differences of Gaussians (DoG) with fitted location, scale and ratio of principal curvatures for feature detection, as described earlier. For the *descriptor*, an orientation for each feature is built from a histogram of gradient orientations, weighted by gradient magnitude. If the histogram of a detected feature contains multiple peaks of orientations, several features are created, which contributes significantly to the stability of matching. Their descriptor consists of a 4x4 array of 8-direction orientation histograms. This array is weighted with a Gaussian function, to compensate for sudden changes in position. The feature vector is modified by two normalization steps, in order to make it invariant to changes in illumination, especially contrast. This feature vector has 128 dimensions. Different feature vector sizes have been experimentally tested against a database of 40.000 detected features, taken from 32 images, and this feature vector size gave the best results. Reliable matching for up to 50 degrees of viewpoint change is achieved. It is argued that, affine matching is often not worthwhile, since it is usually not the limiting factor for 3D objects and since the stability towards small affine changes is decreased. For affine invariance, they recommend the approach by [Prit03] of using SIFT features for 4 affine transformed versions of the training image. Feature *distinctiveness* is shown to be almost unlatered with the growth of their database for object recognition, meaning that the features are highly distinctive in general.

3.2.2 Speeded Up Robust Features: SURF

[BayH06] introduce their Speeded Up Robust Features (SURF) *descriptor*, which they use with their Fast Hessian detector, as described earlier. The descriptor comes in two variants, depending on whether rotation invariance is desired or not. The rotation invariant descriptor first assigns an orientation to the descriptor and then defines the descriptor within an oriented square. The other version, called U-SURF, for Upright-SURF, which is not rotation invariant, simply skips the orientation assignment phase.

The orientation assignment for the descriptor starts by calculating Haar-wavelet responses around the detected feature. This calculation is done at the detected scale of the feature and the responses are weighted by a Gaussian, centered at the feature point. The responses are represented as vectors, which are used to find a dominant orientation.

When they have defined a square for the orientation frame, by either orientation assignment or an upright orientation, the descriptor is defined within this oriented square. This is done by dividing it

regularly into 4x4 subregions and for each subregion, compute simple features at 5x5 regularly spaced sample points. For each such sample point, they compute 4 descriptor values. These are the sums of Haar-wavelet responses and sums of absolute Haar-wavelet responses, both in the x- and y-directions of the oriented frame. This yields a 64-dimensional descriptor, which is made contrast invariant by normalizing it into a unit-vector. They have experimented with alternatives of this descriptor and found that this particular one gives the best results.

They also define a SURF-36 descriptor, using only 3x3 subregions. This is faster to compute and performs only slightly worse. Finally, they define a SURF-128 descriptor, which divides the sums of the wavelet responses into twice as many sums, making it just slightly slower to compute, but also slower to match, due to the higher dimensionality. It performs better than the regular SURF descriptor and has the same number of dimensions as SIFT.

For the matching phase, they introduce indexing by the sign of the Laplacian, which gives a slight performance increase. They evaluate their descriptors and compare them with other well-performing descriptors, showing that their descriptor outperforms the others. They also evaluate their Fast Hessian detector and compare that with other well-performing detectors, where their performance is good. Which feature detector is best, seems to depend on image content. Their detector and descriptors are clearly the fastest to compute though.

3.2.3 Multi-Scale Oriented Patches: MOPS

In [Brow04], they determine an orientation for their detected feature points from the smoothed gradient around the point. The use the integration scale $\sigma_o = 4.5$ to make the orientation vary smoothly accross the image, for greater robustness.

Their feature descriptor is a locally oriented patch around the feature point, sampled at $8x8$ pixels at scale $5s$, where s is presumably the scale of the detected feature. They have verified experimentally that, $5s$ is a good sampling scale. They normalize the patch, to get affine intensity invariance, and then transform it with the Haar-wavelet transform. The first three non-zero Haar-wavelet coefficients, c_1, c_2 and c_3, are used in an indexing strategy.

3.2.4 Other Descriptors

[Tuyt00] use an 18-dimensional descriptor of second order generalized colour moments. [Oshe06] Chapter 15 also uses second order moments as a descriptor, but without taking colour into account.

[Mata02] use rotational invariants as descriptors, based on complex moments, where they diagonalize the regions' covariance matrix. This is an affine method. They have verified that, rotational and affinely invariant generalized colour moments gave similar results. On their own, these affine invariants failed on problems with a large scale change.

[Miko04] uses Gaussian derivatives on their detected affine area, up to 4th order, as their 12-dimensional descriptor. They divide the higher-order derivatives by the 1st order derivatives, to achieve affine invariance to intensity changes. They use the average gradient orientation to orient the descriptor, in order to get rotation invariance.

[Miko05] introduces the Gradient Location and Orientation Histogram (GLOH), which extends the SIFT descriptor by changing the location grid and using Principal Component Analysis (PCA) to reduce its size.

There appears to be at least two general ways of handling rotation invariance: 1) try to find and assign an orientation for the descriptor, thus using the orientation as part of the descriptor or 2) try to ignore the rotational aspect. Methods like SIFT and SURF use the first strategy, where as e.g. a

method like SPIN, which we will not go through, uses the second. The first strategy intuitively seems more powerful and distinctive, which is in some sense confirmed by comparisons of methods, as in e.g. [Miko05].

3.3 Feature Matching Strategies

In [Miko04], they match each descriptor in one image by the Mahalanobis distance measure, in order to find its most similar descriptor match in the other image. They accept this as an initial match, if the distance between the descriptors is below a certain threshold. After this, they reject low score matches by cross-correlation of affine-normalized image patches. They select inliers among the correspondences by RANSAC. They estimate either a homography or a fundamental matrix, where a model selection algorithm can be used to decide which transformation is the most appropriate.

[Dufo02] propose establishing a frame in a low-resolution image, which matches a high-resolution image. The image points are related with a homography, which is specialized into having only a similarity factor, rotation and translation. This is used for correlating points, where the correlation is minimized with non-linear least squares optimization. They do feature-based matching and derive the Harris feature point detection and auto-correlation matrices between features. They desribe and evaluate how scale-space is handled. The matching is done with one scale being separately matched to 8 different scales. They use differential invariants as descriptor vectors and match with the Mahalanobis distance measure. They describe how to match a collection of points in a neighbourhood, which are consistent with a similarty transform. This consists of: 1) match centre and one other point, 2) find the similarity, 3) verify other points' consistency and 4) repeat as a depth-first tree search. Each scale-space representation is matched, in order to find the approximate difference in scale. RANSAC is used for finding inliers and rejecting outliers. They show experiments of matching at several scales. They also examplify that, it works for differing viewpoints as well. In particular, they support finding either an affine transform, a projective transform (a plane homography) or epipolar geometry (a fundamental matrix).

The application in [Brow04] is image stitching, which may not be directly comparable to to image matching, but the methods are still relevant. Feature matching is done by an approximate nearest neighbour algorithm, based on an indexing scheme of the first three Haar-wavelet coefficients: $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial^2 I}{\partial x \partial y}$. They distribute these coefficients into a 3-dimensional bin-array with 10 bins in each dimension, thus covering $\pm n_\sigma = 3$ standard deviations from the mean of those dimensions. Then, outliers are removed, based on noise statistics of correct versus incorrect matches. Finally, RANSAC is applied for geometric constraints and remaining outlier rejection. They found that, nearest neighbour neighbour thresholding is inferiour to nearest and second nearest neighbour ratio thresholding. They also found that, the ratio between the nearest neighbour and the average of of all second nearest neighours in all images, is an even better criterion. They successfully tested their method on hundreds of panoramic images for stitching.

In [Lowe04], they show the application of their methods to object recognition. They use nearest neighbour matching and show that, the ratio between the nearest neighbour and the second-nearest neighbour can be used to filter out unreliable matches, if these two are close to each other. However, their results on this is somewhat specific to object recognition, since they only consider potential second-nearest neighbours from a different object than the nearest neighbour. They describe a way to do efficient approximate nearest neighbour search for a large database of features. They also describe how to find clusters of features with the Hough transform, which allows for affine transformation estimation, which can be done with only 3 features. The affine transform between images is determined by iterations of least squares minimization, outlier filtering and top-down addition of possible matches. They identify clusters

of at least 3 features agreeing on object and pose, before attempting to match features. Matching examples are made, where a priori known objects can be efficiently recognized, despite background clutter, extensive occlusion, illumination changes and viewpoint changes.

In [Mata02], they perform robust matching, by comparing each region of an image to k regions in the other image and then using a voting scheme. They set k to 1% of the total number of regions in the image. They use 216 invariants at each scale, i.e. 864 invariants in total, since they select measurement areas at four different scales of their detected regions, at factors of 1, 1.5, 2 and 3 times the convex hull of the detected regions. The invariant description is used as a preliminary test. The final selection of tentative correspondences is based on image intensity correlation. Epipolar geometry is estimated by using RANSAC on the centres of gravity of the detected regions. They improve on the precision, by pruning correspondences with cross-correspondences and then applying RANSAC again with a narrow threshold. Detected region pairs, whose convex hull are consistent with the epipolar geometry, are added to the RANSAC inliers. The final epipolar geometry is estimated linearly by the 8-point algorithm. This gives a precision with a distance of less than 0.1 pixel from the epipolar line. They have experiments showing success for several difficult cases, including some from other work.

In [Oshe06] in chapter 15 and in [Mona99], they also use a voting scheme for their matching.

3.4 Work Comparing Methods

[Miko04] evaluate their Harris-Laplace and Harris-Affine feature *detectors* and compare them with some other feature detectors: Differences of Gaussians (DoG), Laplacian, Gradient and Hessian. They describe their repeatability criterion and their error thresholds for their evaluation. The evaluate for scale changes, where their comparison is with all of the above methods. They also evaluate affine invariance, but only for their own two detectors and for Differences of Gaussians (DoG) and another method, referred to as Harris-AffineRegion. They also evaluate the location accuracy for those four detectors. They mention a 0.5 percent discrepancy in the number of detected features, due to using accelerated recursive Gaussian filtering. They also present matching results, where Harris points and even multi-scale Harris points are not good enough for RANSAC to succeed. They show successful matching with Harris-Laplace and Harris-Affine, where the former is the most effective for scale changes and the latter the most effective for viewpoint changes. They also show a failure case, which seems to be mostly due to using poor descriptors; 12-dimensions of Gaussian derivitives up to 4th order.

[Miko05] makes a thorough comparison between several feature *descriptors*. The descriptors they consider are: GLOH, a contribution of that article, SIFT, PCA-SIFT, SPIN, gradient moments, cross correlation, complex filters, shape context, differential invariants and steerable filters. Some of these are lower dimensional and some are higher dimensional (FIXME: Specify which). These methods are considered for the follwing region *detection* methods: Harris points, Harris-Laplace, Hessian-Laplace, Harris-affine and Hessian-affine. They compare the discriptors accross different *matching* strategies: Thresholding, nearest neighbour matching and nearest neighbour distance matching. The comparisons are made on an image data set, which contains differences in rotation, scale, viewpoint, illumination, blur and JPEG compression, with many of these differences comparable for both structured and textured scenes. This data set is available on `http://www.robots.ox.ac.uk/~vgg/research/affine`.

[Miko06] compares different affine region *detectors* on the same image data set as in [Miko05], which they aim to establish as reference data for comparison between methods. They compare the detectors: Harris-affine, Hessian-affine, maximally stable extremal regions (MSER) [Mata02], an intensity based region detector (IBR) [Tuyt00], an edge based region detector (EBR) [Tuyt99] and Salient regions. They fit ellipses, using second order moments, around the detected regions for all methods and use that as their *distinguished* region, even though fitting ellipses is normally not part of the MSER and EBR detectors.

The detectors Harris-affine, Hessian-affine and MSER mostly detect small regions, whereas IBR, EBR and Salient regions detect larger regions. The performance of the region detectors are compared by overlap tests, where they evaluate the performance parameters: repeatability and accuracy. The also test region *matching* with the SIFT descriptor, where they evaluate the parameters: matching score and number of correct matches. They also investigate, by separate tests, how changing some of the fixed parameters of the tests affect the performance, thereby showing that the choice of their parameters are sensible.

3.5 General Presentations and Tutorials

[Cyga09] is a general text book on computer vision and as such, covers a wide range of relevant topics. Among other things, it describes methods based on blurring and scaling images, e.g. to form an image pyramid for handling scale-space. For this, it analyses and compares various methods, such as Laplacians of Gaussians (LoG) and Differences of Gaussians (DoG) in detail. It generally covers many details of the basic methods for image analysis and feature detection, but it falls short on covering the more modern feature detectors and descriptors, which are presented in this report. It also describes geometric and camera registration methods and gives an overview of distance measures and many other things. It is quite focused on narrow-baseline correspondences and dense matching methods. Many of the described methods include source code examples.

[Poll00] is a book made as notes for a course, which describes the entire computer vision process for going from input images to obtaining a 3D model from the images. It uses a pipeline based on feature matching and supporting automatic camera calibration. It contains descriptions of the many methods needed along the way and gives a good overview of the process, but does not try to compare many different approaches, although it does present some possible choices. For feature detection and descriptors, it (only) describes Harris points, simple image intensity cross-correlation and the edge based region detector (EBR) described in [Tuyt99]. It has extensive descriptions of geometric methods for camera registration and camera self-calibration.

[Hart03] is a very thorough book on the geometry and methods used and needed for computer vision, particularly projective geometry and camera registration for multiple views. This book is highly recommended for its treatment of these subjects and is particularly useful as a reference book. This book is, however, not very concerned about image processing, feature detection or feature descriptors.

[Szel06] is a tutorial on image alignment and stitching, which also present and use many of the same methods, which are being reviewed in this report. It describes, or at least refers to, both direct pixel based methods and feature based methods for finding image correspondences. The tutorial also describes some methods for camera registrations, including global camera registration, for using more than two images for camera registration. The application focused on in the tutorial is image alignment for stitching, so some of its general conclusions may not be the same in the context of this report.

4 Conclusions of Previous Work

This is a compilation of some important conclusions, drawn in previous work. Many of these were not part of the previous section. This forms a solid basis for choosing between and combining methods.

4.1 Conclusions regarding the Contents of Input Images

- Viewpoint changes is the hardest image transformation to deal with, among the ones considered here, followed by scale changes. This applies to both *detectors* and *descriptors* ([Miko06] section

5.2 and [Miko05] section 4.2). [Dufo02] achieves matching between images, which differ in scale by a factor of up to 6, where their method is specialized for handling large scale changes

- Texture reduce *detector* repeatability ([Miko06] section 4.2) and reduce the *matching* scores significantly (possibly due to repeated content) ([Miko06] section 5.2 and [Miko05] section 4.2)
- Rotation changes do not affect most *descriptors* ([Miko05] section 4.3)
- Blurry images degrade the performance of most *descriptors*, since they are not robust to this kind of transformation, more so for the Harris-affine detector than the Hessian-affine detector ([Miko05] section 4.4). However, most *detectors* handle this quite well, with MSER being the worst ([Miko06] section 5.2)
- JPEG compression degrades performance of *detectors*, but less than blur and mostly at more extreme compression rates ([Miko06] section 5.2). It degrades *descriptor* performance, less than for blur, but more than for rotation and scale changes on structured scenes ([Miko05] section 4.5)
- Illumination changes degrades *descriptor* performance, particularly if the methods do not try to compensate for such changes by e.g. intensity normalization ([Miko05] section 4.6). *Detectors* are generally quite robust to this kind of change ([Miko06] section 5.2)

4.2 Conclusions regarding Direct Pixel-Based Methods versus Feature-Based Methods

- Section 4.4 in [Szel06] describes some trade-offs between direct pixel-based methods and feature-based methods, which, however, is meant for the application of image alignment and stitching. It is argued that, some of the older feature-based methods were less impressive than the direct methods, at the time that those methods were invented. It is also argued that, the more recent feature-based methods are very robust and can handle many difficult cases, some of which may even be hard for direct approaches. Our focus is on feature-based methods, so direct methods will not be investigated any further

4.3 Conclusions regarding Feature Detection

- Detection results generally depend on the type of scene, where some detectors perform better on certain types of scenes, whereas other detectors for other scene types, illustrating that combining detection methods would be a good idea (the introduction and sections 4.2 and 6 in [Miko06], the conclusion in [Lowe04] and [Tuyt00]). The differing performances between the detectors compared in [BayH06], depending on the image content, also suggests that combining detection methods would be useful
- Feature detectors handling affine invariance can generally handle larger changes in viewing angle, but this may come at a cost of their location accuracy and their ability to handle e.g. large scale-changes (section 4.1 in [Miko04]). It is argued in [Lowe04] that, affine matching is often not worthwhile, since it is usually not the limiting factor for 3D objects and since the stability towards small affine changes is decreased. They claim to handle viewpoint changes up to 50 degrees reliably. This might, however, be achieved partly by their method estimating the affine transform by using clusters of 3 features (see below). For affine invariance, they recommend the approach by [Prit03] of using SIFT features for *4 affine* transformed versions of the training image. A similar

way, but with *20 projectively* transformed versions of the image, is suggested in section 15.7.4 of [Oshe06]

- In [Lowe04], they identify clusters of at least 3 features agreeing on object and pose, as a way of estimating the affine transformation, before attempting to match features
- Lower feature density (i.e. number of detected regions per image area) usually means, that the detected features are more stable, which usually increases performance, by giving a higher repeatability (section 4 in [Miko06]). It is also argued at the end of section 5.2 in [Miko06] that, a good detector with a tight threshold should result almost exclusively in correct matches
- Increasing the size of the *measurement area* by some factor of the detected feature increases the descriminative power of a detector, where a factor of 3, or possibly 2, is shown to be a sensible scaling factor in section 5.1 in [Miko06]. In section 3 of [Mata02], they select measurement areas of *several size factors* of the detected feature: 1, 1.5, 2 and 3
- Increasing the size of the *measurement area* generally increases the overlap in overlap tests, giving a false sense of success, which should be compensated for in such tests (section 4 in [Miko06])
- Among the region detectors compared in [Miko06], MSER and Hessian-affine generally performs best, except that the performance of MSER decreases significantly for blurry images ([Miko06] section 4.2). Hessian-affine and Harris-affine are however found to be less discriminating detectors than some of the others ([Miko06] section 5.2). MSER and IBR generally performs best in the presence of homogenious regions with distictive boundaries (section 6 in [Miko06]). Hessian-affine and Harris affine generally provide more regions, which is useful for scenes with clutter and occlusions (section 6 in [Miko06]). Salient regions generally performs the worst, for this application
- Hessian feature detectors are slightly better than Harris feature detectors, due to their higher precision, according to [Miko05]. Hessian-affine is also shown to generally outperform Harris-affine in section 4.2 of [Miko06]. This is also mentioned in section 2 of [BayH06]
- MSER generally outperforms IBR (section 4.2 in [Miko06])
- Among the feature detectors compared in [Miko06], MSER is the most robust for illumination changes, but all the considered detectors are generally robust for such changes (section 4.2 in [Miko06])
- According to section 4.1 in [Miko04], scale-changes are handled best by their Harris-Laplace detector, followed by the Hessian detector and then their Harris-Affine detector, among those three methods. The performance of Harris-Affine, however, degrades with increasing differences in scale, where it is eventually outperformed by Laplacian and Differences of Gaussians (DoG). The location accuracy is is lower for their simplified Harris-Laplace than their full version of Harris-Laplace
- According to section 4.1 in [Miko04], smaller viewpoint changes below 40 degrees are handled best by the Harris-Laplace method, but it gets outperformed by Harris-Affine above 40 degrees. The location accuracy is also best for Harris-Laplace below 40 degrees of viewpoint change, but is outperformed by Harris-Affine above 40 degrees
- According to section 5.2 in [Miko04], Harris points or even multi-scale Harris points are not good enough for RANSAC to succeed, but Harris-Laplace and Harris-Affine are

- In [Brow04] they show experimentally that, their adaptive non-maximum suppression (ANMS) for their multi-scale Harris points improves the spatial distribution of features and that, on a large database of features, this gives fewer dropped image matches for panoramic images, which is important for image stitching applications

- It is argued in [Lowe04] that, Differences of Guassians (DoG) approximate the scale-normalized Laplacian of Gaussian, denoted $\sigma^2\nabla^2 G$, and that normalization by σ^2 is required for true scale-invariance

- [Lowe04] section 3 argues (by reference to [Miko02]) that, the maxima and minima of $\sigma^2\nabla^2 G$ produce the most stable image features, among methods such as gradient, Hessian or Harris corner function. However, extrema which are close together are quite unstable to small image perturbations ([Lowe04] section 3.1) and it is experimentally shown that, sampling 3 scales per octave gives the highest repeatability (section 3.2 in [Lowe04]). It is also experimentally shown in section 3.3 of [Lowe04] that, a Gaussian smoothing by a factor of 1.6, before constructing each scale-space octave, gives close to optimal repeatability, and this is the same factor, which is proven to be theoretically optimal in [Cyga09]. Section 3.3 in [Lowe04] also argues for the creation of an extra scale-space octive, by doubling the resolution of the input image

- In section 3 of [BayH06], it is mentioned that, Guassians are optimal for scale-space analysis, with reference to [Koen84]. However, they also argue that, in practice, this is overrated, since 1) in practice, the filters need to be cropped, 2) aliasing still occurs with Guassian filters when subsampling images and 3) new structures can appear when going to lower resolutions in the 2D case, even though they can't in the 1D case

- The Fast Hessian detector, like Hessian-Laplace, uses the trace of the Hessian matrix for scale-selection. In section 2 of [BayH06] they argue that, using its determinant instead of its trace (the Laplacian) seems advantageus, since it is less triggered by elongated or ill-localized structures

- It is not obvious which of the detection methods Fast Hessian, Differences of Guassians, Hessian-Laplace and Harris-Laplace, which are compared in [BayH06], is the best performing. It depends on image contents and all perform well, but for example, Harris-Laplace is never seen to perform the best in those comparisons

- Computational speed: Among the region detectors compared in [Miko06], Salient regions is computationally very slow and EBR is fairly slow as well, while the other methods are quite fast, with MSER being the fastest. The detectors Hessian-Laplace, Harris-Laplace, Differences of Guassians and Fast Hessian are all fast in [BayH06], with Fast Hessian clearly being the fastest, Differences of Guassians next, Hessian-Laplace third and Harris-Laplace clearly being the worst. In [Miko04], Harris-Affine is the slowest to compute, but a faster algorithm is available, at the expense of its detection performance. The detectors Differences of Gaussians and Hessian are fast in this comparison

4.4 Conclusions regarding Feature Descriptors

- The general strategy of finding and assigning and orientation to a feature for rotation invariance (as in e.g. SIFT and SURF) intuitively seems more powerful and distinctive, than disregarding orientation dependent data (as in e.g. SPIN), which is in some sense confirmed by comparisons of methods, as in e.g. [Miko05]

- Disambiguating matches on descriptors alone is not sufficient and some additional methods of filtering out regions is needed. Methods for this could be geometric filtering based on local spatial arrangement of the regions or on multiple view geometric relations (section 6 [Miko06]). [Lowe04] gives some specific methods for this. Another important way is doing image intensity cross-correlation. The cross-correlation is often done on normalized (e.g. with respect to affinity, scale, rotation) image patches (e.g. [Miko04], [Mata02] section 3 and [Tuyt00] section 3.2)

- Simple (i.e. unnormalized) cross-correlation of intensities as a descriptor gives unstable results, according to [Miko05]

- Among the *descriptors* compared in [Miko05]: GLOH and SIFT performs best, demonstrating the robustness of the SIFT descriptor. Shape context also performs well, but less good for textured scenes or when edges are unreliable. Gradient moments and steerable filters are best among the lower dimensional descriptors. Among the least efficient methods is differential invariants

- [BayH06] compares some of the well-performing descriptors and they found out that, the SURF descriptor, particularly SURF-128, outperforms the descriptors GLOH, SIFT and PCA-SIFT. They argue that, GLOH is more distinctive than SIFT, but also computationally more expensive. They also argue that, PCA-SIFT has lower dimension than SIFT, but is also less distinctive. Their comparisons are not with the affine invariant versions of the descriptors, so these results are not directly comparable to [Miko05]

4.5 Conclusions regarding Feature Matching

- *Nearest neighbour based matching* gives higher precision than threshold based matching, according to [Miko05]. SIFT gives relatively better results if nearest neighbour distance is used for thresholding, according to [Miko05]. In [Lowe04], they demonstrate that, the success of nearest neighbour matching decreases only slowly with the increase in size of a large database of features for object recognition

- [Lowe04] show that, the *ratio* between the *nearest neighbour and the second-nearest neighbour* can be used to filter out unreliable matches, if these two are close to each other. They found that, accepting only matches with this ratio being 0.8 or less gives good results. However, their results on this is somewhat specific to object recognition, since they only consider potential second-nearest neighbours from a different object than the nearest neighbour. In [BayH06], SURF and particularly SURF-128 performs best, out of the evaluated descriptors. This is for both similarity based matching and nearest neighbour to second nearest neighbour ratio matching. Nearest neighbour to second nearest neighbour ratio matching performs best ([BayH06] figure 4 and section 5) and they use the ratio threshold 0.7

- [Brow04] (section 5) found that, nearest neighbour thresholding is inferior to nearest to second nearest neighbour ratio thresholding. They also found that, the ratio between the nearest neighbour and the *average of of all second neighours* in all images, is an even better criterion, which eliminates 80 percent of the false matches, at the cost of less than 10 percent correct matches being lost

- Many methods use some kind of *indexing step* for approximate matching, in order to speed up the matching. E.g. in [BayH06], they introduce indexing by the sign of the Laplacian and in [Mata02],

using lower versus upper level sets (which they call positive and negative regions) achieves something similar. In section 6 in [Brow04], the first three non-zero Haar-wavelet coefficients are used in an indexing strategy. The approximate matching from [Brow04] give them a speed-up factor of 125, for a loss of less than 10 percent of the correct matches. They show that, this gives a 10 percent better recall than indexing on random dimensions (which are pixels) of their descriptor. The particular results from [Brow04] are for the application of image stitching

- [Brow04] (section 7) experimentally found that, doing Lucas-Kanade refinement ([Luca81]) on features actually harms the distinction between correct and incorrect matches
- There are methods to determine whether it is most appropriate to establish a fundamental matrix or a homography, according to section 5.1 in [Miko04]

5 Design Analysis

In this section, the main design descision will be made. Most of the decisions are based on the conclusions of previous work, as well as the intended application and the time constraints of this project.

5.1 Pipeline Considerations

We will start by taking a look at an overall pipeline proposal for feature matching and camera registration. The complete pipeline will not be implemented, but the overview identifies some desired properties of the parts that will be implemented.

5.1.1 A Simple Pipeline

A simple pipeline for feature matching and camera registration can consist of the following sequence of phases, starting from the top:

- Image input
- Image preprocessing
- Feature detection
- Establish measurement areas from the detected features
- Compute feature descriptors
- Feature matching on descriptors
- Camera registration

In this pipeline, the feature detection and descriptor matching methods have to be quite good. They must be able to match features with wide baseline and with few assumptions about the input images. It must also have a high position accuracy in the matched features, to give a precise camera registration.

The detection and matching parts of the pipeline can be done with more than one feature detection and matching method, where the methods are used side-by-side. The motivation is to increase robustness, thus creating an opportunistic system, as in [Tuyt99] and [Tuyt00]. A good reason for considering this is that, image content may vary quite a lot and the performance of different methods is likely to depend on the image content, as argued in section 4.

5.1.2 A More Advanced Pipeline

A more advanced pipeline, which has two rounds of feature detection and matching, can also be used. Such a pipeline would consist of the following sequence of phases:

- Image input
- Image preprocessing
- Wide-baseline feature detection
- Wide-baseline feature matching
- Approximate camera registration
- Possibly: Compensation for camera registration on input images, e.g. by image warping, possibly both ways for an image pair
- Possibly: Image preprocessing
- Feature detection, which can be narrow-baseline, in case the image compensation was done
- Narrow-baseline (in case the image compensation was done) feature matching, or, alternatively, feature matching guided by the approximate camera registration
- Accurate camera registration

This pipeline contains two rounds of feature detection and matching. The first of these must in particular be good for matching features with wide-baseline and with as few assumptions about the input images as possible, but it may compromise on the location accuracy of the correspondences. The second round of feature detection and matching must have a high location accuracy, but it can be guided by an approximate camera registration, which can estimate some feature correspondence parameters, such as in particular the corresponding feature location, but also scale factor, rotation, affine geometry transformation and illumination changes.

As in the simple pipeline, each of the two rounds of detection and matching parts of the pipeline can be done with more than one feature detection and matching method, with each method used side-by-side. This may particularly be useful in the first round of the detection and matching, before the approximate camera registration, since this stage of the pipeline has the fewest assumptions about the input images.

The pipeline proposed here only considers two overall rounds of feature detection and matching. However, multiple passes and iterations can be used, so this is just a manageable example of a pipeline.

5.2 Desired Properties of the Methods for the Pipeline

This report will, due to time constraints, be limited to feature detection, determining measurement areas, computing feature descriptors and simple feature descriptor matching. Hence, we will not build a complete pipeline. Also, only one feature detector will be implemented. The primary focus will be on the feature detector and the feature descriptor, since these are the key non-trivial methods.

Since this report focuses on wide-baseline image correspondences, we will focus on methods, which are suitable for particularly the first round of feature detection and matching in the pipeline. This means that, the detector may sacrifice some location accuracy but that it has to work without any assumptions

on the input image. Both the feature detector and the feature descriptor should generally be as robust as possible. The assignment of measurement areas defines the interface between the detector and descriptor methods, so they have to fit each other.

For the feature matching, we will only do simple brute-force matching.

5.3 Feature Detection

With the summary of arguments from section 4, we can conclude that, among detectors, which handle geometric affine invariance, Hessian-Affine and Maximally Stable Extremal Regions (MSER) seem to be the best methods. MSER does not handle blur as well as Hessian-Affine, but on the other hand, it should be a lot faster to compute. Implementing the Fast Level Set Transform (FLST) is also interesting, since it seems comparable to MSER and offers additional features, but we have no performance comparions and FLST seems more involved to implement. The methods with affine invariance are needed, if handling viewpoint changes of more than e.g. 40 degrees is desired.

If we limit the allowed viewpoint changes to 40 degrees, methods like Hessian-Laplace, Fast Hessian or Differences of Gaussians (DoG) seem to be better (see section 4). Which one of these is best, is not clear. Fast Hessian seems to be both the fastest and less memory intensive and DoG seems to be faster than Hessian-Laplace.

As already argued (section 4 again), combining detectors seems to be a good idea, particularly because their relative performance depends on image content. Combining detectors, which detect different *kinds* of features, thus seems to make most sense. E.g. for a system handling affine invariance, combining Maximally Stable Extremal Regions (MSER) and Hessian-Affine would be interesting, since MSER detects bounded regions, based on image intensities, and Hessian-Affine detects corners. However, to limit the scope of this report, only one detector will be implemented.

Maximally Stable Extremal Regions (MSER) will be the detecor being implemented, since it is able to handle affine invariance and should work well.

5.4 Feature Descriptor

From the conclusions in section 4, the Speeded-Up Robust Features (SURF) descriptor seems to be a good choice. It is very distinctive, uses little memory and does not seem very difficult to implement. The descriptor from the Scale Invariant Feature Transform (SIFT) is also good, but does not seem as good as SURF. SIFT-PCA and GLOH are extended and slightly better versions of SIFT, but they still don't outperform SURF and in being extensions of SIFT, the implementation effort for these methods most likely becomes bigger. Using generalized colour moments, as defined in [Tuyt00], seems interesting, since it is a low-dimensional and simple descriptor. Gradient moments, which should be comparable to, if not the same as generalized colour moments, seems to be one of the best performing low-dimensional descriptors, according to [Miko05]. However, SURF is still better and [Mata02] already uses moments as a descriptor, so implementing that again for the same detector (MSER) as in [Mata02], would not give any new contributions in this report. Hence, the choice is the SURF descriptor.

We will not be using the Upright version of the feature descriptor (U-SURF), since we want to support input images taken with a hand-held camera, where the camera may not be upright. Features may also rotate for other reasons than the camera rotating, e.g. due to perspective changes of tall objects seen from various angles from the ground level, such as the corners of the temple in the pictures in figure 5, which we will be considering later.

We will use the high-precision descriptor, SURF-128, rather than the regular SURF descriptor or the SURF-36 descriptor. The reason is that, SURF-128 gives better results and the only advantages of using

the other descriptors are improved computation speed, which we are not too concerned about in this project.

5.5 Measurement Areas

For affine invariance for the Maximally Stable Extremal Region (MSER) detector, fitting an ellipse to the detected region, by using second moments (see e.g. [Miko04], [Tuyt00] and chapter 15.2.2 in [Oshe06], for inspiration on how to do this), seems to be an obvious choice of measurement areas. However, the SURF descriptor, as it is described in [BayH06], does not support elliptical measurement areas, so it would have to be modified somehow, in order to support that. We will look at suggestions on how to do this in section 7.4, but the implementation will be limited to using circles as measurement areas.

Actually, using circular measurement areas removes one of the primary advantages of the MSER detector, namely that, it quite naturally handles affine invariance, by using the shape of the detected region, so this is actually quite a miserable limitation. However, it will not be a big effort to extend the implemented methods to handling affine invariance, so for the long-term intentions with this report, which is creating a commercial application, the chosen methods are still very relevant.

5.6 Descriptor Matching

In a real implementation, a clever data structure or a fast approximate heuristic for the feature descriptor matching should probably be used, rather than brute-force matching. However, approximate heuristics would not be as good as the brute-force approach for evaluating the performance of the feature detector and the feature descriptor. The detector and descriptor are the primary components evaluated in this report, which is why the brute-force matching is a simple and actually better choice for this report.

As it turns out, what is referred to as brute-force matching in the litterature actually has different options, as we shall see in the more detailed implementation description. Overall, it is possible to do either one-way matching or two-way matching. One-way matching is appropriate for tasks like object recognition. For matching two input images with each other, two-way matching seems more appropriate. Thus, two-way matching will be implemented, but not evaluated.

For the performance evaluations however, we will use one-way matching, instead of two-way matching. The reason is that, two-way matching turns out to be inappropriate for computing the performance measures. It can result in repeatability values for the detector and recall values for the descriptor, which are either above 100 percent or overly pessimistic. It also seems that, [Miko04], [Miko05] and [Miko06] uses one-way matching in their evaluations, even though this may not be stated directly.

As evident from the conclusions in section 4, it seems most relevant to do nearest neighbour matching, rather than threshold based matching. There is also much evidence pointing to that, using the ratio between the nearest match and the second nearest match is good for filtering out bad matches. For the chosen descriptor, Speeded Up Robust Features (SURF), the authors use the criterion that, this ratio must be less than 0.7 ([BayH06]). The article [Lowe04] uses 0.8, but this is for another descriptor, the Scale Invariant Feature Transform (SIFT). Hence, we will use 0.7 and, in order to limit the scope of the report, we will not evaluate this criterion any further. It should be noticed that in [Brow04], they suggest replacing the second nearest match with the average of the second nearest matchs from all other images, which seems like a very good idea. However, we only consider two images in this report, so we will leave this for suggested future work.

5.7 Considering Colours

The methods which are implemented are described only for black and white image intensities in the cited references. Therefore, and to limit the scope of this report, we will not consider taking advantage of colour information in the presented implementation. Proposals are given for extensions to colours in section 7.7.

6 The Implemented Methods

This section describes the methods implemented, including the key implementation details. The set of implemented methods can be summarized as:

- *Feature detector*: Maximally Stable Extremal Regions (MSER), as in [Mata02]
- *Measurement areas*: The radius of a cirle with an area of the number of pixels of an MSER region establishes the scale, s, of each detected region. The feature descriptor method defines its measurement area from this scale
- *Feature descriptor*: The oriented Speeded Up Robust Features (SURF-128) descriptor, as in [BayH06]
- *Feature matching*: Image pair-wise brute-force descriptor matching with a threshold on the ratio between the nearest and the second nearest match

This gives an automatic wide-baseline method for establishing candidates for correspondence points between images, without any a priori knowledge about the image contents. It has been implemented for a pair of input images, where only 8-bit black and white image intensities (0-255) are considered.

6.1 Disjoint Unifiable Sets and Union-Find

One of the primary tools for implementing the Maximally Stable Extremal Regions (MSER) method is a disjoint unifiable sets data structure, which we shall go through here. It can also be found in e.g. [Corm90] in the section "Data Structures for Disjoint Sets".

The data structure works by representing an element of a set by the value of the element, along with an optional reference to a parent element of the same set. With this parent reference, the elements form a tree, where the leaves point towards the root. The sets are disjoint, meaning that, an element belongs to precisely one set, initially its singleton set. A set is represented by its representative element, which is the element in the set without any parent reference, i.e. the root of the tree. Given any element, it is possible to find the representative element of its set, by traversing the parent references. The three most important operations on the data structure are: 1) creation of a singleton set, 2) unification of two sets and 3) querying whether two sets are the same set or two disjoint sets. When unifying two disjoint sets, one set is annihilated, by making the reference of its representative element point to an element in the other set.

The data structure with references between elements and how it changes by two unify operations is illustrated in figure 3, where the parts shown as `area =` $[a_0, a_1, a_2, \ldots]$ are only relevant for the Maximally Stable Extremal Region (MSER) detector, so they can be disregarded for now.

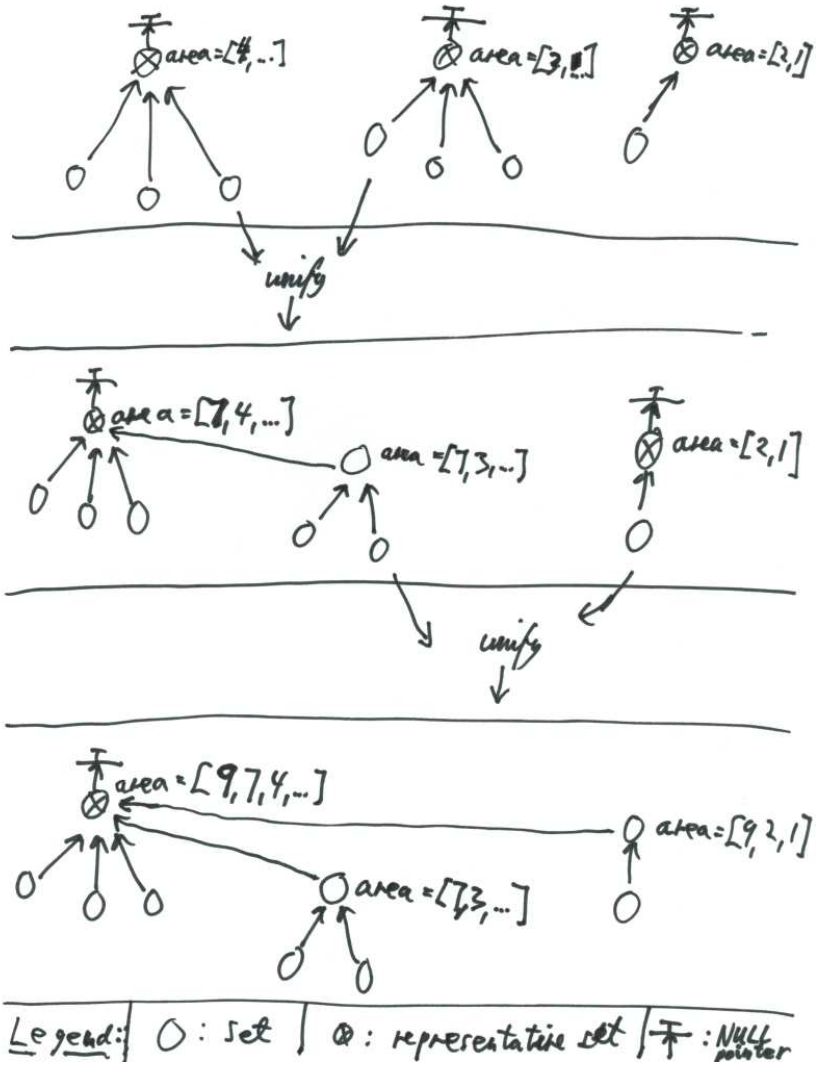

Figure 3: The disjoint unifiable sets data structure with references between elements. The horizontal lines split between the data structure as it looks at three different points in time, illustrating how it changes with two unify operations. The parts shown as area = $[a_0, a_1, a_2, \ldots]$ are only relevant for the MSER method and they illustrate maintained pixel areas of the sets for this method

Figure 4: An example of an upper level-set and a lower level-set, where a pixel's neighbourhood is its 4-neighbourhood. The squares represent pixels, whose intensities are given by the numbers

6.2 Level-Sets

For the Maximally Stable Extremal Region (MSER) detector, we will need the concept of a level-set. Specifically, the kinds of level-sets we will be considering are intensity level-sets, where we let S be the set of possible intensities. We need two kinds of level-sets: upper level-sets and lower level-sets. An upper level-set is defined as a set of pixels, which have high intensities, i.e. bright colours, down to some given lower limit on the intensity level. Formally, this can be defined as:

$$\mathcal{Q}^{\lambda} \quad = \quad \{I(p) \in S \mid I(p) \geq \lambda\}$$

Similarly, a lower level-set contains low intensities, i.e. dark colours, up to a given upper limit on the intensitiy level. More formally:

$$\mathcal{Q}_{\mu} \quad = \quad \{I(p) \in S \mid I(p) \leq \mu\}$$

The sets of pixels considered, which have the coordinates p in the above definitions, are usually connected neighbours of pixels. The neighbouring relation between pixels may vary. E.g. a pixel's neighbours may be its immediate four neighbours, its eight closest neighbours or something different. Such a pixel level-set can be specified by the following data: 1) a pixel belonging to the level-set, 2) whether it is an upper or lower level-set and 3) the intensity level, which defines the upper or lower limit of the level set. An example of an upper level-set and a lower level-set is illustrated in figure 4, where a pixel's neighbourhood is its 4-neighbourhood.

In [Mata02], they call the level-sets "extremal regions" and they denote them as $\mathcal{Q}_i$, where $i \in S$ and where it is unspecified whether it is an upper level-set or a lower level-set. We shall also use the subscript i, when upper vs. lower limit is unspecified.

As a litterature note, the references [Oshe06] and the preprint [Case08] use the definitions of level-sets seen here. However, [Mona00] denotes upper level-sets by X_{λ} and lower level-sets by X^{μ}, but otherwise defines an upper versus lower level set as here, i.e. $X_{\lambda} \equiv \mathcal{Q}^{\lambda}$ and $X^{\mu} \equiv \mathcal{Q}_{\mu}$. [Mona99], on

the other hand, uses X^{μ} to denote upper level-sets and define them oppositely, as having an upper limit and containing low intensity values. They have a corresponding definition of X_{λ} for lower level-sets. [Mata02] does not relate the definition of regions to level-sets.

6.3 Maximally Stable Extremal Region Detector: MSER

The implementation of this method follows the description in [Mata02], except for some elaboration on the criteria for when a region is stable, as we shall see.

6.3.1 Maintaining and Unifying Pixel Level-Sets

As metioned in section 6.1, the primary tool, used for implementing this method, is the disjoint unifiable sets data structure. The sets, which are maintained, are intensity level-sets of neighbouring pixels, where the neighbours of a pixel are its four immediate neighbours. The algorithm works by making two separate passes, one for upper level-sets and one for lower level-sets, so in each pass, we don't need to specify for each set, whether it is one or the other kind of level-set.

The data, which is maintained for each set, i.e. for the representative element of that set, is the following:

- The current area of the set, in pixels
- The current intensity level limit of the set
- A history of entires of how the set was before each unify operation. Each entry in this history contains the area in pixels and the intensity level limit

The intensity level limits mentioned here will be the lower intensity limit, for upper level-sets, and the upper intensity limit, for lower level-sets. New history entries are only recorded for a set, when its current intensity level differs from the new intensity level, such that we only record area changes as a function of level changes.

An example of the data stored for a set might be the following, where the set is an upper level-set, which has been unified at least two times:

```
{area    = 7,
 level   = 240,
 history = [{area = 5, level = 246},
            {area = 1, level = 251}]
}
```

This is shown using Standard ML value expression syntax, where records with named fields are surrounded by { and } and lists of elements are surrounded by [and]. List elements, as well as record fields, are separated by commas and a record field has the form <field name> = <value expression>.

When two sets are merged, one set is annihilated by the merge, as mentioned earler, but its former representative element keeps its data and ceases to be a representative element. The data of this former representative has its current area and level updated to that of the new set, while maintaining the history of former areas and levels; this is exactly the same as for representative elements. Former representative elements, which means most elements after this algorithm has run, thus have a history of area changes as a function of level changes, which includes the area and level of the unified set, which made the element cease to be a representative element. This in turn means that, the history of the life-times of

all sets created by unification is maintained. In figure 3, the parts shown as `area = [`$a_0, a_1, a_2, \ldots$`]` illustrate the maintained areas of the sets, where a_0 is the current area for representative elements; for former representative elements, a_0 is the area of the unified set, which annihilated the set formerly represented by that element.

6.3.2 The Main Algorithm

Knowing the machinery of how sets are unified and their data maintained, we can now go through the main algorithm.

We start by sorting all pixels in the image according to their intensity. This can be done particularly efficiently, by using Bucket-Sort, also known as Bin-Sort, described in section 9.4 in [Corm90], if pixel intensities are in a small discrete set of values, such as integers from 0 to 255 for an 8-bit image.

For generating the lower level-sets, the intensities are traversed from the lowest intensity level to the highest, where the intensities are considered the upper level limit. For each intensity level for each pixel at that level, a singleton level-set for the pixel is created and unified with the possibly existing level-sets for the pixel's four neighbours. For doing this, it is convenient to have a two-dimensional array, whose domain is the same as that of the frame buffer coordinates, such that a pixel level-set element can be associated with each pixel. Each entry of such an array is initially empty. Entries get a pixel level-set element associated with it, when the corresponding pixel is traversed, during the ordered intensity traversal. After this traversal, each entry of the array contains a history of pixel level-set area changes as a function of intensity level limit, for sets which have had its representative element at that pixel. In the next section, will use this history for extracting level-sets, which are considered stable.

Upper level-sets are handled similarly, except that the intensities are traversed from the highest intensity level to the lowest and that the intensities are considered lower level-set limits.

6.3.3 Extracting Stable Regions

This section contains quite a lot of description and details, but this is significant for the final results.

[Mata02] uses the following definition of a stable region: "Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_{i-1}, \mathcal{Q}_i, \ldots$ be a sequence of nested extremal regions, i.e. $\mathcal{Q}_i \subset \mathcal{Q}_{i+1}$. Extremal region $\mathcal{Q}_{i^*}$ is maximally stable if and only if $q(i) = |\mathcal{Q}_{i+\Delta} \backslash \mathcal{Q}_{i-\Delta}|/|\mathcal{Q}_i|$ has a local minimum at i^* ($|\cdot|$ denotes cardinality). $\Delta \in S$ is a parameter of the method.". The set S here is the set of image intensities and it should be noted that, this notation assumes that the ordering of the indices i is reversed, when considering upper level-sets, as opposed to lower level-sets. The formula

$$q(i) = \frac{|\mathcal{Q}_{i+\Delta} \backslash \mathcal{Q}_{i-\Delta}|}{|\mathcal{Q}_i|} \tag{9}$$

means that, we look at the histories of level-sets and consider a level-set $\mathcal{Q}_i$ at a point in time, where its level limit is i. Its area in pixels is $|\mathcal{Q}_i|$. The backslash denotes set-exclusion, so $|\mathcal{Q}_{i+\Delta} \backslash \mathcal{Q}_{i-\Delta}|$ is the number of pixels that have been added between intensity levels $i - \Delta$ and $i + \Delta$. So the formula finds minima in the relative rate of area change over a range of intensities.

The article [Mata02] does actually not say anything about what the parameter Δ should be or how to establish it. To get some intuition about this parameter, figure 5 introduces two photographs of the $A\kappa\rho\acute{o}\pi o\lambda\eta$ in Athens, which we will use for illustration. These pictures are images 6 and 7 in a sequence, taken in the year 2002 by the author, so the pictures were taken without prior knowledge of how the implemented methods presented in this report would be, but the author had read [Poll00] at

that time. Figure 6 illustrates extracted regions with Δ values of 50 and 10. Figure 7 is a zoomed-in version of the same pictures. As can be seen, the Δ value has a substantial impact on how many regions are detected.

Δ specifies the extent of a range of intensities and the minima are where there is the least amount of region area change over an intensity range of that extent, as compared to neighbouring intensity ranges. However, if the detected minima have a high value, it means that, there is *still a significant change in region size* at that minimum. Hence, lower minima should clearly be preferable to higher minima, so we can use the detected minimum values as a measure of confidence for how good the detected regions are. For a region to remain as stable as possible, it is also preferable that Δ is as large as possible. However, Δ is fixed in the formula in equation 9, so this formula does not easily allow for optimizing both the extent of the range of intensities and the minimal area change of regions at the same time. For now, we can consider figure 8, which can be seen as three example graphs of the formula in equation 9. High and low minima are pointed out and a horizontal line is drawn, which could be set as a limit between good and bad minima. Notice though that, in reality, this graph would not be smoothly varying, but piece-wise linear.

Another problem, unrelated to the above considerations, is that, there may be multiple minima detected close together on a history of region area changes. As it turns out, this problem actually seems to be quite serious in practice, since it detects several almost identical regions on top of each other. This can confuse the feature matching later on, which is very undesirable. There are examples of such multiple minima close together in figure 9 and we shall return to this figure later.

To summarize a few conclusions about the stability of the areas of regions, we have that:

- *Lower* minima of the rate of area change are better than *higher* minima
- Minima, which are sustained over a *longer range of intensities* are better
- *Oscillations* in the rate of area change can give *several minima close together*. This results in multiple almost identical regions on top of each other, which we would prefer to avoid

One solution to the problem with oscillations and several minima close together is to use a Hysteresis limit, in the way that it is used for e.g. battery charging or thermostats; see e.g. [HystWi]. Specifically, traverse the graph left to right (or right to left) while remembering a state flag, which denotes either "inside" a minimum or "outside" a minimum. The limit for switching from *inside to outside* is set *higher* than the limit for switching the opposite way. This actually changes the algorithm from detecting minima into detecting ranges of intensities, within which the rate of area change is acceptably low. An illustration of this can be seen in figure 9. In running the algorithm this way, we not only get the points of the minima, but also the range of how many intensity levels, through which the minimum was sustained. It should be noted that, the start of the minimum is detected at the low Hysteresis limit, while the end of the miminum is detected at the high Hysteresis limit. Thus, the detection of minima will become more symmetric, if we remember the last time that we passed the lower Hysteresis limit, before also passing the high limit to leave the minimum. The centre of the minimum can then be computed as the intensity in between the intensities, where the low Hysteresis limit was passed. This centre is then the intensity level, where the region is considered most stable. The location of the beginning, centre and end of a minimum is shown in figure 9. An example of regions extracted with this method, where Δ is 20, can be seen in the top image in figure 10. Figure 11 contains a zoomed-in version of the same picture.

An alternative solution to the problem with multiple minima detected close together, is to just detect all possible minima initially. After this detection, sequences of minima, which are close together, can be merged such that, only one of those minima is returned. For this, we would need a notion of when

Figure 5: Two pictures from a sequence of photographs of the $A\kappa\rho\acute{o}\pi o\lambda\eta$ in Athens. These pictures were taken in the year 2002 by the author, so the pictures were taken without prior knowledge of how the implemented methods presented in this report would be. These images are fairly challenging for the methods implemented here, since they contain lots of trees and bushes and lots of repeated texture on the brick walls. In this report, we will only be looking that those two pictures from this photographed image sequence. Top: Image six in the sequence. Bottom: Image seven in the sequence

Figure 6: Image six of the *Ακρόπολη* image sequence, where all detected MSER regions are shown by painting the edge pixels outside each detected region white. The computed centre of each region is shown as a white pixel with four black pixels around it. In this picture, regions are extracted with the formula in equation 9, where Δ is 50 in the top image and 10 in the bottom image. As can be seen, the Δ value has a substantial impact on how many regions are extracted. E.g. in the top image, the front side of the temple appears to be one big region

Figure 7: These pictures are zoomed-in versions of the pictures in figure 6, illustrating Δ values of 50 (top) and 10 (bottom) for the region stability criterion from the formula in equation 9. As can be seen, the Δ value has a substantial impact on how many regions are extracted. E.g. in the top image, the front side of the temple appears to be one big region

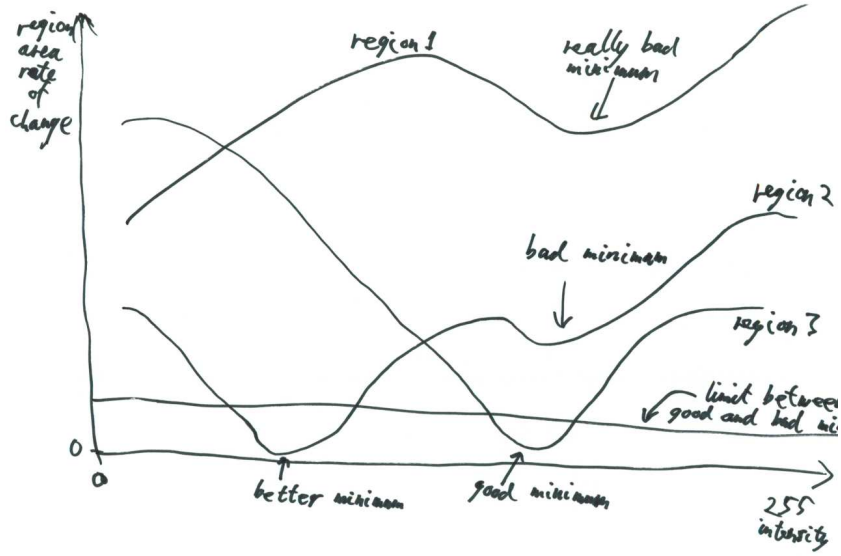

Figure 8: Example graphs of the formula in equation 9. The graphs can also be considered the rate of area change of regions as a function of the region's intensity level limit, when the region is formed as a level-set, as expressed by the formula in equation 11. The minima of the rate of area change are illustrated and low minima are better than high minima. A horizontal line is shown, which could be used for separating between good and bad minima

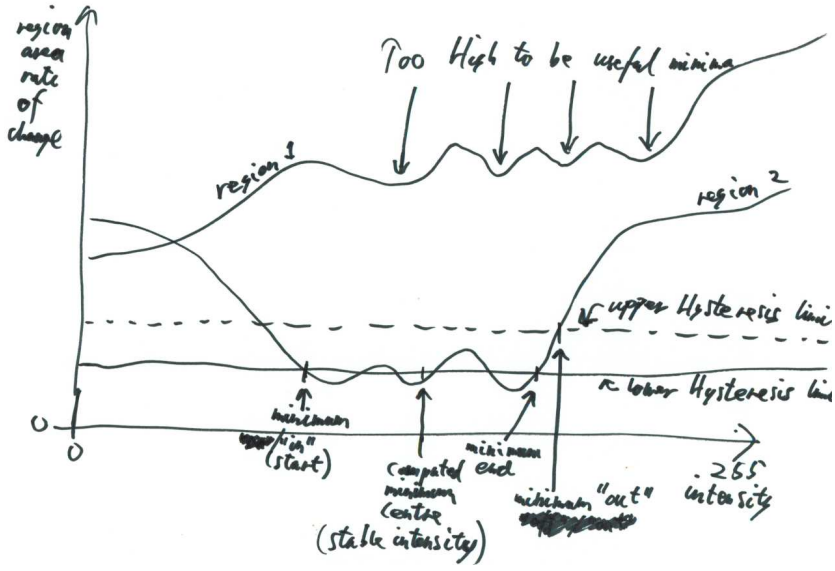

Figure 9: Two graphs of the formula in equation 9. The graphs can also be considered the rate of area change as a function of a region's intensity level limit, when the region is formed as a level-set, as expressed by the formula in equation 11. Minima are detected with a Hysteresis limit, which consists of a lower and an upper limit. These limits switch a flag, maintained while traversing the graph left to right (or right to left). The flag denotes when the rate of area change is "inside" a minimum versus "outside" a minimum. This minimizes oscillation artifacts, when using the limits to find a range of intensities, with acceptably low rates of area change

minima are close together. One definition is that, if the area increase between two successive minima $q(i^*)$ and $q(j^*)$, is sufficiently small, they are considered as being close together. One possible formula for this is the following, where we assume that $|Q_{j^*}| \geq |Q_{i^*}|$ (exchange j^* and i^* if this is not the case):

$$\frac{100\ (|Q_{j^*}| - |Q_{i^*}|)}{|Q_{i^*}|} \quad < \quad p \tag{10}$$

In this formula, p is the maximum allowed percentage-wise increase in area from $|Q_{i^*}|$ to $|Q_{j^*}|$.

The ideas presented so far are the primary ideas for the implementation that we will be evaluating. However, we will also go through a different algorithm for detecting stable regions, which combines some of the ideas presented so far in a different way. Instead of using the formula in equation 9, we can consider this alternative formula:

$$q(i) \quad = \quad \frac{|\mathcal{Q}_{i+1} \backslash \mathcal{Q}_i|}{(|\mathcal{Q}_i| + |\mathcal{Q}_{i+1}|)/2} \tag{11}$$

This formula expresses the rate of area change as a function of the region's intensity level limit. It somehow corresponds to the formula in equation 9, with Δ being 0.5, but $\Delta = 0.5$ is not strictly valid for that formula. We can use this formula as before, i.e. either 1) for detecting minima and possibly merging minima, which are close together, or 2) by using Hysteresis limits to derive ranges of intensities, where the regions are stable. Notice that, since there is no a priori fixed range of intensities in this formula (equation 11), it should more readily accomodate using the extent of the detected ranges of intensities as measures of confidence, as opposed to the formula in equation 9. An example of regions extracted with this method can be seen in the bottom image in figure 10. Figure 11 contains a zoomed-in version of the same picture. In this picture, only regions, for which the rate of area change was below the lower Hysteresis limit over *at least two intensity levels* were finally kept, which is the way that the extents of ranges of intensities were used as confidence measures here.

Until now, we have talked about using either the detected minimum values or the extents of detected ranges of intensities as measures of confidence. One possible way of doing this is that, the best half of the regions could be selected, e.g. those with the best measures of confidence. Alternatively, a fixed limit could be set, for how good a measure of confidence has to be, in order to be classified as acceptable. A fixed limit on the minimum value, which is one kind of measure of confidence, is shown as a horizontal line in figure 8. Such a limit could be fixed a priori or it could be set, based on an analysis of the actual measures of confidence of all detected region histories. One simple analysis is making simple statistics on the detected regions, e.g. finding the smallest, the largest and the mean values of the measures of confidence. A limit could be set, based on those statistical values. When using the formula in equation 9 for detecting stable regions, we use this technique by afterwards setting a limit, which is half-way between the smallest minimum value and the mean of all minimum values. We use this limit for removing all detected regions, whose minimum values are above the limit. For short, this method will be called *half-mean filtering*.

As an example of half-mean filtering, we will consider image nine in the sequence of images of Valbonne Church, since an image from this image sequence is also shown in the article [Mata02], which allows for a fairly direct visual comparison with that article. Even though the image in that article is not the same image from that image sequence, the detected regions in [Mata02] still seem somehow better than what we get here. Figure 12 has regions detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$. Figure 13 is a zoomed-in version of the same picture. For comparison, figure 14 shows regions detected in the same way, but

Figure 10: Image six of the *Ακρόπολη* image sequence, where all detected MSER regions are shown by painting the edge pixels outside each detected region white. The computed centre of each region is shown as a white pixel with four black pixels around it. Top: Regions are extracted with Hysteresis limits on the formula in equation 9, where Δ is 20. The Hysteresis limits were computed based on statistical analysis of the detected minimum values of an initial region detection (without Hysteresis), as described in the text. Bottom: Regions are extracted with Hysteresis limits on the formula in equation 11. The low Hysteresis limit is 0, which is tested by $\leq$ and the high limit is 1, which is tested by $>$. Only those regions, for which the rate of area change was below the lower Hysteresis limit over *at least two intensity levels* were finally kept

Figure 11: These pictures are zoomed-in versions of the pictures in figure 10. Top: Regions are extracted with Hysteresis limits on the formula in equation 9, where Δ is 20. The Hysteresis limits were computed based on statistical analysis of the detected minimum values of an initial region detection (without Hysteresis), as described in the text. Bottom: Regions are extracted with Hysteresis limits on the formula in equation 11. The low Hysteresis limit is 0, which is tested by $\leq$ and the high limit is 1, which is tested by $>$. Only those regions, for which the rate of area change was below the lower Hysteresis limit over *at least two intensity levels* were finally kept

with regions subsequently removed according to the statistical limit just described: half-mean filtering. Figure 15 is a zoomed-in version of the same picture. As can be seen, this removes some regions, which do not seem very well-defined. In practice, half-mean filtering seems to improve the overall matching performance.

We can also use the same kind of statistical analysis for establishing the Hysteresis limits, if using those methods. In the top image in figure 10, and its zoomed-in version in figure 11, the Hysteresis limits were computed based on such a statistical analysis. The statistics was computed on the detected minimum values of an initial region detection without Hysteresis. The high Hysteresis limit was the largest detected minimum value and the low Hysteresis limit was obtained by linear interpolation between the largest detected minimum and the mean value of all detected minima, where the interpolation weight is 0.99, meaning close to the high Hysteresis limit. When Δ is 20, as in this case, the detected minima have quite high values, up to around $10,000$, which is the reason for the fairly suspiscious looking weight of 0.99.

As a final suggestion for a very simple method of extracting regions from the histories of area change, the following was used in the initial stages of this project: accept all regions, which do not change area over two or more intensity levels in the histories of area change. This works fairly well, but intuitively, it seems to depend on regions not having too blurred or soft edges.

A few conclusions about the methods proposed in this section will be given here. However, they are based only on a few experiments, which were not particularly systematic, and not all the implemented methods are thoroughly tested, so these are not scientifically verified results:

- When using the formula from equation 9, a Δ value of 20 seems to work fairly well for finding corresponding image features. This formula and $\Delta = 20$ is what we will use in the experiments, unless otherwise stated. A large Δ value, like 50, is good for segmenting the image into a few regions, but not very good for detecting several regions usable for finding corresponding image features. Values smaller than 20 seem useful, but they can result in many detected regions and the *percentage of correct matches* may even decrease. Figures 6 and 7 are relevant here

- Merging regions detected in sequence on the same history of area changes seems to improve the overall matching performance. At least it seems to be the case, when the merged sequence of regions successively grow by less than 10 percent of their size, i.e. when using the formula in equation 10 with $p = 10$. An important performance measure here is that of *1-precision*, which we shall use for the evaluation in section 8. We will use this merging from equation 10 with $p = 10$ in the experiments, unless otherwise stated

- The methods using Hysteresis limits, whether it is based on the formula in equation 9 or the one in equation 11, seem to be good at detecting only a few regions. However, just as for the case of large Δ values in the formula from equation 9, this does not seem particularly well-suited for finding corresponding image features. Figures 10 and 11 are relevant here

- Using the value of the detected minima seems to be useful as a measure of confidence for the detected regions. In particular, "half-mean filtering" seems to improve the overall matching performance. This is done by setting a limit, which is half-way between the smallest minimum value and the mean of all minimum values, and removing all detected regions, whose minimum values are above this limit. This will be done in all experiments, unless otherwise stated. Figures 12, 13, 14 and 15 are relevant here

- The simple method of just accepting all regions, which do not change area over two or more intensity levels in the histories of area change, is not to be underestimated, but it is not particularly

Figure 12: Image nine of the Valbonne Church image set, from which an image is also shown in the article [Mata02]. All detected MSER regions are illustrated by painting the edge pixels outside each detected region white. The computed centre of each region is shown as a white pixel with four black pixels around it. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$

Figure 13: This is a zoomed-in version of the image in figure 12. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$

Figure 14: Image nine of the Valbonne Church image set, where all detected MSER regions are illustrated by painting the edge pixels outside each detected region white. The computed centre of each region is shown as a white pixel with four black pixels around it. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$. The method that we refer to as half-mean filtering, is also applied to the regions in this image. Notice that, compared to the image in figure 12, some regions are removed by the half-mean filtering. In particular, many regions are removed around the cars and the fences in the scene and the walls towards the left of the picture, which did not seem like very clearly defined regions

Figure 15: This is a zoomed-in version of the image in figure 14. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$. The method that we refer to as half-mean filtering, is also applied to the regions in this image. Notice that, compared to the image in figure 13, the regions which have been removed by the half-mean filtering are less clearly defined than those which remain in this picture

impressive either. It detects a very large amount regions though, often between $5,000$ and $20,000$ regions, for the images shown in this report

- All of the above considerations apply, when handling *upper level-sets separately from lower level-sets*, as we do consistently in this report
- In *all implementations and experiments presented in this report*, regions with an *area of* 16 *pixels or less* or an *area of one fourth of the total image area or more* are removed. This is done after all other detection methods and tricks have been applied. For some of the statistical analyses, it might be worthwhile to investigate what the impact would be, if these very small and very large regions were removed before computing the statistics

As a final example of which regions the final combined region detector can detect, consider figure 16. Figure 17 is a zoomed-in version of the same picture. This shows what the detector finds, when one picture (the bottom picture) is more blurred than the other (the top picture). Blur is one of the hard challenges for many region detection methods. Yet, the implemented MSER method finds regions, which seem fairly consistent between the two pictures.

According to Søren Ingvor Olsen, there has been other work by the authors of the article [Mata02], which address some of the issues discussed in this section, but that other work has not been looked into for this report.

6.4 Summed Area Tables

The Speeded Up Robust Features (SURF) method uses a utility method called Summed-Area Tables, introduced back in [Crow84] and well-known in folklore (see p. 114-115 [Moll99], p. 145-146 [Watt92] or p. 827 [Fole96], depending on what's on the nearest book shelf). They call it Integral Images in [BayH06] and we will go through it here.

An integral image I_Σ is constructed from the original input image I. Let both images be parameterized over a coordinate pair (x, y). I_Σ consists of, at each location (x, y), the sum of pixel values in the image $I(i, j)$ which have image coordinates $i \leq x$ and $j \leq y$. I.e. $I_\Sigma(x, y) = \Sigma_{i=0}^{i \leq x} \Sigma_{j=0}^{j \leq y} I(i, j)$. The area of an upright rectangular area can now be computed by four look-ups in I_Σ, one addition and two subtractions, assuming that we have checked that we are inside the image boundary. This is illustrated in the left image in figure 18.

6.5 The Measurement Areas

When the features have been detected, it is necessary to establish a measurement area for each feature. This is needed by the feature descriptor, described below in section 6.6.

The specific feature descriptor that we use needs a feature scale, s, for determining its measurement area. This scale can roughly be thought of as the radius of a circle approximating the detected feature. From this scale s, the descriptor method looks at a circular neighbourhood of radius $6s$, weighted with a Guassian function with $\sigma = 2.5s$, for determining an orientation of the measurement area. The actual measurement area for the descriptor is then defined as a an oriented box of size $20s$, weighted with a Gaussian function with $\sigma = 3.3s$. Both of the Gaussians here are centered at the centre of the detected feature.

The scale s for the detected MSER regions is established as the radius of a cirle with an area A equal to the number of pixels of the detected region. Thus, s is given by the High School formula for the radius of a cirle, given its area A:

Figure 16: Images one (top) and three (bottom) from the bikes image set, which we will see again in for the performance evaluation in section 8. All detected MSER regions are shown by painting the edge pixels outside each detected region white. The computed centre of each region is shown as a white pixel with four black pixels around it. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$. The method that we refer to as half-mean filtering, is also applied to the regions in this image. The bottom image is more blurred than the top image and blur is one of the difficult challenges for many feature detection methods. The MSER method seems to detect regions, which are useful for fe

Figure 17: Images one (top) and three (bottom) from the bikes image set, which we will see again in for the performance evaluation in section 8. These pictures are zoomed-in versions of the pictures in figure 16. The regions are detected with the formula in equation 9 with $\Delta = 20$ and with sequences of regions merged according to equation 10 with $p = 10$. The method that we refer to as half-mean filtering, is also applied to the regions in this image. The bottom image is more blurred than the top image and blur is one of the difficult challenges for many feature detection methods. Notice in particular that, the regions of the black windows on the white doors, the sign on the brick wall, the license plates, side mirrors, back lights and side packs of the motor bikes are detected in both images

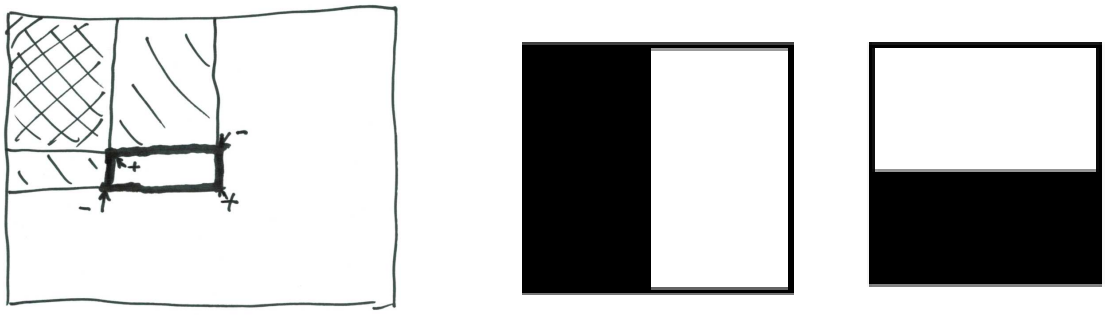

Figure 18: Left: The dark square illustrates an area of summed pixels. The sum is calculated by using the four summed rectangles shown, starting from the upper-right corner of the image and extending down to their lower-right corner. The rectangles, which have a plus sign at their lower-right corner, are added to the final sum and those, which have a minus sign, are subtracted. Right: The Haar wavelet filters in the x and y directions, respectively. The white areas have filter value 1 and the black areas have filter value -1

$$s = \sqrt{\frac{A}{\pi}}$$

6.6 Speeded Up Robust Feature Descriptor: SURF-128

The implementation of this method follows the description in section 4 of [BayH06]. As described in the previous section, this method includes two steps: 1) calculating what the measurement area is, specifically its orientation, and 2) calculating a descriptor within that measurement area. The method relies on having determined a feature scale, s, as described in the previous section.

6.6.1 Calculating the Orientation

This part of the method gives a reproducible orientation for the measurement area of the descriptor. The primary tool for this is making Haar-wavelet filter responses in the x and y direction. These filters are illustrated in the right image in figure 18 and are computed efficiently by the summed-area tables described earlier. Notice that, the y direction filter is upside-down, as compared to [BayH06], since we want the upwards response, in keeping with the standard mathematical coordinate system conventions, despite that the y-axis of our image coordinate system points downwards in the implementation. In the implementation, we are also not mirroring the filters, as defined for the convolution operation, but we should get "rightwards" and "upwards" gradient-like responses in this way.

The first step in determining the orientation, is to make regularly spaced Haar-wavelet filter samples, with the distance between samples being s, the determined scale of the detected feature. Each sample is made with Haar-wavelet filters with side length $4s$. We make these samples in a neighbourhood of radius $6s$ around the centre of the detected feature and weight the samples with a Gaussian filter with $\sigma = 2.5s$. In the implementation, we thus traverse a square area of size $12s$, where the Gaussian-weighted sample pattern is illustrated in the left image in figure 19. Notice though that, in the article [BayH06], it is

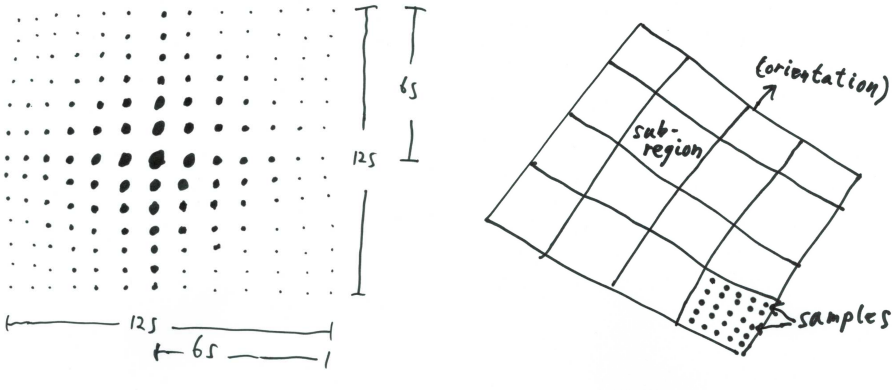

Figure 19: Left: The sample pattern for determining an invariant rotation for the SURF descriptor. The dots illustrate sample points, where Haar-wavelet samples are made and where the size of the dots illustrate the Gaussian weight of that sample. Right: The sample pattern of the SURF descriptor. The dots illustrate the 25 sample points, which are used for creating 8 different sums within each sub-area

stated that, they do this sampling in a circular region, which would indeed be possible, thus saving a few samples near the corners of the square region, but it has currently not been implemented that way.

We collect the samples in a list of pair-wise samples, each pair with one x and y Haar-wavelet response. We treat these pair-wise responses as vectors, with given x and y coordinates. From these vectors, we find a dominant orientation. This is found by considering a sliding orientation window of angle $\frac{\pi}{3}$ radians, containing the vectors with angles within this window. Within each possible such window, the vectors are summed. The longest of these summed vectors is chosen as the dominant orientation. This orientation is specified as an angle in radians, according to mathematical standards, and is the orientation of the measurement area for the descriptor. The window size $\frac{\pi}{3}$ is a parameter, which is experimentally determined to be a good choice in [BayH06], so we will use this constant.

6.6.2 Calculating the Feature Descriptor Vector

After we have assigned an orientation in the previous step at the given feature scale s, we can now define the measurement area as an oriented box, rotated with the angle from the previous step. The size of this box is $20s$. All samples that we make within this box are weighted by a Gaussian with $\sigma = 3.3$. Both the box and the Gaussian filter are centered at the centre of the detected feature.

The samples we make in this box are done in 4x4 regularly divided square sub-regions of the box. Within each sub-region, we make 5x5 regularly spaced samples, weighed by the aforementioned Gaussian filter. The sample pattern and sub-regions are illustrated in the right image in figure 19. At each sample point, we compute Haar-wavelet filter responses in the x and y directions, similar to the previous step. However in this case, the size of the filter is $2s$ and the filter is oriented as the box. This means that, we cannot use the summed-area tables here. Implementation details, including source code, for making such a Haar-wavelet filter sample, are given in the appendix.

The samples from each of the 4x4 sub-regions are summed in various ways. Specifically, we calculate eight sum-values for each sub-region, which gives a total of 128 feature descriptor vector values. Let us denote the Haar-wavelet responses by d_x and d_y for the x and y directions, respectively, and we assume that they are pair-wise related. Then we calculate the following sums: $\Sigma_{d_y<0} d_x$, $\Sigma_{d_y \geq 0} d_x$,

$\Sigma_{d_x>0}d_y$, $\Sigma_{d_x\leq 0}d_y$, $\Sigma_{d_y<0}|d_x|$, $\Sigma_{d_y\geq 0}|d_x|$, $\Sigma_{d_x>0}|d_y|$ and $\Sigma_{d_x\leq 0}|d_y|$. This can be described as the eight possible combinations of sums of d_x, d_y, $|d_x|$ and $|d_y|$, where the d_x values are separated into two sums, depending on the sign of its related d_y and vice versa for d_y. In [BayH06], section 4.2 and figure 3 are good for giving an intuition about the meanings of these values, but this will not be repeated here.

The final 128-dimensional feature vector is normalized, to get contrast invariance for the descriptor. Intensity offset invariance is already achieved by the fact that, the Haar-wavelet filters only measure image intensity gradients, not actual intensities.

The article [BayH06] mentions that, they partition the features into two groups, depending on the sign of the Laplacian. This is analogous to separating the detected MSER features into upper and lower level-sets, since this also, like the Laplacian sign, distinguishes between black-on-white versus white-on-black features. We therefore already have this advantage.

6.7 Feature Matching

This section describes the pair-wise matching of features from one image with features in the other image. In the following, we will refer to one image as "the first image" and the other image as "the second image", but the roles of which is which are sometimes exchanged, as stated in the relevant places.

The matching distance function is the Euclidean distance on the 128-dimensional feature vector. The matching is done with the criterion that, a feature in the first image matches a feature in the second image, if the distance to the closest, in terms of the Euclidean distance, feature in the second image is closer than 0.7 times the second-closest feature in the second image. In that case, the closest feature in the second image is chosen as a match for the considered feature in the first image. This is as previously argued and as suggested in [BayH06].

The pair-wise feature matching is done by brute-force matching. Alas, brute-force matching is apparently not just brute-force matching nowadays. The first and simplest way of doing brute-force matching is to traverse all feature descriptors in the first image. Each of these traversed features in the first image is compared to all features in the second image, where a match is found, if the matching criterion is met. This strategy is illustrated in figure 20. One may notice that, the features in the second image may get multiple correspondence matches to the first image, but not vice versa. Therefore, this strategy is not symmetric. We call this strategy "one-way brute-force matching". It might be an appropriate strategy for tasks like object recognition, where an unknown test image is matched against one or more known images. This is also the method that we will use for the performance evaluations, since the performance measures don't work very well with two-way matching. However, for feature matching between two images for camera registration, it does not seem appropriate, so we extend the one-way matching strategy further.

The first extension we will consider, is to first run the above one-way matching from the first to the second image and then do the same with the roles of the first and the second image exchanged. We then merge the set of matches found, taking care to switch around the matched feature pairs in one of the sets, such that we have consistent roles of first and second images in the merged set of matches. The merged set of correspondence matches is likely to contain duplicate matches, so we remove those, since they are obviously redundant. We call this strategy "conservative brute-force matching".

A second alternative to extending the one-way matching, is to run the two one-way matching passes, as in the previous strategy. In the step of merging the two sets, we only keep the matches, which are present in both sets of matches. We call this strategy "cross-correlated brute-force matching".

A third alternative, is to first run the one-way matching pass once. Then a new pass is made, where only the matched feature vectors from the second image are kept, with duplicates removed. Now a

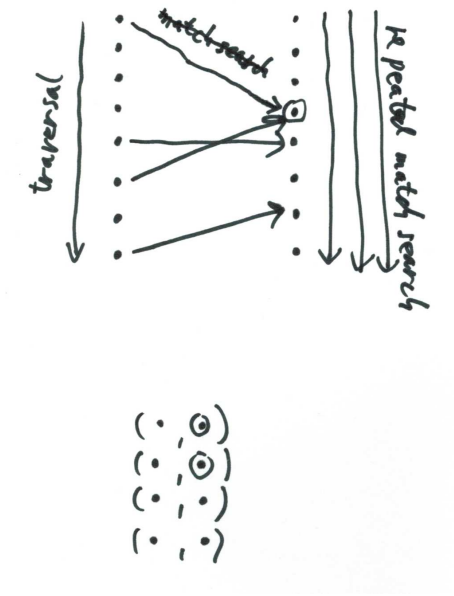

Figure 20: The feature descriptor list traversals, which are done for a brute-force one-way feature match pass. Notice that, feature descriptors in the list at the right may be present in multiple correspondence matches

second one-way matching pass is made, where the features kept from the second image are traversed, each one being compared to all feature vectors of the first image. In this matching result, matches, which have duplicate features from the first image, have their duplicates removed. This final duplicate removal ought to be done in a way, which favours the best matches, but that has not been implemented. This is the most aggressive strategy, so we call it "aggressive brute-force matching".

No thorough comparison between these methods has been made. However, source-code for automated simple unit-tests of the three extended strategies are given in Appendix D. For simplicity, the unit-tests just use integers as feature descriptors and the difference between integers as the distance measure between them. The reason that we can test the actual implementation in this way, with integers as descriptors, is that, the matching functions are polymorphic in which feature vector and distance measure are used, which is very convenient. From these unit-tests, it is evident that, the conservative brute-force matching keeps the largest amount of matches and that the aggressive brute-force matching keeps the fewest matches.

Early in this project, the aggressive brute-force matching and the cross-correlated brute-force matching seemed to filter out obviously good matches. However, this has not been verified for the final implementation. Therefore, the conservative brute-force matching strategy is the one we will use here, but this may not necessarily be ideal.

As mentioned earlier, for the performance evaluations of the detector and descriptor, the one-way matching will be used, since the performance measures will not work properly for the two-way matching methods, at least not for the conservative two-way matching.

Regarding execution time, it can be noted that, the aggressive brute-force matching is faster than the conservative and the cross-correlated strategies, since it filters out several features from consideration already in its first two passes. Doing just one-way matching is obviously even faster.

Figure 21 shows oriented descriptors on the $A\kappa\rho\acute{o}\pi o\lambda\eta$ images from earlier. The descriptors shown are those, which have been successfully matched by conservative two-way brute force matching. This is the only image in this report, which uses two-way matching. The matches between these two images do not seem particularly good, since only few matches seem to be correct. This already suggests that, the implementation has to be improved.

7 Suggested Implementation Improvements

This section outlines some ways, in which the implemented methods could be improved.

7.1 Improvements to the MSER Detector

As made evident by section 6.3, particularly section 6.3.3, there are many possible ways of fine tuning the parameters of the implemented MSER detector. A general observation is that, those methods, which are extensions of what is presented in the article [Mata02], are generally concerned with finding stable regions. In particular, finding low minima in the histories of area change, while also sustaining those minima over as many intensity levels as possible, is a general theme. It would be interesting to find a method, which can optimize for both of these criteria, while still avoiding multible detected regions on top of each other. Even more drastically, the MSER detector could be replaced by the Fast Level Set Transform (FLST), in case it turns out to be better.

The article [Mata02] suggests using a measurement area of size 3 times that of the convex hull of the detected region. This corresponds quite well to the Gaussian weight with $\sigma = 3.3s$ used by the SURF descriptor. Section 3 of [Mata02] also suggests selecting measurement areas at different scales, 1, 1.5, 2 and 3 times the scale s of the detected region, which could also be considered for the implementation presented here. It is not clear from their description how exactly this should be done, but descriptors on measurement areas of size $1s$ in one image could be matched only to descriptors on measurement areas of size $1s$ in other images; similarly for the other sizes. Considering that, the MSER method detects relatively few regions, this improvement is probably important to consider for this particular method, but it may also be used for other feature detection methods.

7.2 Incorporating Multiple Detectors

As already mentioned and suggested by previous work (e.g. [Tuyt00], [Lowe04] and [Miko06]), it is worthwhile to consider incorporating several feature detectors into the same framework, in order to get a robust system. In particular, a point-based detector, such as e.g. the Fast Hessian detector from [BayH06] or Hessian-Affine from [Miko06], would make sense to combine with an MSER detector, since these kinds of detectors find significantly different kinds of features. Both of these two kinds (region based and point-based) of methods have strengths and weaknesses, which, to some extent, complement each other. Part of the implementation from the SURF-128 descriptor, e.g. the summed area tables, could even be reused for implementing the Fast Hessian detector.

Features should be detected and matched independently for each detector method. In a first phase of the pipeline, some estimates of the camera view relationship could be made from the resulting feature correspondences. This could be done both individually from the correspondences obtained with each detector method, as well as by combinations of feature correspondences from multiple detector methods. Each of these estimates could be used in a second phase with a more narrow search for matching features. The estimate, which result in most matches being found in this second phase, could be assumed to be the most reliable estimate to work from. The phrase "most matches" here will probably have to be

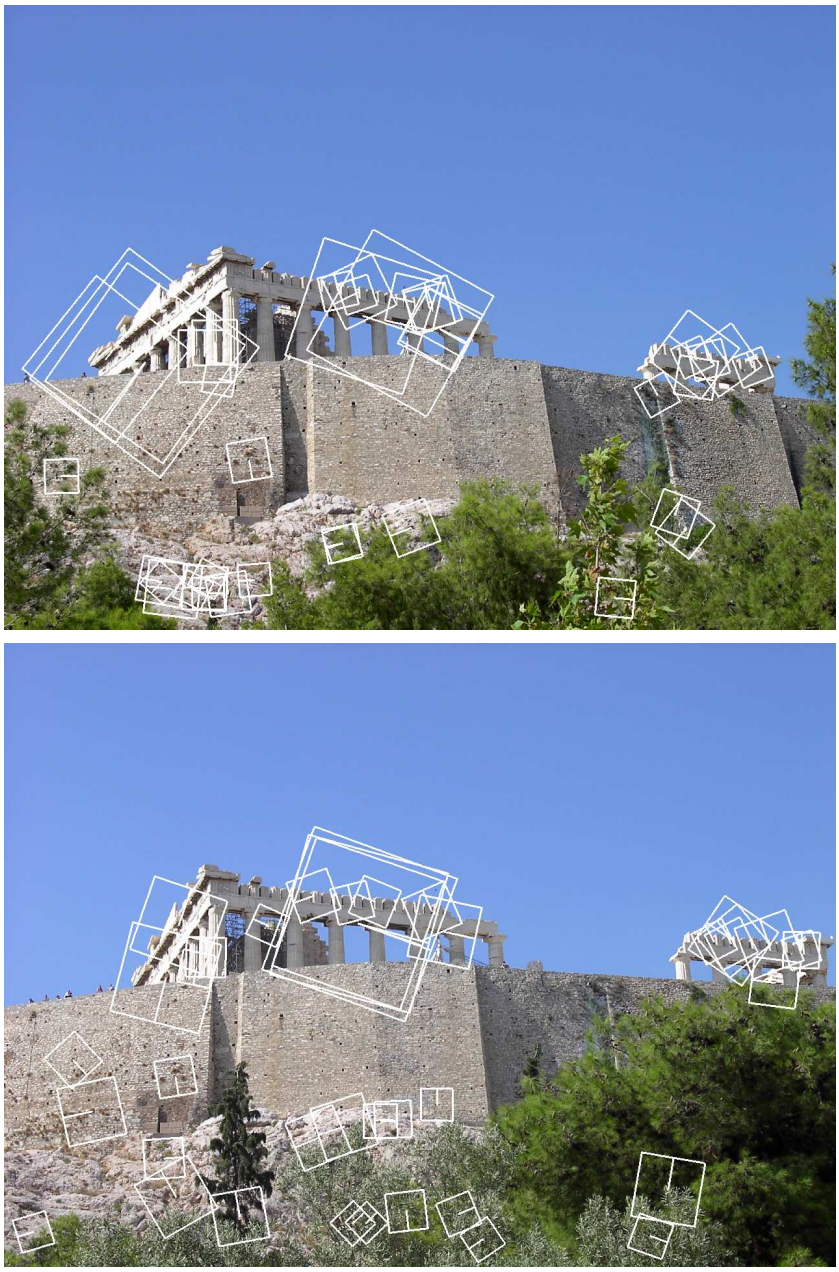

Figure 21: The two images of the *Ακρόπολη* from earlier, where all matched descriptors are shown as oriented boxes. This image uses the conservative two-way matching strategy. The matches between these two images do not seem particularly good, since only few matches seem to be correct

adapted to the individual feature detection methods, since some methods generally detect fewer features than others, where MSER detects relatively few features.

7.3 Improvements to the SURF Descriptor

As already mentioned, in the article [BayH06], it is stated that, they do the sampling in the orientation step in a circular region. This would indeed be a good and simple optimization, but it has currently not been implemented.

The article also describes that, the descriptor calculation on the oriented area is done in the box area and weighted with a Gaussian. Sampling in a square area also seems most natural, with the way that their samples are divided into sub regions. However, it seems that doing this sampling in a circular region instead, thus cutting away the corners of the box, would make sense here, especially given the presence of the Gaussian weight. This would give *fewer sub region samples*, thus yielding an optimization. Changing this would also make it possible to calculate feature descriptors *slightly closer to the edges of the input images*, thus including more features in the images. This improvement has currently not been implemented.

If sampling in a circular region is to be made, the sub regions could also be reorganized a bit, such as to only have the four central sub regions and one outer region in each of the four principal directions, as shown in figure 22. This would *reduce the dimensionality* of the descriptor, possibly without much loss of its descriptive power, since it maintains the dense sampling where the Gaussian weights are high. The samples in the proposed four outer regions could even be placed in a *more strategic sample pattern* than regular placement, which is also illustrated in figure 22. Notice though that, if the space between any of the samples is increased, the sample filter size should probably be increased correspondingly for those particular samples. Decreasing sampling density in this way might be a good idea near the proposed circular cut-off edge of the Gaussian filter, since it is furthest away from the centre of the detected feature, which is where e.g. affine and projective distortions are likely to be worst. Such distortions are also the reason for the Gaussian weight to be there in the first place.

7.4 Considering Affine Measurement Areas

In the article [Miko06], they suggest fitting an ellipse to the detected MSER regions and use this for achieving affine invariance. In fact, the ability to do this quite easily, thus acheiving affine invariance, is one of the strengths of the MSER method. It would also be worthwhile to consider this here for the measurement areas used for the SURF descriptor. However, it may require a bit of care, in the way that such an elliptical measurement area is utilized, when calculating the descriptor.

For the orientation step, it seems quite natural to warp the placement of the samples, as well as the shape of the weighting Gaussian, according to the determined ellipse shape. However, the calculated orientation of the box may not be compatible with the orientation of the ellipse, i.e. the two oriented axes of the box may not coincide with the orientation of the two axes of the ellipse. Thus, we cannot simply squish the two axes of the box according to the ratio of the two axes of the ellipse. However, we may warp the box and the positions of all the samples in the box, as well as the weighting Gaussian, according to the elliptical shape. This would turn the box into a parallelogram and the Gaussian weight would get an elliptic shape. Notice that, in doing this, the individual Haar wavelet samples of the descriptor should probably also be warped, thus also turning into parallelograms. Doing this for the orientation step would remove the optimization of using summed area tables, but since this step is only used for assigning an orientation, it might not be too bad to keep the square Haar wavelet samples in this step. Square samples are also used in the orientation step of the current implementation, which

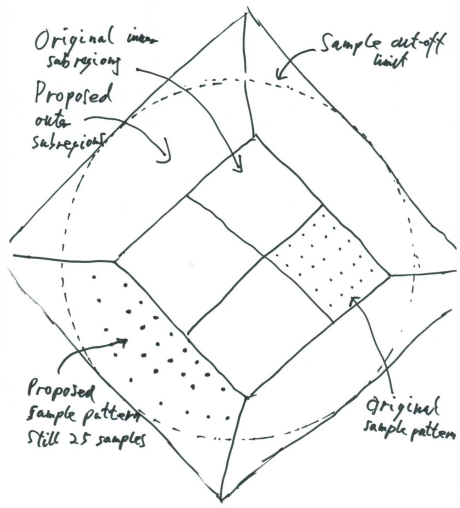

Figure 22: Proposed improvements to the SURF-128 descriptor: 1) sampling only in a circular region, 2) sampling in fewer sub regions and 3) sampling the outer regions with a more strategic sample pattern. This should give advantages such as being able to handle features closer to the edges of the input images, making affine invariant sampling more natural and getting fewer descriptor dimensions, hopefully without significant decrease in its descriptive power

seems to give fairly stable orientations.

The warping of sample locations according to an elliptic shape in the above two steps of the descriptor calculation seem to particularly advocate that the descriptor samples are only made in a circular region, rather than in a square, as suggested in the previous section. So, these proposed improvements are quite compatible with each other, but could also be made and evaluated individually.

The main consideration here though is, whether the affine invariance of the measurement areas are worthwhile. As mentioned, previous work states that, it may not be worthwhile and that it in fact sacrifices some matching accuracy ([Miko04] and [Lowe04]). However, as we shall see in the evaluation, the affine invariance seems to be needed. With the methods suggested here, we have both options, i.e. having affine invariance or not, where the performances could be compared by evaluation.

7.5 Faster Matching

In a real implementation, a clever data structure or a faster approximate heuristic for matching should probably be used, rather than brute-force matching. However, as mentioned earlier, the brute-force matching is good for evaluating detector and descriptor performances.

7.6 Image Intensity Cross-Correlation

It would be worthwhile to consider implementing normalized image intensity cross-correlation, as a way to reject unreliable feature correspondences, after they have been matched with the descriptor. This is also done in section 3 in [Mata02] and e.g. [Tuyt00] and [Miko04].

7.7 Considering Colours

The current implementation uses only black and white image intensities. One practical limitation, which may arise from this, is the case where neighbouring objects in a scene have different colours with similar luminance.

One simple way of extending the implemented methods to consider colours would be to run the feature detector for each of the three colour bands, red, green and blue, individually. Features detected in one colour band should only be matched to features detected in the same colour band in the other image, since it is reasonable to assume that, objects do not significantly change colour between varying view points. It would also be possible to do this extension on other colour spaces than the RGB colour space proposed here.

More elaborate methods of exploiting colour information may also be possible, but would most likely require some kind of extension to the actual detector or descriptor methods used here.

8 Evaluation of the Implemented Methods

The performance evaluation in this project will be limited to making some relevant experiments on the set of test images supplied at:

- `http://www.robots.ox.ac.uk/~vgg/research/affine`

The reason is that, these test images have estimated homographies available and that, this is a well-known set of test images for the kind of methods presented in this report.

The evaluations that will be performed for the feature *detectors* are computation of *repeatability* and *location accuracy*, in the spirit of [Miko06] and [Miko04].

For feature *descriptors*, we will compute *recall* and *1-precision*. [Miko05] shows the recall of the descriptors as a function of 1-precision, by varying their match criterion threshold to obtain differing 1-precision values. To limit the extent of this evaluation, we keep a fixed nearest neightbour to second nearest neighbour distance ratio of 0.7, as was described in section 6.7. This means that, we don't have any parameter to change, to get differing 1-precision values. Hence, we will not use 1-precision as in [Miko05], but only as a measure of false positives for the descriptor for a given image pair. Similarly, we will also only use recall as a single value per test image pair.

8.1 Criterion for Correct Feature Matches

To determine if a feature match is correct, we use a predetermined *homography*, a 3x3 matrix, which gives a way of mapping features in one image into the other image, provided that the image features lie on a plane in the depicted scene. We have such homographies available for the set of test images, which is being evaluated.

To determine if a feature match is correct, take the feature x_a in one image, transform it with the homography into x'_a in the other image and check that, the corresponding feature x_b in that image in some sense approximates the transformed feature x'_a.

There are at least two ways of measuring whether x_b approximates x'_a. A conceptually simple way is to check that, the distance between the features is within some maximum allowed distance. If we assume that, x_b and x_a are the central feature points, this can be expressed by the formula:

$$\|x_b - Hx_a\| \quad < \quad \epsilon \tag{12}$$

This is what is used in [Miko04], where they use an ϵ value of 1.5 pixels. However, measuring pixel distances is a somewhat arbitrary measure, since, for example, it depends on the image resolutions.

In [Miko04], [Miko05] and [Miko06], they also use an overlapping criterion of transformed ellipses, which formalises the intuitive desciption given in the introduction of this report, in section 2.2. This comparison not only evalutes the distances between the features, but also e.g. how accurately the scale and affine invariance is achieved by the feature detector.

The overlap criterion can be defined by the following formula, if we assume that C_a and C_b are the circles associated with the central feature points x_a and x_b:

$$1 - \frac{C_b \cap H^{-T} C_a H^{-1}}{C_b \cup H^{-T} C_a H^{-1}} \quad < \quad \epsilon_o \tag{13}$$

First off, it should be noted that, a circle, or more generally a conic C, is transformed into another conic C' (which will be an ellipse, when C is a circle) by a homography H according to the formula $C' = H^{-T} C H^{-1}$, where the conic is represented by a special $3x3$ matrix. Notice the notation, where H^T means the transposed matrix, H^{-1} the inverse matrix and H^{-T} the transposed of the inverse matrix. The transformation of conics is described in section 2.3.1 in [Hart03] and we shall not dig into the details of it here. We use $C \cap C'$ to denote the area of the intersection between the two conics, here a circle C and an ellipse C', and we use $C \cup C'$ to denote the area of their union.

The above formula expresses that, the *overlap error* between the two areas should not be more than ϵ_o. In [Miko04], [Miko05] and [Miko06], they consider the overlap criterion to be met, when the overlap error is less than 40 percent, meaning that ϵ_o is 0.4. We use the same threshold here.

Since we are only using circles as measurement areas in the implementation, the overlap criterion (equation 13) most likely gives larger errors than what could have been achieved, if we were using affine ellipses. However, the overlap criterion possibly also gives some insights into how much our circles are wrong, in terms of affine geometry changes between the images, since a circle in one image is transformed into an ellipse in the other image. Indeed, earlier in this project, the evaluations in this section were done with the pixel distance criterion from equation 12 above, where better results were obtained than for the overlap criterion (equation 13) for larger differences in viewing angles, but worse results than for the overlap criterion for smaller differences in viewing angles.

As mentioned in section 4 (from section 4 in [Miko06]), increasing the size of the measurement area, as e.g. the implemented SURF descriptor does, generally increases the overlap in overlap tests, giving a false sense of success. However, we are measuring the overlap of the circles, whose radius is established as the scale s of the detected regions, from their detected area in pixels, as described in section 6.5. Hence, this should not be an issue here.

8.2 Repeatability for Detectors

The performance of the detector is measured by its *repeatability*. It is defined by the ratio between the total number of candidates for feature correspondences and the number of detected features in the first image of the image pair, which are within the frame of the second image:

$$repeatability \quad = \quad \frac{\#\ all\ candidates\ for\ correspondences}{\#\ detected\ features\ in\ first\ image\ within\ frame\ of\ second} \tag{14}$$

Features are considered to be candidates for correspondences, when they meet the overlap criterion in equation 13, with $\epsilon_o = 0.4$. The candidates for feature correspondences are searched for among all

detected features in the first image, where a feature meeting the overlap criterion exists in the other image. Notice that, this search is done by the one-way brute-force match, described in section 6.7, except that the match criterion is not the nearest-to-second-nearest-neighbour ratio but the overlap criterion in equation 13.

Notice that, the denominator uses the number of features in the first image, which are within the area of the second image. The reason for considering the features in the first image is that, the one-way matching assigns at most one feature from the second image to each feature in the first image. Hence, the number of features in the first image is the theoretical upper limit on the number of *candidates for feature descriptor matches*. The reason for considering only the features, which are within the frame of the second image is that, only these features can be candidates for *correct correspondences*. Hence, the denominator is the upper limit on the number of possible correct correspondences. [Miko04] and [Miko06] use the number of features in the image with the smallest number of features. To do this correctly, we would have to switch roles between the images, such that the image with fewest features is the first image in the one-way matching. However, to limit the extent of the evaluation, this has not been done. The repeatability should be as high as possible, ideally 1, also represented as 100 percent.

We normally use the overlap criterion from equation 13 with $\epsilon_o = 0.4$ when computing repeatability, except when evaluating the location accuracy. For evaluating the location accuracy, we use the pixel distance criterion from equation 12 instead.

8.3 Location Accuracy for Detectors

Another performance measure of the detector is its *location accuracy*, which is significant for how accurate camera view relationships can be determined from its detected features. The location accuracy is evaluated by computing the repeatability as a function of varying ϵ in the pixel distance criterion from the equation 12.

8.4 Recall for Descriptors

The descriptor performance is partly evaluted by computing the *recall* value. This is defined as the ratio between the correctly matched features and the total number of candidates for feature correspondences:

$$recall \quad = \quad \frac{\#\ descriptor\ matches\ found\ and\ being\ correct}{\#\ all\ candidates\ for\ correspondences} \tag{15}$$

Features are considered both correct matches and candidates for correspondences, when they meet the overlap criterion from equation 13 with $\epsilon_o = 0.4$. The correct matches are searched for among the descriptor matched correspondences, while the candidates for feature correspondences are searched for among among all detected features in the first image, where a feature meeting the overlap criterion (from equation 13 with $\epsilon_o = 0.4$) exists in the other image. The recall value should be as high as possible, ideally 1, also represented as 100 percent.

Notice that, the denominator of the recall is the same as the nominator of the repeatability.

8.5 1-Precision for Descriptors

The other performance measure for the descriptor is *1-precision*. This is defined as the ratio between the number of incorrect descriptor matches and the number of feature correspondences found by the descriptor matching:

$$1 - precision \;\; = \;\; \frac{\#\ descriptor\ matches\ found\ but\ being\ incorrect}{\#\ all\ descriptor\ matches\ found\ (correct\ +\ incorrect)} \qquad (16)$$

The incorrect descriptor matches are searched for among the descriptor matched correspondences and consist of those which do not match the overlap criterion (from equation 13 with $\epsilon_o = 0.4$). The 1-precision value should be as low as possible, ideally 0, also represented as 0 percent.

More details about how the recall and 1-precision values can be used for computing things like the number of correct matches can be found in [Miko05]. As mentioned, we are not showing repeatability as a function of 1-precision, as in [Miko05]. We show these values individually per image pair.

8.6 Explanation for the Kinds of Graphs

We shall see several kinds of graphs. Two similar kinds of graphs are the graphs, which show *repeatability* and *recall*, respectively, as a function of the image pairs, which are being matched. Each sequence of six images is of the same scene, but with some property of the image changing gradually throughout the sequence. To form the image pairs for the graphs, the first image is matched pair-wise with the other images in the sequence, images two to six.

In the *repeatability* graphs, a perfect detector would have a repeatability of 100 percent for all images. However, the detector performance usually decreases, as the image pairs become more challenging. In the *recall* graphs, a perfect descriptor would also have a recall of 100 percent for all images. This is also usually decreasing in practice with more challenging image pairs.

The graphs for *1-precision* are also shown as a function of the image pairs, which are being matched. 1-precision is a measure of false positives for the descriptor and the ideal descriptor has a 1-precision of 0. If it is above 50 percent, it may not be possible to automatically obtain an estimate of the camera view relation by the *RANSAC* method. Hence, being able to keep this value low is quite important.

The graphs evaluating *location accuracy* show repeatability as a function of the maximum allowed pixel distance in the pixel distance criterion from equation 12, which is used when calculating the repeatability for location accuracy. This kind of graph is made for one specific matched image pair at a time.

8.7 Performance Graphs

This section contains all graphs for the performance evaluations. All the images and graphs shown here were generated automatically by the implemented performance measurement program. The output of this execution is shown in Appendix E. The output shows the file names of all saved files, where detection and matching is done before saving the resulting images. Thus, it should be possible for the reader to locate the exact numbers of detected and matched features for the evaluations, which may be of some interest.

For each image sequence used, two example images from the sequence is shown first. Then, four example images from the image sequences are shown with correctly matched descriptors, before the graphs, which evaluate those image sequences. Notice that, only the two topmost of such four images are matched with each other. The matched descriptors of the two lower images are also matched with the first image in the sequence, but we don't show the first image multiple times to illustrate this. The descriptors are shown as oriented boxes. For all image sets, homographies are given between the images pair-wise and those homographies are used for verification in the tests.

The graphs for location accuracy are at the end of this section.

Figure 23 shows two images from the graffitti image sequence. This is a sequence of six images of increasing camera viewing angle of a painted wall. Some images in this sequence also have some rotation of the camera. Figures 24 and 25 show the performance evaluation on this image set. The performance is acceptable for the first two image pairs, which should have viewpoint angles of 20 and 30 degrees, respectively. Performance drops significantly for the third image pair, which should have a viewpoint angle of 40 degrees, and becomes practically useless for the last two images. This shows that, the implementation does not handle viewpoint changes very well, which could be attributed to the fact that, it is not handling affine invariance.

Figure 26 shows two images from the wall image sequence. This is a sequence of six images of increasing camera viewing angle of a brick wall, which has regularly repeated texture. Figures 27 and 28 show the performance evaluation on this image set. The performance is acceptable for the first two image pairs. Detector performance and 1-precision practically useless for the last images. Recall for the descriptor keeps its level for image pairs three and four as well. This shows that, the implementation, particularly the detector, does not handle scale changes particularly well. However, the descriptor seems to handle this fine, although these numbers may be misguiding, due to the very low number of correspondences.

Figure 29 shows two images from the boat image sequence. This is a sequence of six images of decreasing camera zoom and changing camera rotation. This gives changes in scale and rotation between the images. Figures 30 and 31 show the performance evaluation on this image set. The detector performance (repeatability) is acceptable for the first two image pairs, which should have viewpoint angles of 20 and 30 degrees, respectively. Detector performance drops significantly for the third image pair, which should have a viewpoint angle of 40 degrees, and becomes even worse for the last two images. The overall performance, including the descriptor (repeatability and 1-precision), is quite bad for all images. 1-precision does not get below 50 percent for any of the images. This confirms that, the detector does not handle viewpoint changes very well. It also shows that, the implementation does not handle regularly repeated texture very well, which is also known to be a challenging scenario for these kinds of methods.

Figure 32 shows two images from the Leuven image sequence. This is a sequence of six images of decreasing camera exposure time, resulting in decreasing illumination. Figures 33 and 34 show the performance evaluation on this image set. The performance starts out acceptably for the first image pairs, but drops steadily throughout the sequence. The detector performance (repeatability) and the 1-precision are more affected than the descriptor performance (recall). This shows that, the detector handles illumination changes somewhat acceptably, but that there is definitly room for improvements. It also shows that, the descriptor seems to handle illumination changes better than the detector.

Figure 35 shows two images from the bikes image sequence. This is a sequence of six images of changing camera focus, resulting in increasing image blur. Figures 36 and 37 show the performance evaluation on this image set. The performance of the detector (repeatability) drops fairly quickly with increasing blur. The 1-precision and descriptor performance (recall) are maintained quite well though, except for the final image pair, where the second image in the pair is very blurry. This shows that, the detector does not handle blur particularly well, but that, the descriptor seems to cope with it quite well.

Figure 38 shows the location accuracy of the detector for the graffitti image set. The top graph is for the first image pair and the bottom graph is for the third image pair, between images one and four. This shows that, location accuracy is quite good for small viewpoint changes, but that it seems to decrease with larger viewpoint changes.

Figure 39 shows the location accuracy of the detector for the boat image set. The top graph is for the first image pair and the bottom graph is for the third image pair, between images one and four.

This shows that, location accuracy is quite good for small viewpoint changes, but that it decreases quite a lot with larger viewpoint changes. This should probably be expected though, since we are measuring the accuracy as a distance in pixels. When one image has a higher resolution than the other image, one pixel in the low resolution image corresponds to several pixels in the high resolution image, giving inaccurate locations.

Figure 40 shows the location accuracy of the detector for the bikes image set. The top graph is for the first image pair and the bottom graph is for the third image pair, between images one and four. This shows that, location accuracy is quite good and not severely affected by blur. This is may be surprising, since blur severely affects how sharp edges appear.

Figure 23: Two pictures from the graffitti image set, which is a sequence of six images of increasing camera viewing angle of a painted wall. Some images also have some rotation of the camera. Homographies are given between these images pair-wise and those homographies are used for verification in the tests. A homography is valid for verification here, because the scene is a planar. Top: First image in the sequence. Bottom: Third image in the sequence

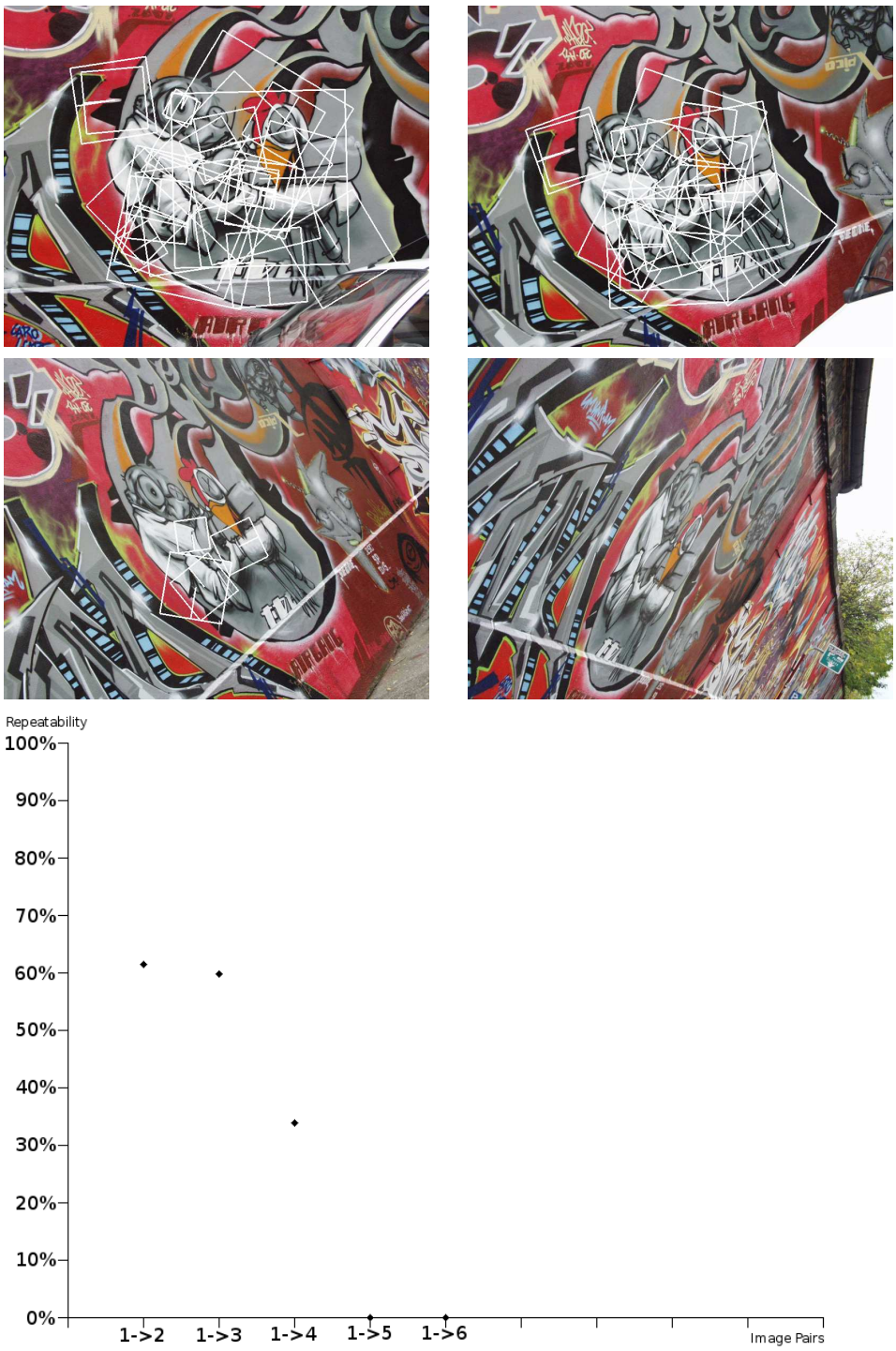

Figure 24: Top: Images one, two, four and six of the graffitti image set, which has increasing camera viewing angle, as well as some rotation in some images. Bottom: Repeatability of the detector for the six images of the graffitti scene, showing detector performance as a function of increasing viewing angle

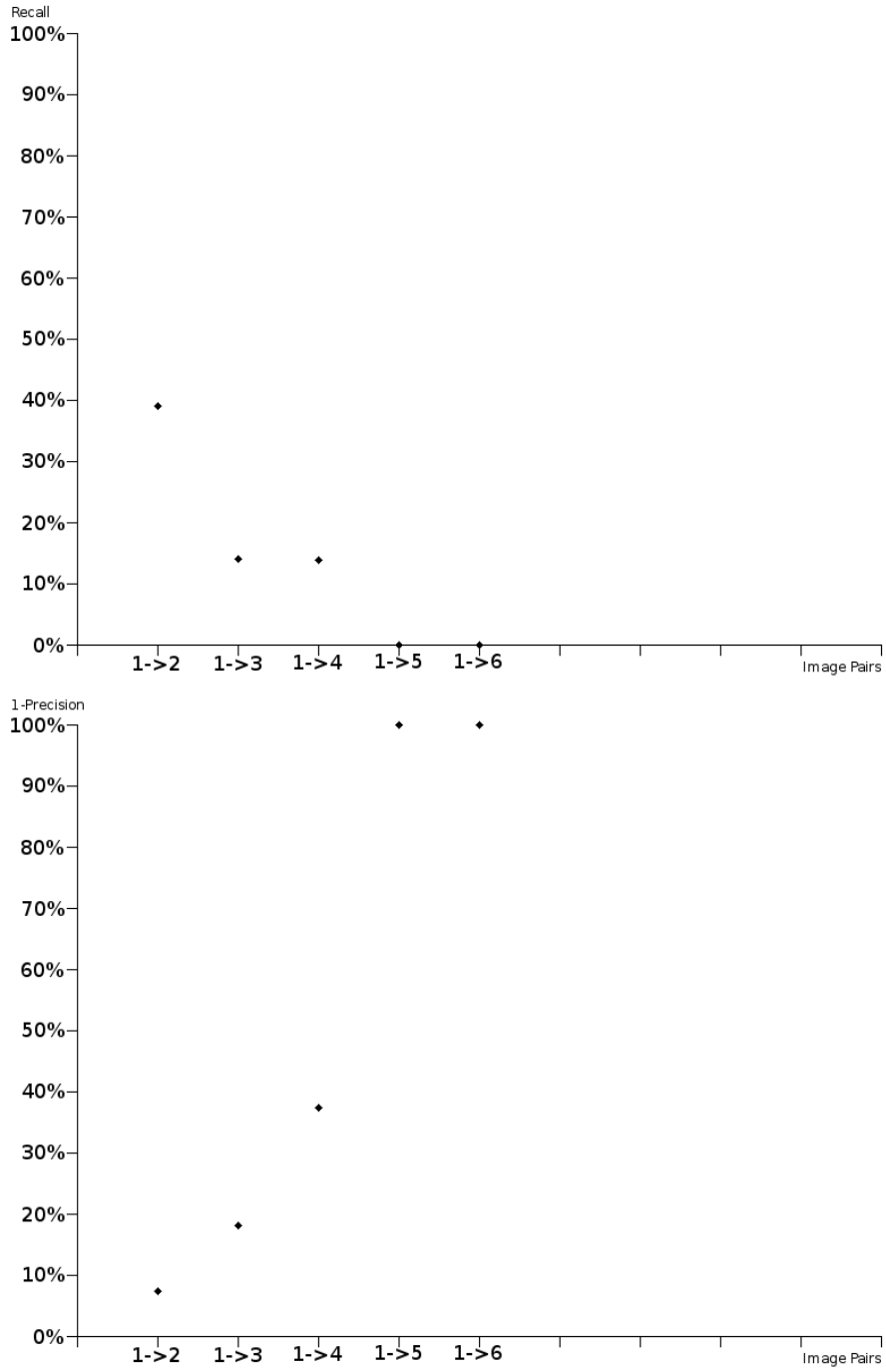

Figure 25: Top: Recall of the descriptor for the six images of the graffitti scene, showing descriptor performance as a function of increasing viewing angle. Bottom: 1-precision for the same sequence, showing the descriptor's false positives as a function of increasing viewing angle

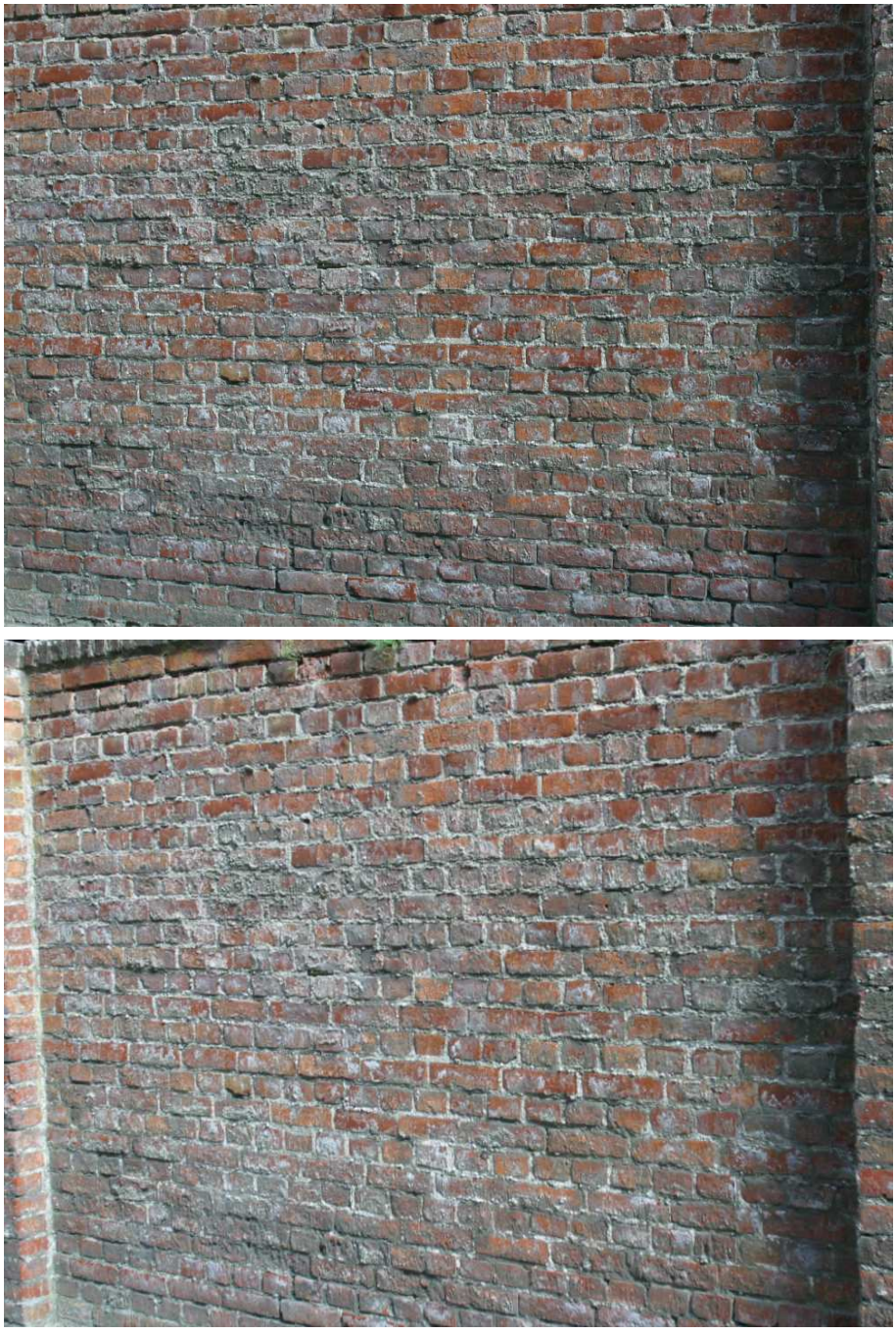

Figure 26: Two pictures from the wall image set, which is a sequence of six images of increasing camera viewing angle of a wall. The wall has a regularly repeated textured surface. Homographies are given between these images pair-wise and those homographies are used for verification in the tests. A homography is valid for verification here, because the scene is a planar. Top: First image in the sequence. Bottom: Third image in the sequence

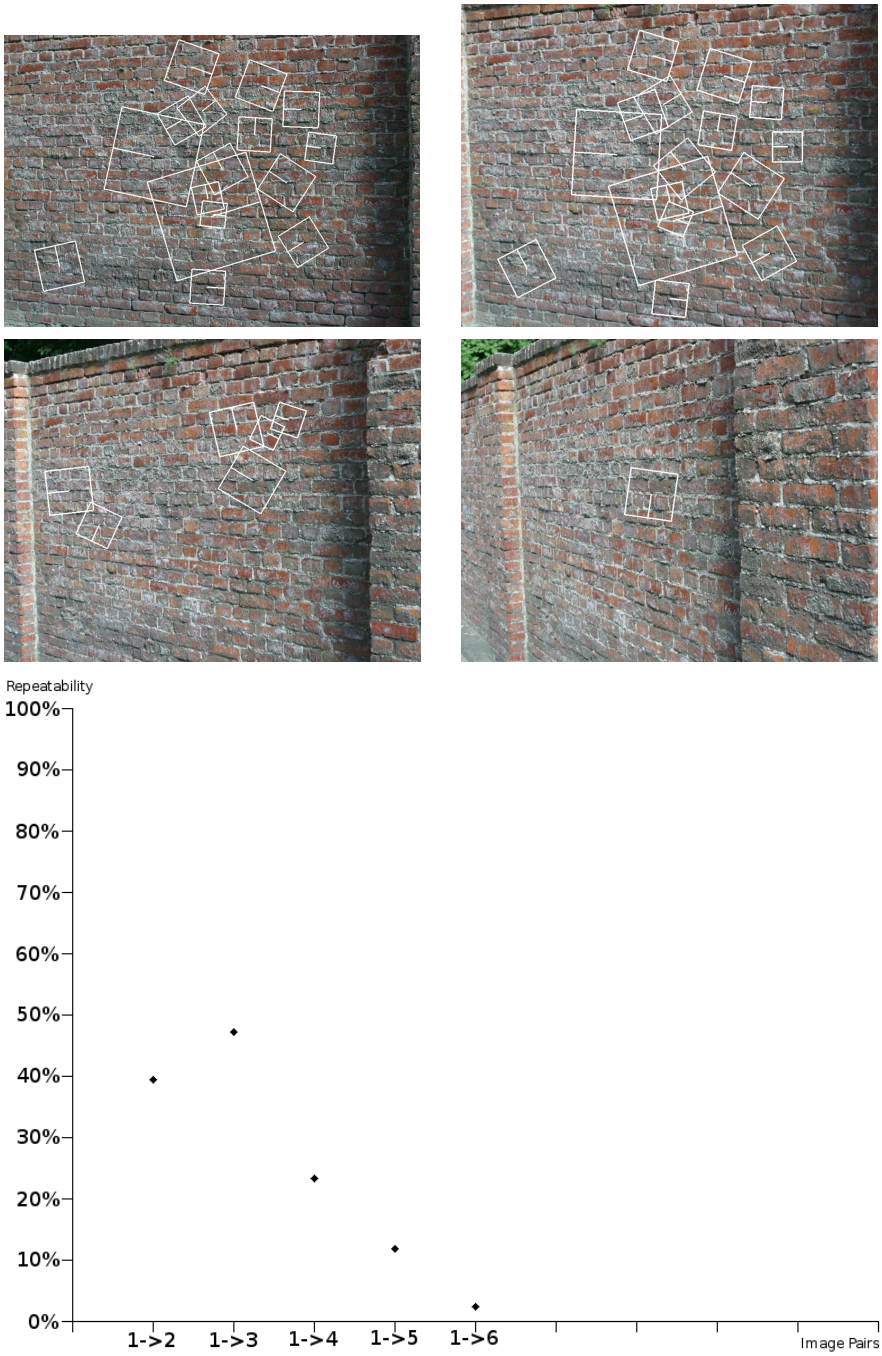

Figure 27: Top: Images one, two, four and six of the wall image set, which has increasing camera viewing angle of a regularly repeated textured surface. Bottom: Repeatability of the detector for the six images of the wall scene, showing detector performance as a function of increasing viewing angle on regular repeated texture

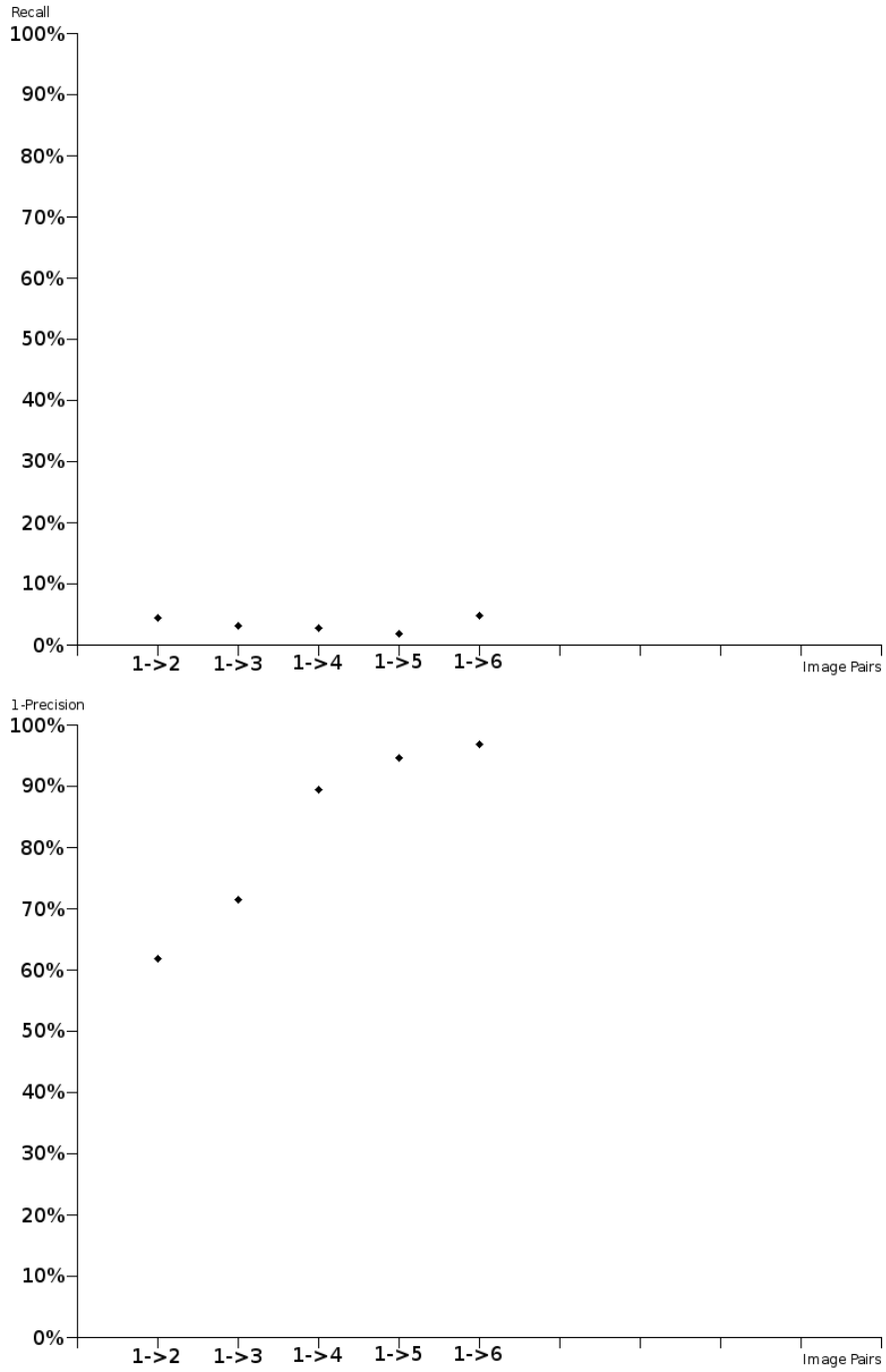

Figure 28: Top: Recall of the descriptor for the six images of the wall scene, showing descriptor performance as a function of increasing viewing angle on regular repeated texture. Bottom: 1-precision for the same sequence, showing the descriptor's false positives as a function of increasing viewing angle on regular repeated texture

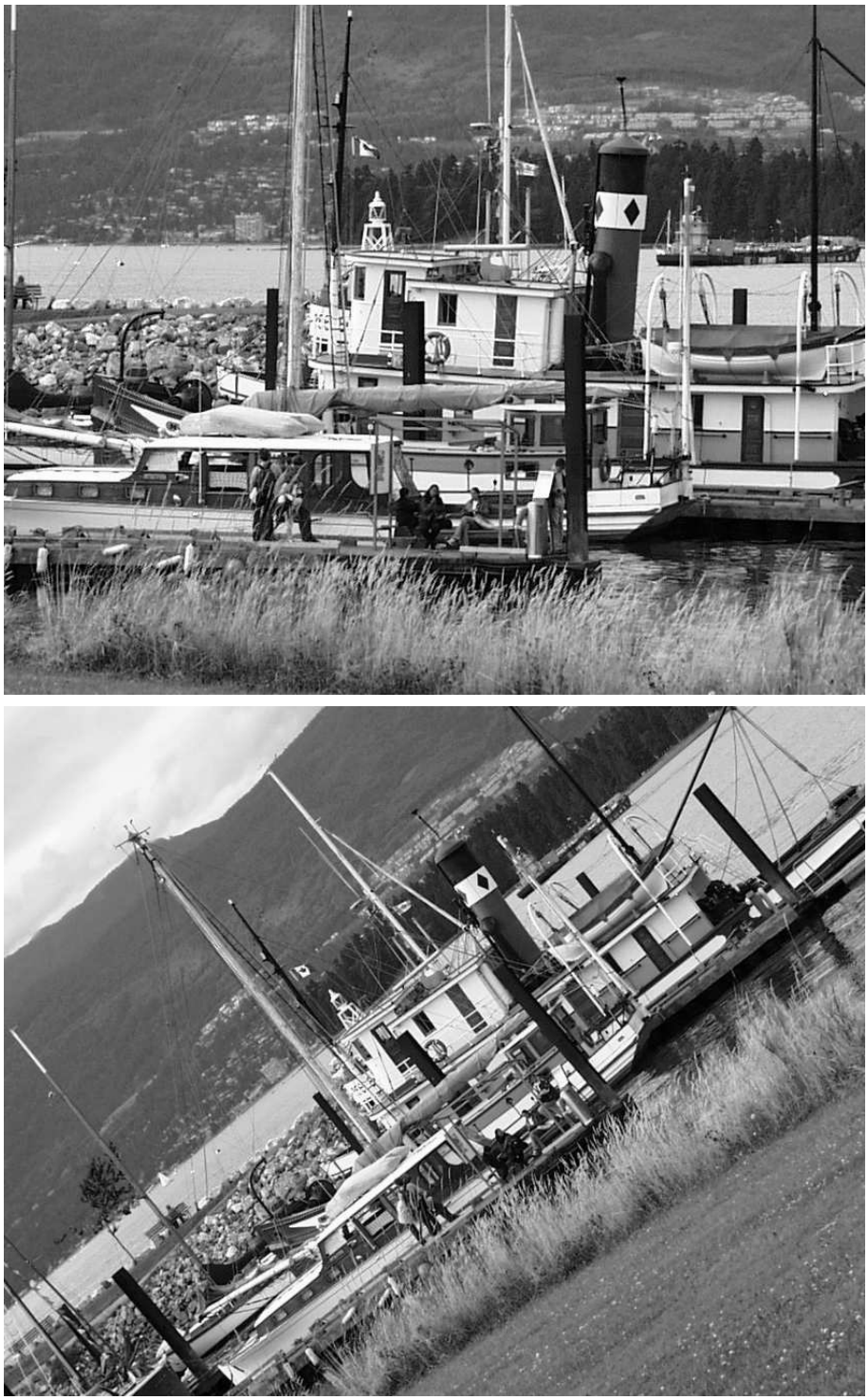

Figure 29: Two pictures from the boat image set, which is a sequence of six images of decreasing camera zoom and changing camera rotation. This gives changes in scale and rotation between the images. Homographies are given between these images pair-wise and those homographies are used for verification in the tests. A homography is valid for verification here, because the camera position is fixed, so the image plane can be considered a planar surface of the scene. Top: First image in the sequence. Bottom: Third image in the sequence

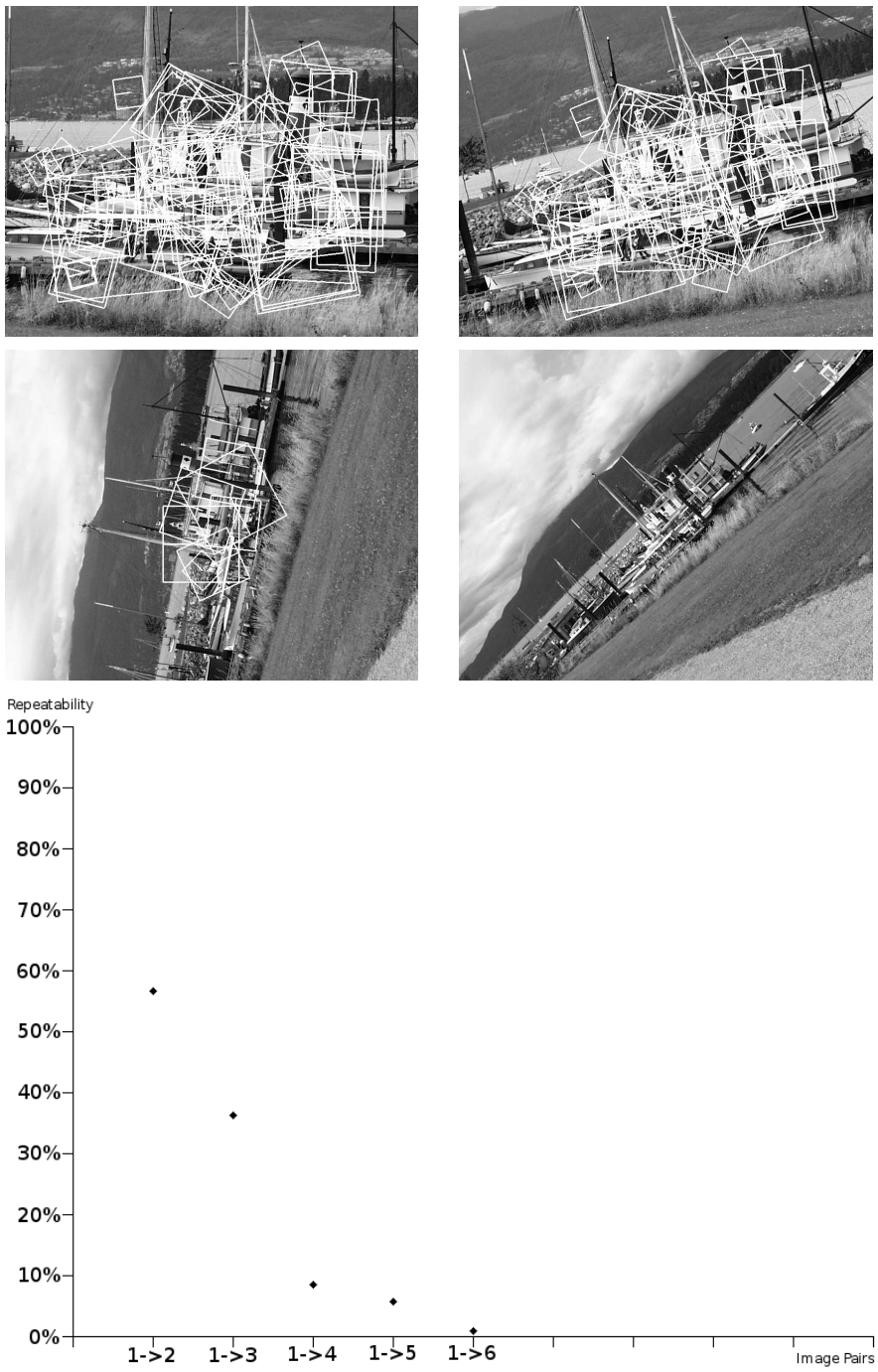

Figure 30: Top: Images one, two, four and six of the boat image set, which has increasing scale and changing rotation. Bottom: Repeatability of the detector for the six images of the boat scene, showing detector performance as a function of increasing scale and some changes in rotation

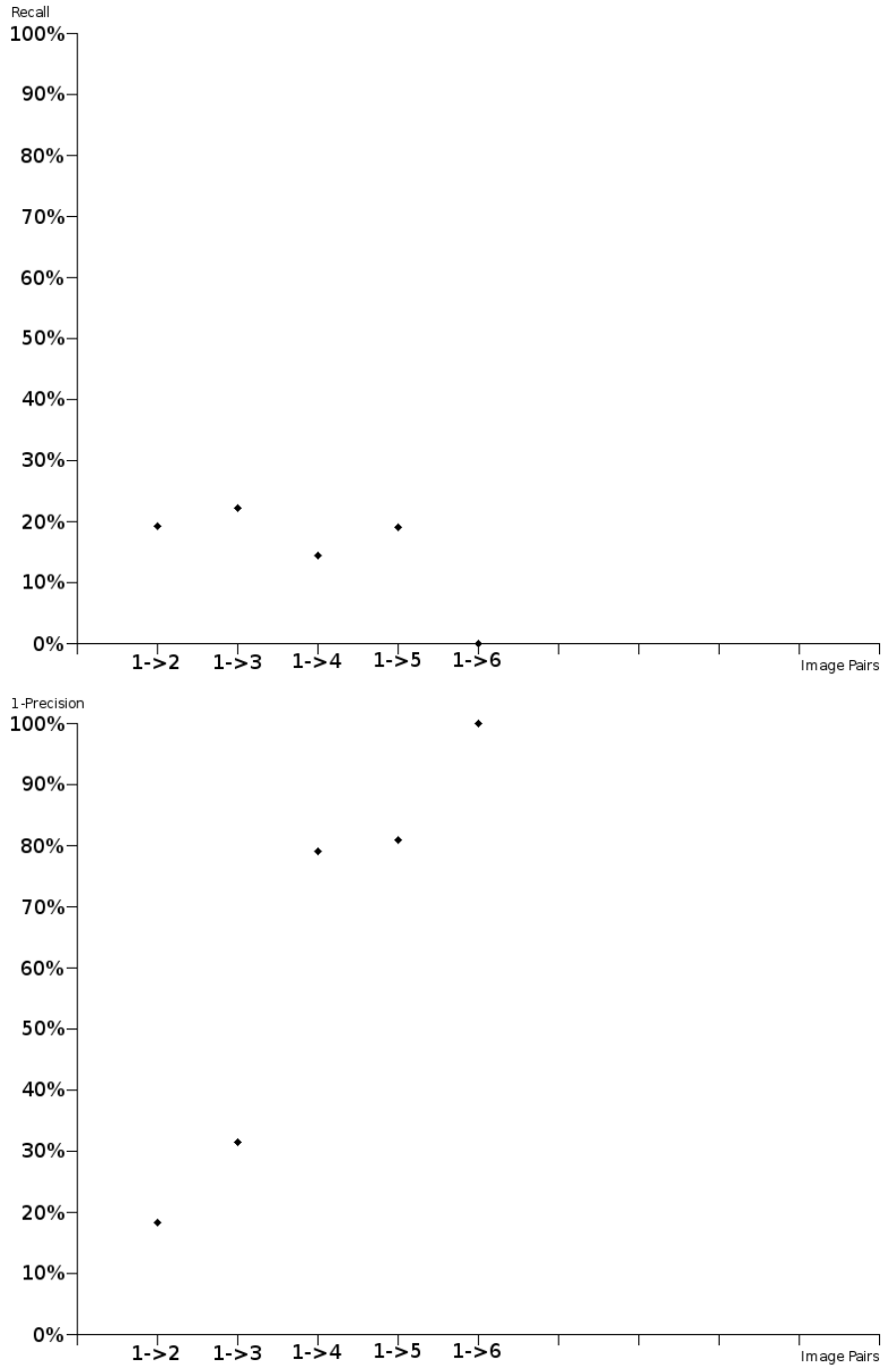

Figure 31: Top: Recall of the descriptor for the six images of the boat scene, showing descriptor performance as a function of increasing scale and some changes in rotation. Bottom: 1-precision for the same sequence, showing the descriptor's false positives as a function of increasing scale and some changes in rotation

Figure 32: Two pictures from the Leuven image set, which is a sequence of six images of decreasing camera exposure time, resulting in decreasing illumination. Homographies are given between these images pair-wise and those homographies are used for verification in the tests. A homography is valid for verification here, because the camera position is fixed, so the image plane can be considered a planar surface of the scene. Top: First image in the sequence. Bottom: Third image in the sequence

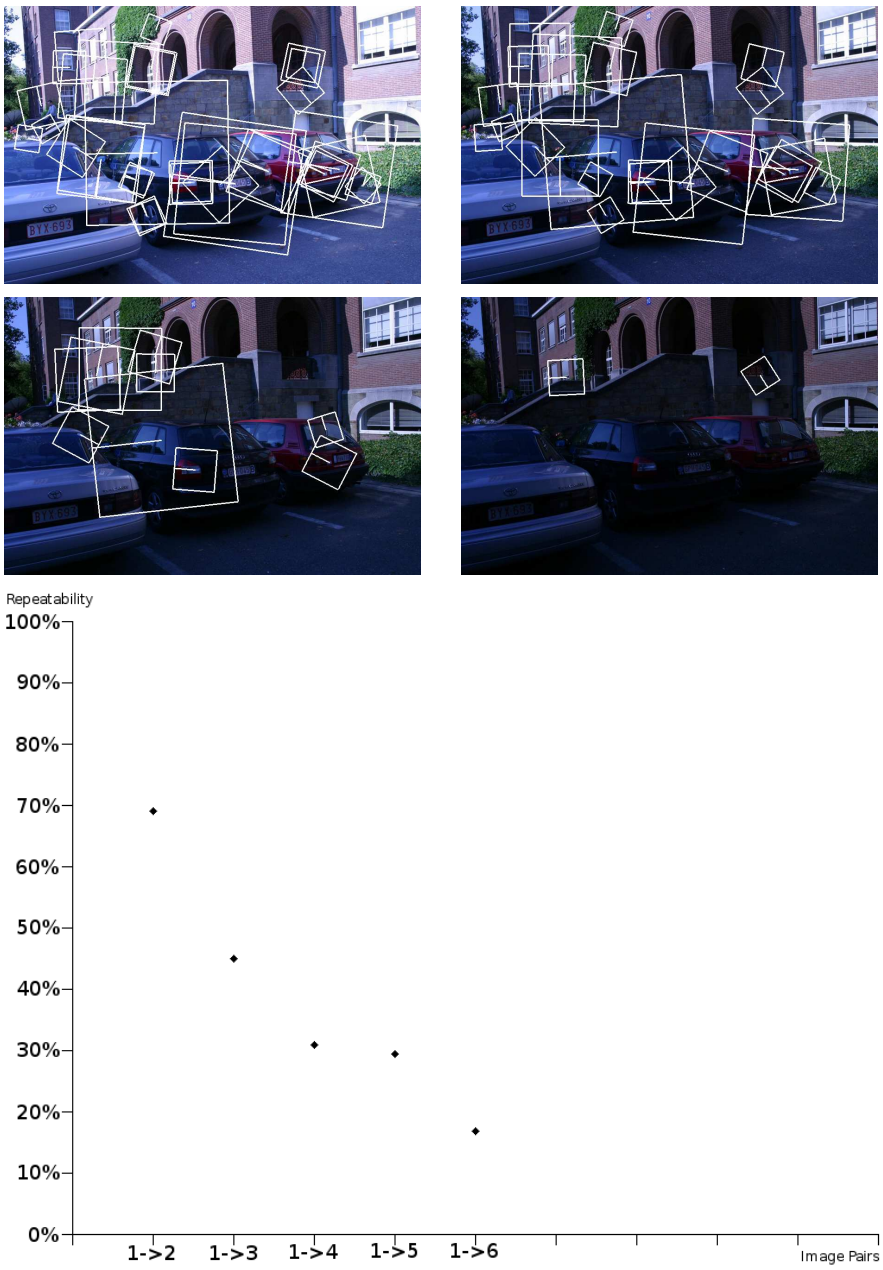

Figure 33: Top: Images one, two, four and six of the Leuven image set, which has decreasing illumination, due to increasing the camera shutter time. Bottom: Repeatability of the detector for the six images of the Leuven scene, showing detector performance as a function of decreasing illumination

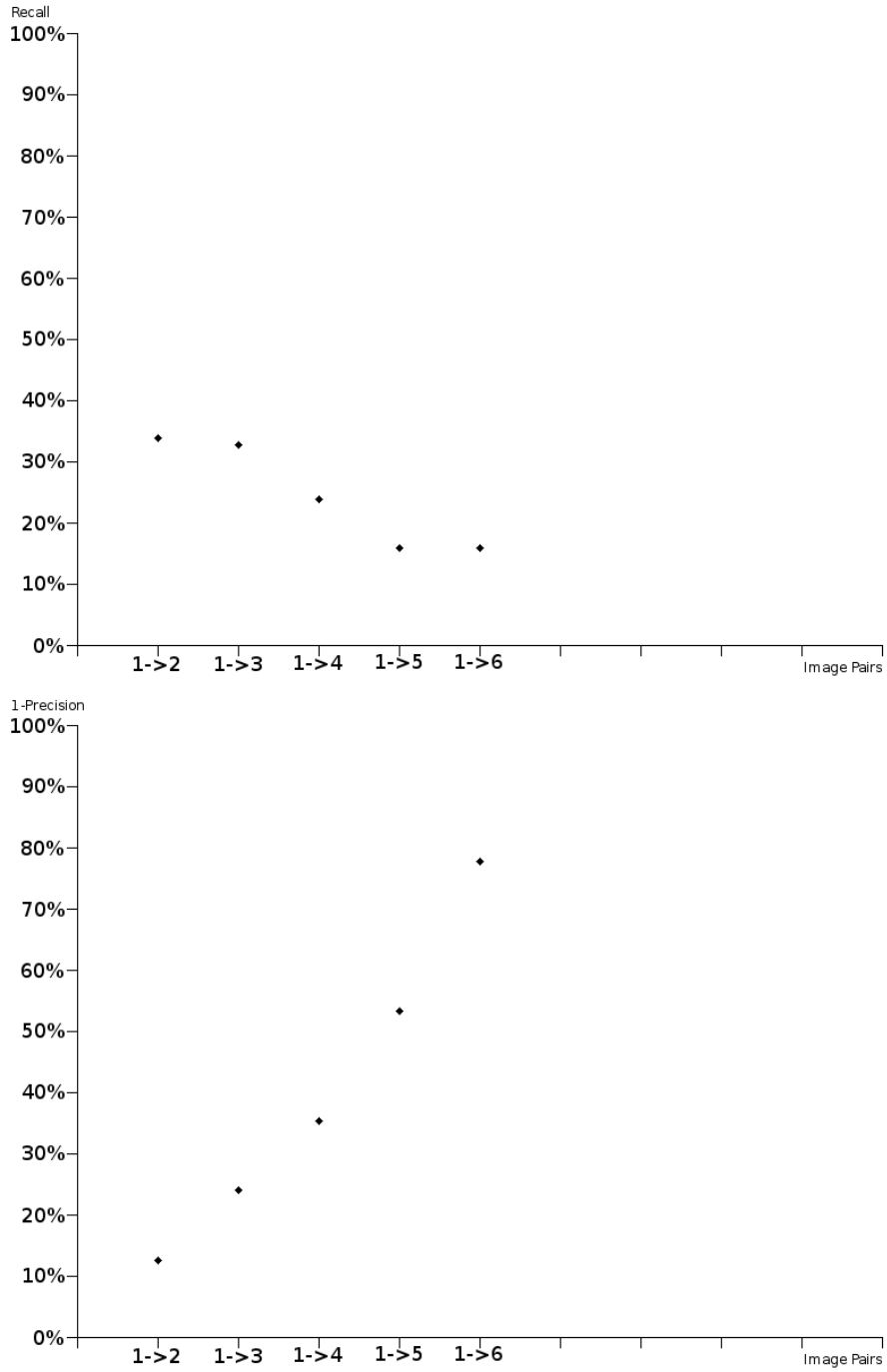

Figure 34: Top: Recall of the descriptor for the six images of the Leuven scene, showing descriptor performance as a function of decreasing illumination. Bottom: 1-precision for the same sequence, showing the descriptor's false positives as a function of decreasing illumination

Figure 35: Two pictures from the bikes image set, which is a sequence of six images of increasing blur. Homographies are given between these images pair-wise and those homographies are used for verification in the tests. A homography is valid for verification here, because the camera position is fixed, so the image plane can be considered a planar surface of the scene. Top: First image in the sequence. Bottom: Third image in the sequence

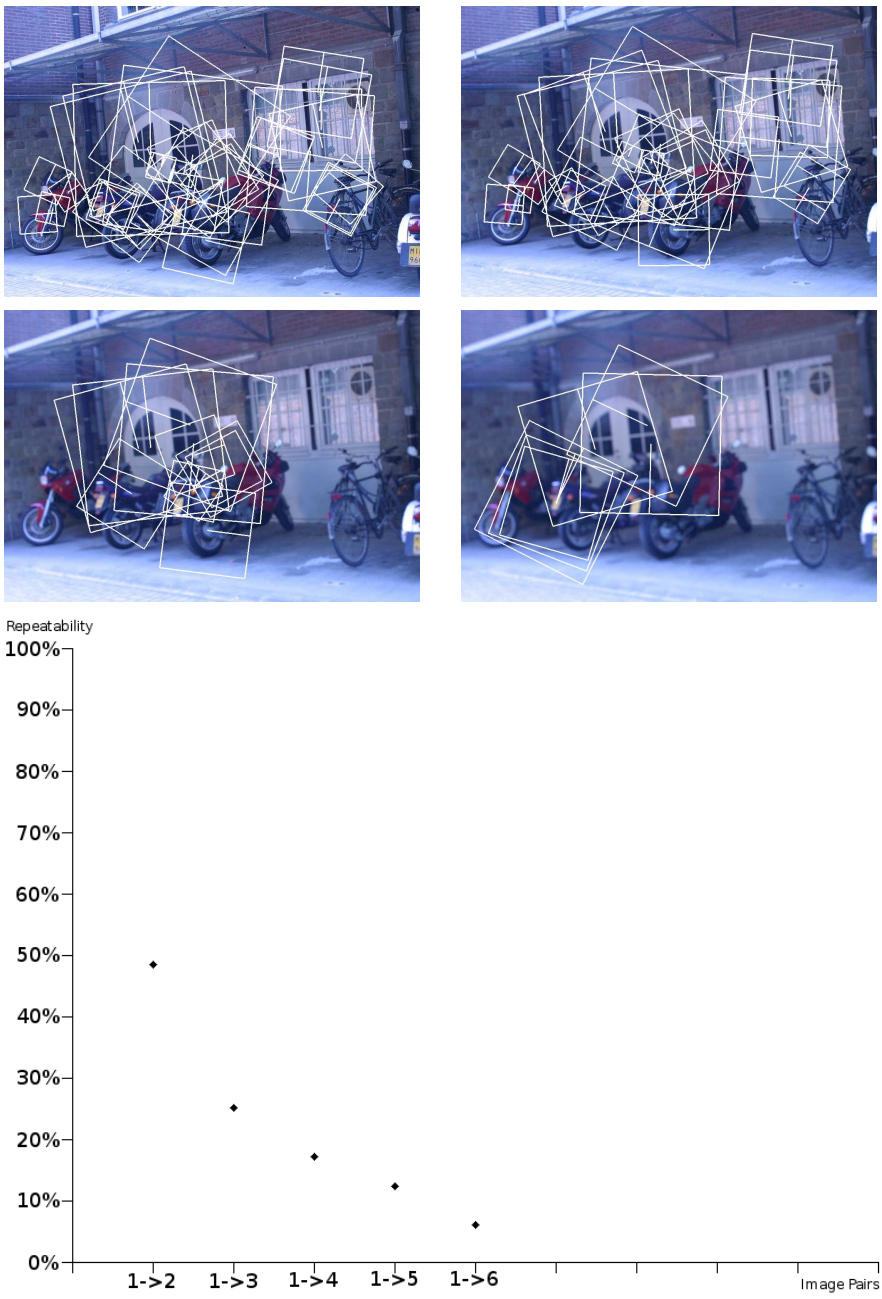

Figure 36: Top: Images one, two, four and six of the bikes image set, which has increasing blur. Bottom: Repeatability of the detector for the six images of the bikes scene, showing detector performance as a function of increasing blur

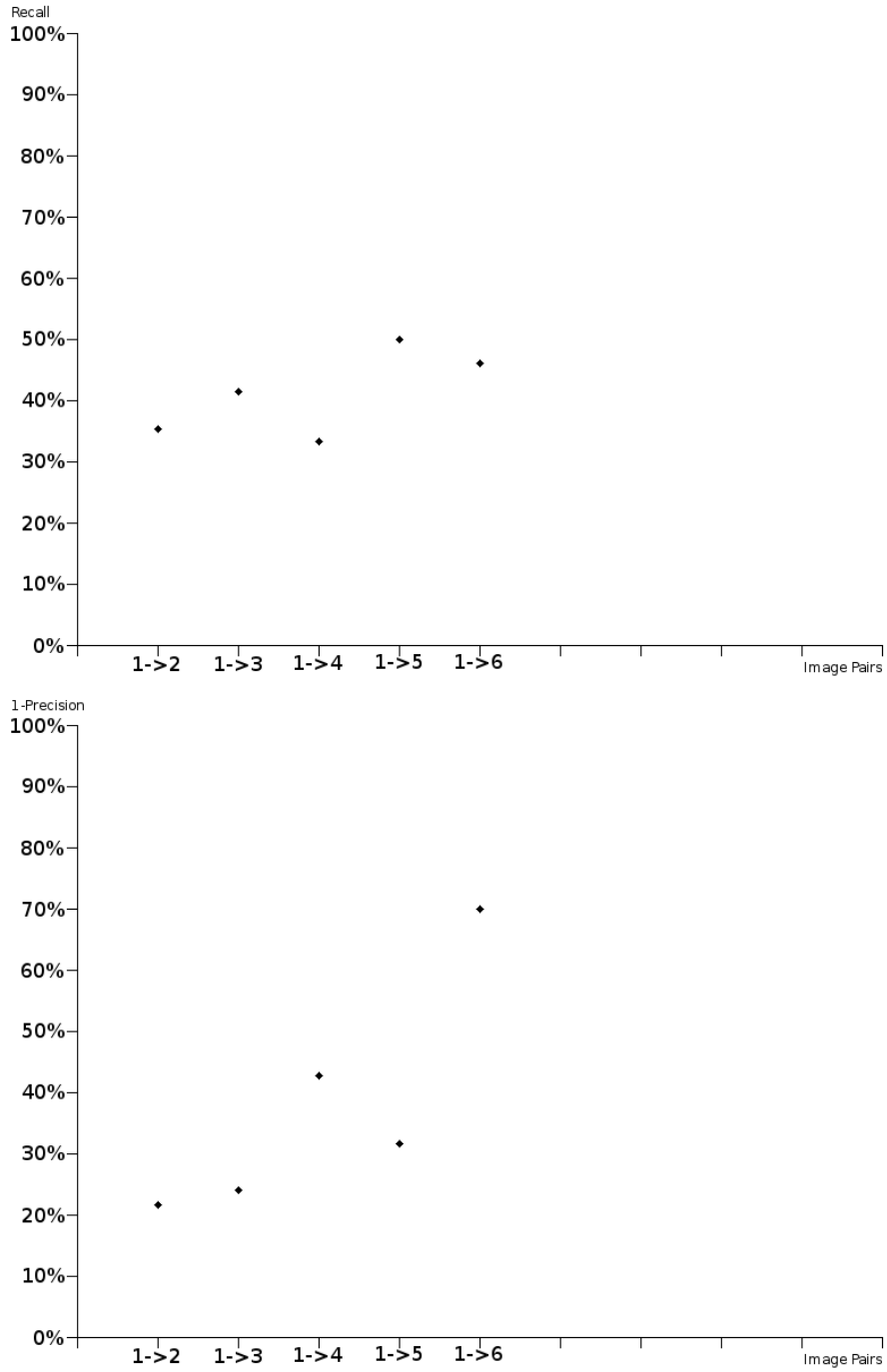

Figure 37: Top: Recall of the descriptor for the six images of the bikes scene, showing descriptor performance as a function of increasing blur. Bottom: 1-precision for the same sequence, showing the descriptor's false positives as a function of increasing blur

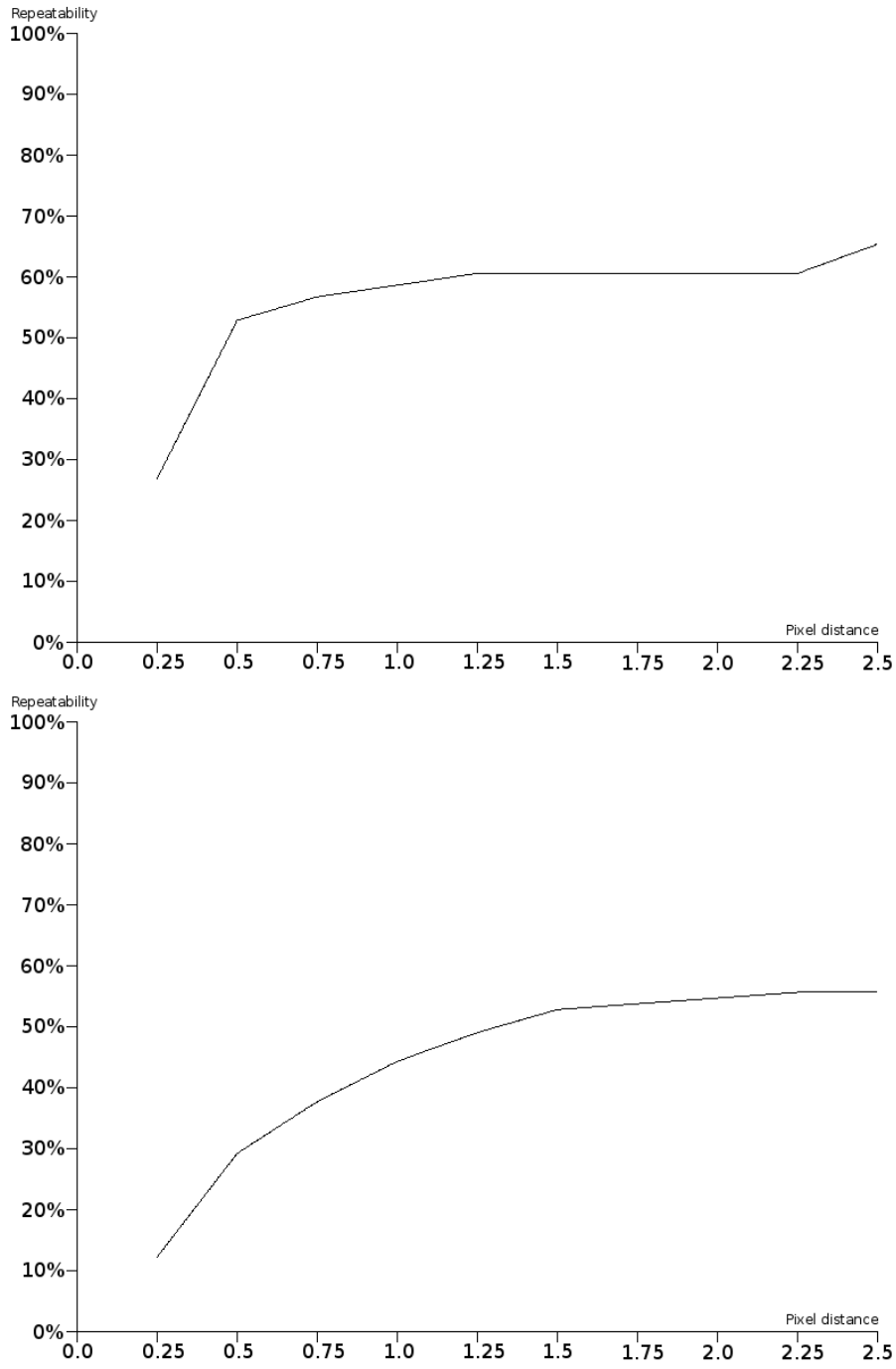

Figure 38: Top: Location accuracy of the detector for images one and two of the graffitti scene, showing detector accuracy for the smallest difference in viewing angle in that image set. Bottom: Location accuracy for images one and four of the same scene, showing the accuracy for a larger difference in viewing angle

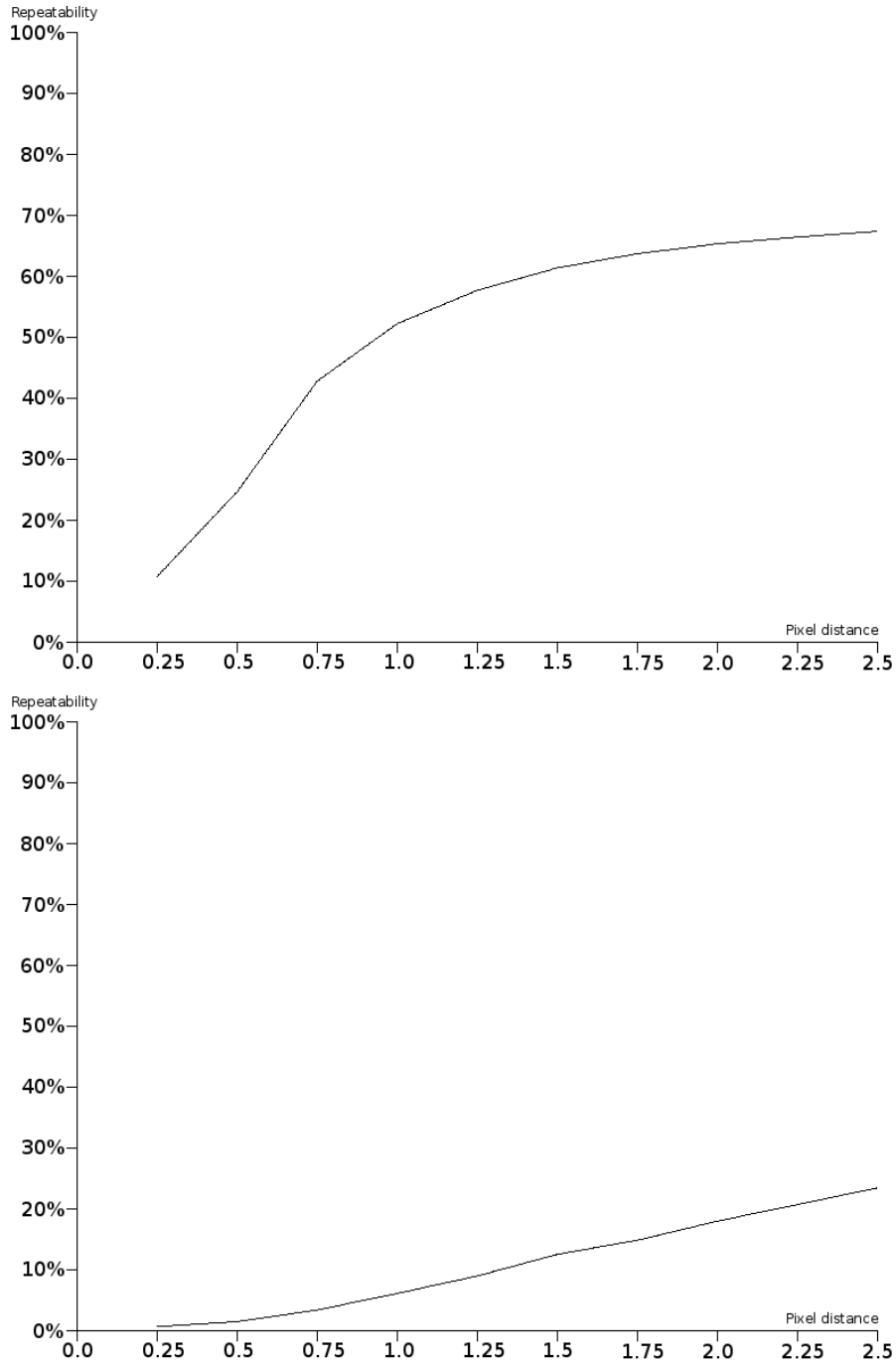

Figure 39: Top: Location accuracy of the detector for images one and two of the boat scene, showing detector accuracy for the smallest difference in scale in that image set. Bottom: Location accuracy for images one and four of the same scene, showing detector accuracy for larger difference in scale

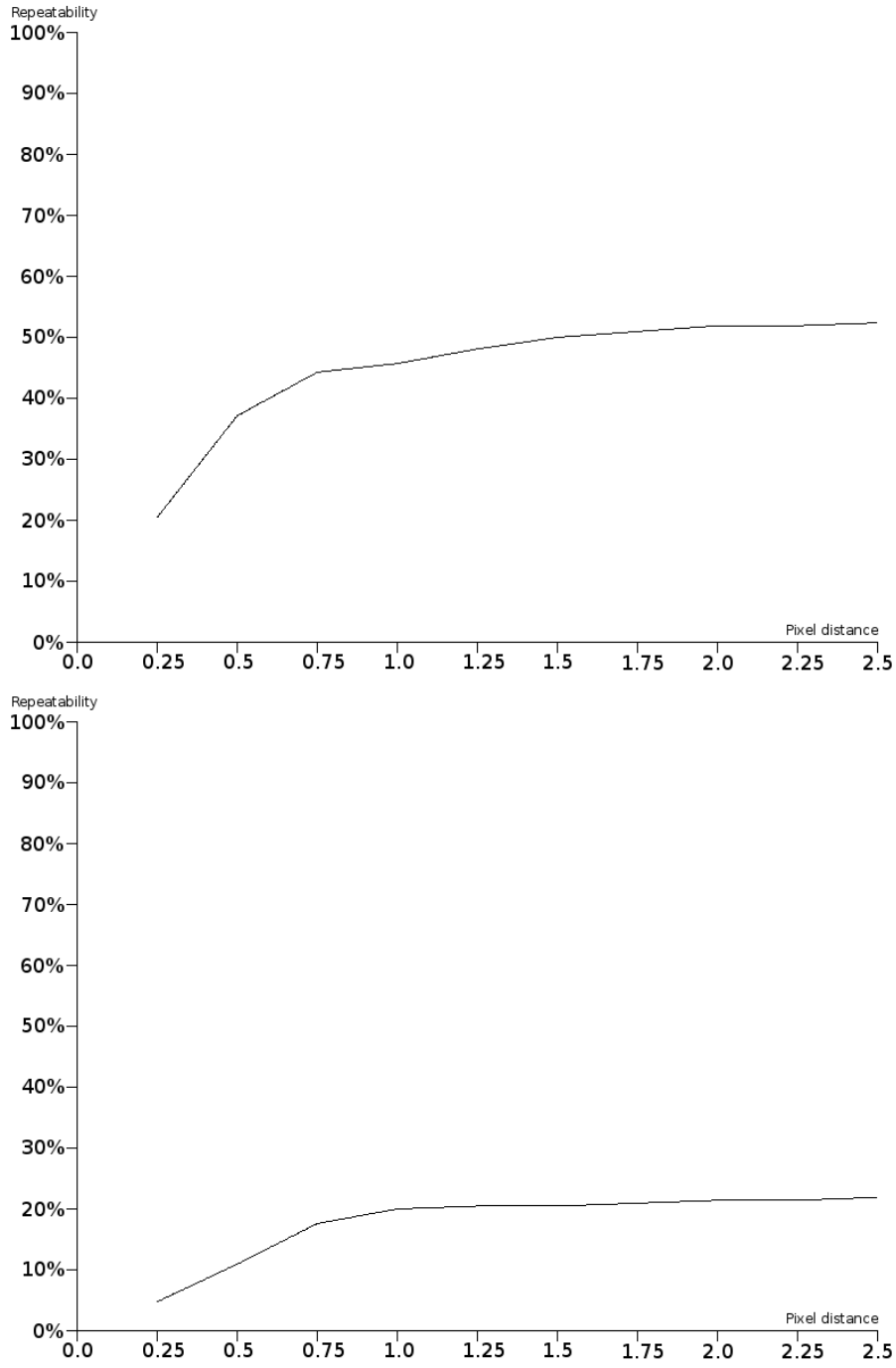

Figure 40: Top: Location accuracy of the detector for images one and two of the bikes scene, showing detector accuracy for the smallest amount of blur in that image set. Bottom: Location accuracy for images one and four of the same scene, showing detector accuracy for larger amounts of blur

8.8 Conclusions of Performance Evaluation

The performance evaluation shows that, overall, the SURF descriptor seems to perform better than the MSER detector. It also shows that, for handling larger viewpoint changes, improvements of the methods are needed, but this can at least be explained by the fact that, affine invariance is not handled. For challenges like scale changes, the detector does not perform well and should probably be improved. For repeated texture, the entire implementation does not perform well. This is known to be a challenging problem, but it should be possible to handle better than here. Overall, blur and illumination changes are handled better than the other challenges. Illumination changes are also less challenging, but blur is usually fairly challenging.

All in all, the implementation is a good starting point, but should certainly be improved.

9 Future Work

Most of the proposals for future work are given in section 7. A complete summary of possible future work is the following:

- Improving the SURF-128 descriptor with a circular sample footprint and reduced dimensionality by using different sample patterns
- Improving the methods to handle geometric affine invariance, particularly the measurement areas and the descriptor
- Adding normalized image intensity cross-correlation for the matching
- Selecting measurement areas with several size scale factors for the descriptor, e.g. factors 1, 1.5, 2 and 3 of the scale of the detected feature, as suggested in [Mata02]
- Adding a point-based detector and matching on the features detected with each kind of detector independently. The Fast Hessian detector from [BayH06] would be interesting, since it is fast, uses little memory and much code can be reused from the SURF descriptor. Hessian-Affine from [Miko06] is also interesting, since it handles affine invariance
- Handling more than two input images
- Evaluating the nearest-to-second-nearest ratio criterion and considering to change it such that, the second nearest match is replaced with the average of the second nearest matchs from all other input images
- Forming the complete pipeline for camera view estimation and eventually a full application
- Fine-tuning the methods to improve the overall performance and to make the methods work as well as possible in each stage of a pipeline
- Evaluating how the MSER and Fast Level Set Transform (FLST) ([Oshe06] chapters 7 and 15, [Case08], [Mona00] and [Mona99]) methods compare as feature detectors would be interesting
- Considering to take advantage of colour information

10 Conclusions

This report gives a quite comprehensive overview of feature detectors and feature descriptors, as well as an overview of matching methods. This includes a detailed overview of previous work in the area with very concrete conclusions with specific references to the relevant work.

A complete image feature detection and correspondence matching framework has also been built, along with an evaluation framework for evaluating its performance. The implementation gives some fairly usable results and the evaluation framework and the references given forms a stable basis for improving it further.

The **contributions of this report** can be summarized as follows:

- A fairly detailed introduction to (section 2) and comparison of previous work (section 3) in the area of feature based image correspondences
- A summary of concrete conclusions of previous work (section 4) with specific references
- Implemented (section 6) and, to some extent, evaluated (section 8) the performance of one way of combining the Maximally Stable Extremal Region (MSER) detector from [Mata02] with the Speeded-Up Robust Features (SURF) descriptor from [BayH06]
- Elaborated on the *stability criterion* (section 6.3.3) of the Maximally Stable Extremal Region (MSER) detector from [Mata02], with suggestions for:
  - Using the *minimum values* or the *extent of the intensity ranges* as measures of confidence for detected regions
  - Avoiding detection of *minima close together* on the histories of area change, which result in almost identical regions on top of each other, by either introducing *Hysteresis limits* on the minima or *merging minima close together*
  - Exploring a *different formula* for region stability
- Suggested extensions for the Speeded-Up Robust Features (SURF) descriptor from [BayH06]:
  - Sampling a *circular region* for the descriptor, which makes it possible to calculate descriptors *closer to the image boundaries* than is otherwise possible with the SURF descriptor
  - *Warping the sample pattern* of the descriptor, to allow using it as an *affine invariant descriptor*
  - Suggested *alternative sampling patterns*, which *reduce the dimensionality* of the descriptor, hopefully without decreasing its descriptive power, but this has not been evaluated
- Suggested other improvements to the implemented methods, as summarized in section 9
- Showed how the methods can be implemented elegantly and efficiently in a functional programming language like Standard ML (the appendices)

The **conclusions regarding the implementation** are the following:

- The Maximally Stable Extremal Region (MSER) detector, combined with the Speeded-Up Robust Features (SURF) descriptor can give fairly usable, but, so far, few feature correspondences
- The MSER detector handles at least some cases fairly well, but there are some parameters to fine-tune and required improvements to be aware of, before it starts to work properly

- The SURF-128 descriptor seems very powerful and good at discriminating between features
- Handling affine invariance seems to be important for the MSER detector, so the conclusions from previous work that, affine invariance may not be worthwhile, does not seem to hold for this particular method. It may be more true for methods which detect features like corners, since these kinds of features are probably distorted less by changes in camera viewpoint
- The MSER detector is by far the fastest part of the implementation, so for combining multiple feature detectors in the same implementation, this method is not computationally expensive to include; and it could probably be optimized even more
- It should be easy to optimize the current computational performance of the SURF-128 descriptor
- None of these methods (MSER and SURF-128) require a lot of memory
- Due to using Bin-Sort on pixel intensities in the MSER method, the current implementation is limited to using 8bit images
- Due to the use of the Summed Area tables with unsigned 32-bit precision per pixel, the size of the input images are currently limited to around 4096x4096 pixels (see Appendix A for details)

11 Acknowledgements

Thanks to Søren Ingvor Olsen for supervising this project with constructive feedback and good litterature references. Thanks to Pascal Monasse for comments and references regarding the Fast Level Set Transform (FLST) and its related methods. Thanks to Martin Elsman for the reference to the path-halving optimization for the disjoint unifiable sets data structure. Thanks, in alphabetical order, to David Lowe, Jiri Matas and Pascal Monasse for answering the author's enquiries of patents on the various methods[1].

12 Appendices

12.1 A: Additional Implementation Details

This appendix, Appendix A, contains some general explanations of the implementation; things which may be a bit tricky or relating to the software design.

The full source code for the Disjoint Unifiable Sets data structure is given in Appendix B. The source code for the oriented Haar wavelet filter response is given in Appendix C. Appendix D contains automated unit-test code for the brute-force two-way matching implementations. Appendix E is a log of the execution of the implemented performance evaluation program.

12.1.1 Summed-Area Tables

There are a few details to be aware of, when implementing summed area tables. If we only consider the input images as 8-bit grey-scale images, a summed area image with unsigned 32-bit precision per pixel is only guaranted to work for input images of up to 4096x4096 pixels. This maximum input image size is a limitation of the current implementation.

[1] Information regarding patents will not be given here, since many people working in the industry have employment clauses, which disallows them to read about patents

Notice also that, the addition and subtractions must be done in the correct order for the summed area look-ups, to avoid that the temporary values become negative or overflow, since Standard ML checks for overflow in its `Word32` module. An alternative here would be to use a representation, which overflows in a well-defined way, such as the arithmetic operations used in *C*.

The check that we are inside the frame buffer boundary may be skipped, if we make sure that, we ignore any detected features, which are so close to the frame buffer boundary that, the box used for determining SURF descriptor orientation would not fit. This optimization is currently not done.

12.1.2 The SURF-128 Descriptor

In the calculation of the dominant orientation of vectors, we ignore all response vectors, which have a length smaller than some small epsilon value. This is to avoid degeneracies and divisions by zero in the angle calculations. If the final dominant orientation vector has a length below the same epsilon, which could at least happen if there were no valid vectors left from the previous epsilon check, then the angle is set to zero. Again, this is to avoid degeneracies and divisions by zero.

12.1.3 Elegant, Yet Efficient, Design in Standard ML

Some of the features in the Standard ML programming language can contribute to an elegant design. A few examples worth noting are the *parametric types* and the *module system*.

The implementation of the Disjoint Unifiable Sets in Appendix B uses parameterized types. The data structure is parameterized over which data the sets contain. Hence, the data structure can be used for any application requiring a disjoint unifiable sets data structure. The two-way matching functions, for which unit-tests are supplied in Appendix D, also use parameterized types. The type of descriptor value is a parameter, meaning that, the implementation can been used for the descriptor matching in the implementation, with 128-dimensional descriptors and an Euclidian distance metric, while allowing for simple unit-tests to be created, as in Appendix D, where simple integers are used as descriptors. Incidently, the one-way matching is similarly parameterized and has been used for both the descriptor matching and for the computation of some of the performance measures for the evaluation, which also require a one-way matching.

The Standard ML module system has also been used with great advantage. The best example is probably that, most modules operating on images are parameterized by a frame buffer layout and a pixel layout. For example, the MSER detector module is instantiated as:

```
structure MaximallyStableExtremalRegionDetectorRgba32 =
  FMaximallyStableExtremalRegionDetector
    (structure FbTraversal =
       Rgba32FrameBufferTraversal
     structure FbPixelLayout =
       FrameBufferRgba32AsIntensity8BitInt
    )
```

It is instantiated with an RGBA32 (Red, Green, Blue, Alpha) frame buffer layout module, which abstracts the fact that, the frame buffer uses 32-bits per pixel. However, the MSER detector implementation does not depend on this and traverses the frame buffer in a way independent of the memory size and layout of a pixel. The MSER detector module is also instantiated with a pixel layout module, which interprets the pixel layout as a single 8bit integer, a discrete grey value in the range 0 to 255. Hence, if

we wanted to detect MSER regions on a different colour space than grey values of RGB pixels, it is a matter of replacing the pixel layout module parameter.

All module instantiations in Standard ML are statically computed, meaning that, when compiled with a decently optimizing compiler, such as MLton [MLto07], there should not be any performance penalty associated with this. This has also been verified by the author on a previous occasion of doing frame buffer rasterization.

12.2 B: Disjoint Unifiable Sets

This section presents the full implementation of the disjoint unifiable sets, described in section 6.1, since this is an important data structure for the MSER method. The most important operations on the data structure are implemented by the following functions:

- `singleton`: Creation of a singleton set
- `unify`: Unification of two sets
- `findRepresentative`: Find the representative element of a set, given any element of the set
- `isRepresentative`: Returns true for an element, if and only if it is a representative element of its set
- `sameElem`: Determine whether two elements are the same. When used on representative elements, this is equivalent to querying whether the two represented sets are the same set or two disjoint sets

The basic data structure is described in [Corm90]. [Corm90] describes the optimizations: *union by rank* and *path compression*. Union by rank has not been implemented here, but path compression is in the uncommented part of the function `findRepresentative`. The code used (i.e. not uncommented) for `findRepresentative` is known as *path halving*. Using path halving gave a slight improvement in execution time over path compression, which seemed to be between 0.01 and 0.4 seconds in the MSER detection timings in Appendix E, when that change was made. No performance decrease was observed in any of the executions, so it is worth changing those ten lines of code. The image `akropolh2002_6...`, for which MSER regions are detected many times in the log in Appendix E, is 1280x960 pixels, meaning that, around 1.2 million sets are created and on the order of twice as many unify operations are performed. The subsequent extraction of region histories (see section 6.3.3) are also part of the MSER timings in Appendix E though, which also seems to take a significant amount of time and be worth optimizing.

```
signature UNIFIABLE_SETS =
sig
  (* 'a t is the type of set elements, some of which are set representatives *)
  type 'a t

  (* The type of a function, which given two set elements, return the
     two set representatives of the sets, ordered such that the first
     returned set is the preferred one to survive a unify operation. *)
  type 'a setorder = 'a t * 'a t -> 'a t * 'a t

  (* Create a singleton set *)
  val singleton : 'a -> 'a t
```

```
  (* Get the value of the element *)
  val getValue : 'a t -> 'a

  (* Set a new value for the element *)
  val setValue : 'a t * 'a -> unit

  (* Returns whether a given set element is the
     representative element of some set.
     This property may change whenever a unify operation is done.
     Iterating a known list of elements and filtering out only those,
     for which this function returns true, yields the complete list of
     disjoint sets in the iterated list of elements. *)
  val isRepresentative : 'a t -> bool

  (* Return whether or not two elements are the same *)
  val sameElem : 'a t * 'a t -> bool

  (* Find the representative element of the set,
     which the given element is a member of. *)
  val findRepresentative : 'a t -> 'a t

  (* Given the function of type 'a setorder, for ordering the importance
     of which set is to survive unification, it unifies two sets of
     elements, given by a member element from each set. It returns
     the representative element of the unified set and possibly the
     representative of an annihilated set, if the given sets were disjoint. *)
  val unify : 'a setorder -> 'a t * 'a t -> 'a t * 'a t option
end

functor FUnifiableSets() :> UNIFIABLE_SETS =
struct
  (* 'a t is the type of set elements, some of which are set representatives *)
  datatype 'a t =
    Elem of
      (* Whether this would be implemented as a ref of the record
         or as a record with two ref fields, is a design choice. *)
      {merged : 'a t option ref,
       value  : 'a ref
      }

  (* The type of a function, which given two set elements, return the
     two set representatives of the sets, ordered such that the first
     returned set is the preferred one to survive a unify operation. *)
  type 'a setorder = 'a t * 'a t -> 'a t * 'a t

  (* Create a singleton set *)
  fun singleton value =
      Elem
        {merged = ref NONE,
         value  = ref value
        }
```

```
(* Get the value of the element *)
fun getValue (Elem {merged, value}) =
    !value

(* Set a new value for the element *)
fun setValue (Elem {merged, value}, newValue) =
    value := newValue

(* Returns whether a given set element is the
   representative element of some set.
   This property may change whenever a unify operation is done.
   Iterating a known list of elements and filtering out only those,
   for which this function returns true, yields the complete list of
   disjoint sets in the iterated list of elements. *)
fun isRepresentative (Elem {merged, value}) =
    (* Only sets, which have not yet been updated to point to a merged set,
       are representative sets *)
    (case !merged of
       SOME _ =>
         false
     | NONE =>
         true
    )

(* Return whether or not two elements are the same *)
fun sameElem (Elem {merged, value = _},
              Elem {merged = merged', value = _}) =
    (* Set elements are uniquely identified by the ref field: merge *)
    merged = merged'

(* Find the representative element of the set,
   which the given element is a member of. *)
fun findRepresentative (elem as (Elem {merged, value})) =
    (* Currently used implementation: Path halving
       This implementation traverses two steps at a time and updates every
       second traversed set to point "two steps closer to"
       the representative set *)
    (case !merged of
       SOME (e1 as (Elem {merged = merged2, value = value2})) =>
         (case !merged2 of
            SOME e2 =>
              let
                (* Update link only at every second step *)
                val () = merged := (SOME e2)
              in
                (* Recursively search for the representative.
                   Notice the tail-recursion, which improves efficiency *)
                findRepresentative e2
              end
          | NONE =>
              (* This is the representative element, return it *)
              e1
         )
```

```
       | NONE =>
           (* This is the representative element, return it *)
           elem
      )
      (* Alternative implementation: Path Compression:
         This implementation updates the paths of all traversed sets,
         to point directly to the representative set, after it has been found. *)
      (*
      (case !merged of
         SOME e =>
           (* Recursively search for the representative, then update the
              reference to point directly to that representative. *)
           let
             (* Notice: This is not tail-recursive *)
             val rep = findRepresentative e
             val () = merged := (SOME rep)
           in
             rep
           end
       | NONE =>
           (* This is the representative element, return it *)
           elem
      )
      *)

  (* Given the function of type 'a setorder, for ordering the importance
     of which set is to survive unification, it unifies two sets of
     elements, given by a member element from each set. It returns
     the representative element of the unified set and possibly the
     representative of an annihilated set, if the given sets were disjoint. *)
  fun unify setorder (elem1, elem2) =
      let
        val rep1 = findRepresentative elem1
        val rep2 = findRepresentative elem2
      in
        if sameElem (rep1, rep2) then
          (* Sets are already unified *)
          (rep1, NONE)
        else
          let
            (* Order elements according to preference of survival *)
            val (rep1, rep2 as (Elem {merged, value})) = setorder (rep1, rep2)

            (* Annihilate rep2 and make rep1 the new representative *)
            val _ = merged := (SOME rep1)
          in
            (* Return the unified set and the annihilated set *)
            (rep1, SOME rep2)
          end
      end
end

structure UnifiableSets = FUnifiableSets()
```

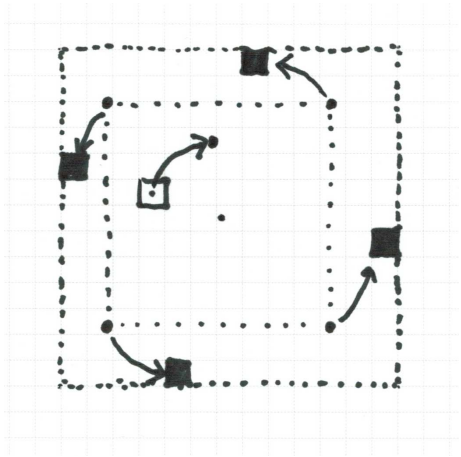

Figure 41: This illustrates part of the computation of the oriented Haar-wavelet filter response. The implementation first determines the four (real valued) corners of an upright box. These are illustrated by small dots. Then rotates the corners according to the orientation of the box, where the resulting four (integer rounded) corners are used for determining a bounding box. These corners are shown as black squares. Within this bounding box, each pixel is traversed. For each traversed pixel, shown as a white square, within this bounding box, the pixel is rotated in the opposite direction as the orientation of the box, in order to transform the pixel into the original upright box, where it is simple to compute the relevant Haar wavelet coefficient

12.3 C: Oriented Haar-Wavelet Filter Response

The source code for the oriented Haar-wavelet filter response, as used in section 6.6.2, is given below. The implementation first determines the four (real valued) corners of an upright box, then rotates them according to the orientation of the box, where the resulting four (integer rounded) corners are used for determining a bounding box. Within this bounding box, each pixel is traversed. For each traversed pixel within this bounding box, the pixel is rotated in the opposite direction as the orientation of the box, in order to transform the pixel into the original upright box, where it is simple to compute the relevant Haar wavelet coefficient. This is illustrated in in figure 41.

We assume that the following functions have been implemented, in addition to the Standard ML Basis Library [SMLB04] modules:

- `Matrix2x2.Const.rotate`:
  Create a 2x2 rotation matrix with a given angle
- `Matrix2x2.Trans.transformPoint`:
  Transform a 2D point with a 2x2 matrix
- `BoundingBox2D.boundPoints`:
  Calculates the bounds of a list of 2D points, in coordinates of type `real`
- `Point2D.subtract`:
  Subtract two 2D points from each other, in coordinates of type `real`
- `IVector2D.add`:
  Adds two 2D vectors to each other, in coordinates of type `int`

- `IVector2D.zeroValue`:
  The constant value `{x = 0, y = 0}`

- `FbTraversal.getOrigin`:
  Gets the upper-left pixel coordinates of the frame buffer

- `FbTraversal.getExtent`:
  Gets the lower-right pixel coordinates of the frame buffer, or actually, one pixel below and to the right of it, referred to as "exclusive" coordinates

- `getPosPixelIntensity`:
  Get the 8-bit integer pixel intensity at the given frame buffer pixel position

- `Int.forl`:
  A higher-order function for looping on integers. The first parameter is the starting value, the second is the number of iterations, the third is the step-value per iteration, the fourth is the accumulating iteration function and the last parameter, is the initial accumulation value. It is implemented in the publicly available project [AMLB09] and will be documented in an upcoming report

```
(* Create a pair of 2x2 rotation matrices:
   The first one rotates angle (in radians) counter-clockwise
   (mathematical standard for angles). The second one clock-wise. *)
fun rotationMatricesFromAngle angle =
    (* Both angles are negated here, becuase the transformation is in
       image pixel coordinates, where the y-axis grows downwards.
       This reverses the angle, compared to mathematical standards *)
    (Matrix2x2.Const.rotate (~angle),
     Matrix2x2.Const.rotate angle)

(* Filter with the oriented Haar wavelet basis for X and Y axis changes,
   respectively, where the orientation is given by an angle *)
fun filterOrientedHaarXyFromRadiusAnglePos
      fbArea (radius, angle) =
    let
      (* Compute rotation matrices once *)
      val (mRotAngle, mRotNegAngle) = rotationMatricesFromAngle angle

      (* Calculate the four corners of the upright filter domain *)
      val upLeft    = {x = ~radius, y = ~radius}
      val upRight   = {x =  radius, y = ~radius}
      val downLeft  = {x = ~radius, y =  radius}
      val downRight = {x =  radius, y =  radius}

      (* Calculate the four corners when rotated *)
      val rotUpLeft    = Matrix2x2.Trans.transformPoint mRotAngle upLeft
      val rotUpRight   = Matrix2x2.Trans.transformPoint mRotAngle upRight
      val rotDownLeft  = Matrix2x2.Trans.transformPoint mRotAngle downLeft
      val rotDownRight = Matrix2x2.Trans.transformPoint mRotAngle downRight

      (* Get the dimensions of the frame buffer area *)
      val {x = origX, y = origY} = FbTraversal.getOrigin fbArea
```

```
val {x = extX , y = extY } = FbTraversal.getExtent fbArea

(* Filter function for a given position, for staged computation *)
fun filterAtPos pos =
    let
      (* Calculate bounds of rotated filter domain centered at pos *)
      val {xmin, ymin, xmax, ymax} =
            BoundingBox2D.boundPoints
              [Point2D.add(pos, rotUpLeft),
               Point2D.add(pos, rotUpRight),
               Point2D.add(pos, rotDownLeft),
               Point2D.add(pos, rotDownRight)]

      (* Calculate integer pixel bounds:
         {x|y}min values are "inclusive" pixel coordinates while
         {x|y}max values are "exclusive" pixel coordinates *)
      val xmin = Real.floor xmin
      val ymin = Real.floor ymin
      val xmax = Real.ceil  xmax
      val ymax = Real.ceil  ymax

      (* Intersect area with frame buffer area, where min values
         are also "inclusive" and max values "exclusive" *)
      val xmin = Int.max(xmin, origX)
      val ymin = Int.max(ymin, origY)
      val xmax = Int.min(xmax, extX)
      val ymax = Int.min(ymax, extY)

      (* Calculate iteration dimensions for sampling pixels *)
      val itersX = xmax - xmin
      val itersY = ymax - ymin

      (* Function for making a filter sample at given pixel center p *)
      fun pixelSample (pixel as {x = ix, y = iy}) =
          let
            (* Calculate pixel center coordinate, by adding 0.5 and
               move it back to the coordinate system centered at
               (0, 0), by subtracting pos *)
            val p = Point2D.subtract
                      ({x = Real.fromInt ix + 0.5,
                        y = Real.fromInt iy + 0.5},
                       pos)

            (* Rotate p back into the upright filter domain *)
            val {x, y} = Matrix2x2.Trans.transformPoint
                           mRotNegAngle p
          in
            (* Check if the new point is within the upright domain *)
            if x >= ~radius andalso
               x <=  radius andalso
               y >= ~radius andalso
               y <=  radius then
              let
```

```
                  (* Get pixel value of the sampled pixel *)
                  val pixelInt = getPosPixelIntensity (fbArea, pixel)
                in
                  (* Calculate responses for the Haar X and Y filters *)
                  {x = if x > 0.0 then
                         pixelInt
                       else
                         ~pixelInt,
                   y = if y < 0.0 then
                         pixelInt
                       else
                         ~pixelInt
                  }
                end
              else
                (* outside the filter domain, no filter response *)
                {x = 0, y = 0}
            end

          (* Function making a pixel sample and adding it to
             the accumulated filter result *)
          fun accPixelSample y (x, acc) =
              IVector2D.add(pixelSample {x = x, y = y}, acc)

          (* Iterate function accPixelSample over one row of pixels *)
          fun iterRow (y, acc) =
              Int.forl (xmin, Word.fromInt itersX, 1,
                        accPixelSample y, acc)

          (* Iterate over the calculated rectangular area of pixels *)
          fun sampleArea () =
              if itersY > 0 andalso itersX > 0 then
                Int.forl (ymin, Word.fromInt itersY, 1,
                          iterRow, IVector2D.zeroValue)
              else
                IVector2D.zeroValue
        in
          (* Calculate the X and Y Haar responses for the area of pixels *)
          sampleArea ()
        end

  in
    (* Return function for second stage of the staged computation *)
    filterAtPos
  end
```

12.4 D: Unit-Test for Brute-Force Matching Strategies

The source code for the unit-tests for three different brute-force two-way matching strategies, as described in section 6.7. The matching strategies are implemented by the functions:

- `M.conservativeBruteForceMatch`

- `M.crossCorrelatedBruteForceMatch`
- `M.aggressiveBruteForceMatch`

These functions all have the Standard ML type:

```
'a metric -> (('a * 'b) list * ('a * 'b) list) -> (('a * 'b) * ('a * 'b)) list
```

where the type `'a metric` is defined by:

```
(* A metric, for the distance between two discriptors *)
type 'a metric = 'a * 'a -> real
```

This interface means that, we can use any type of value as descriptor, since it is parameterized by the type `'a`. The metric determines the distance between two descriptor values as a `real` value.

We assume that, the functions `compareInt` and `compareIiList` have been implemented, for comparing two integers and two lists of pairs of integers, respectively, in addition to the Standard ML Basis Library [SMLB04] modules. The function `TestUtil.doTest` runs the given list of named test functions, to assure that they all return `true`.

```
fun testBasic () =
      let
        (* The first component of the pairs in these lists are the
           integer "descriptor" values whose distance we measure.
           The second component of the pairs can carry arbitrary data,
           so we carry a unique identifier for each list element,
           which is what we use for the comparisons below. *)
        val regions1 = [(96, 1), (1000, 2), (900, 3), (500, 4),
                        (103, 5), (1600, 6), (57, 7), (2000, 8)]
        val regions2 = [(32, 10), (1050, 20), (2050, 30), (1001, 40),
                        (950, 50), (101, 60), (102, 70), (700, 80)]

        (* Run the three different kind of matching algorithms,
           with both possible orders of the two list parameters for each. *)
        val cc12  = M.conservativeBruteForceMatch intDist (regions1, regions2)
        val cc21  = M.conservativeBruteForceMatch intDist (regions2, regions1)

        val ccc12 = M.crossCorrelatedBruteForceMatch intDist (regions1, regions2)
        val ccc21 = M.crossCorrelatedBruteForceMatch intDist (regions2, regions1)

        val ac12  = M.aggressiveBruteForceMatch intDist (regions1, regions2)
        val ac21  = M.aggressiveBruteForceMatch intDist (regions2, regions1)

        fun removeDescriptors ((_, d1), (_, d2)) =
            (d1, d2)

        (* The match also returns the original descriptors - remove them *)
        val cc12  = List.map removeDescriptors cc12
        val cc21  = List.map removeDescriptors cc21
        val ac12  = List.map removeDescriptors ac12
        val ac21  = List.map removeDescriptors ac21
        val ccc12 = List.map removeDescriptors ccc12
```

```
  val ccc21 = List.map removeDescriptors ccc21

  fun swap (i1, i2) = (i2, i1)

  (* Setup the expected results *)
  val cexpected12 = [(1, 60), (2, 20), (2, 40), (2, 50), (3, 50), (3, 80),
                     (4, 80), (5, 60), (5, 70), (7, 10), (8, 30)]
  val cexpected21 = List.map swap
                    [(7, 10), (2, 40), (3, 50), (2, 50), (1, 60), (5, 60),
                     (2, 20), (8, 30), (5, 70), (3, 80), (4, 80)]

  val aexpected12 = [(2, 40), (3, 50), (5, 70), (7, 10), (8, 30)]
  val aexpected21 = List.map swap [(7, 10), (2, 40), (5, 70), (3, 80), (8, 30)]

  val ccexpected12 = [(2, 40), (5, 70), (7, 10), (8, 30)]
  val ccexpected21 = List.map swap [(7, 10), (5, 70), (2, 40), (8, 30)]

in
  (* Execute the tests *)
  TestUtil.doTest
    [("length ac12", compareInt (List.length ac12) 5),
     ("length ac21", compareInt (List.length ac21) 5),
     ("ac12", compareIiList ac12 aexpected12),
     ("ac21", compareIiList ac21 aexpected21),
     ("length cc12", compareInt (List.length cc12) 11),
     ("length cc21", compareInt (List.length cc21) 11),
     ("cc12", compareIiList cc12 cexpected12),
     ("cc21", compareIiList cc21 cexpected21),
     ("length ccc12", compareInt (List.length ccc12) 4),
     ("length ccc21", compareInt (List.length ccc21) 4),
     ("ccc12", compareIiList ccc12 ccexpected12),
     ("ccc21", compareIiList ccc21 ccexpected21)
    ]
end
```

12.5 E: Log of the Execution of the Performance Evaluation

This is the output of running the implemented performance evaluation program on a Ubuntu Linux 8.04 on a 3Ghz Pentium Dual Core 2 with 4GB (i.e. plenty) of memory.

```
Detecting MSER regions (starting timer)
Detected 64 (upper) + 60 (lower) = 124 MSER regions [2.15secs]
Saving image ../outputData/images/akropolh2002_6_mserAnoq09Hysteresis_regions.bmp
Detecting MSER regions (starting timer)
Detected 263 (upper) + 338 (lower) = 601 MSER regions [1.53secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02HysteresisDelta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 6439 (upper) + 5907 (lower) = 12346 MSER regions [1.01secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02MergedDelta5_regions.bmp
Detecting MSER regions (starting timer)
Detected 8672 (upper) + 7690 (lower) = 16362 MSER regions [0.96secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02Delta5_regions.bmp
Detecting MSER regions (starting timer)
```

```
Detected 4201 (upper) + 3791 (lower) = 7992 MSER regions [0.98secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02MergedDelta10_regions.bmp
Detecting MSER regions (starting timer)
Detected 5961 (upper) + 5049 (lower) = 11010 MSER regions [1.19secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02Delta10_regions.bmp
Detecting MSER regions (starting timer)
Detected 460 (upper) + 440 (lower) = 900 MSER regions [1.09secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02MergedHalfMeanFilterDelta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 1786 (upper) + 1574 (lower) = 3360 MSER regions [1.30secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02MergedDelta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 2859 (upper) + 2268 (lower) = 5127 MSER regions [1.27secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02Delta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 174 (upper) + 98 (lower) = 272 MSER regions [1.28secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02MergedDelta50_regions.bmp
Detecting MSER regions (starting timer)
Detected 302 (upper) + 152 (lower) = 454 MSER regions [1.29secs]
Saving image ../outputData/images/akropolh2002_6_mserMata02Delta50_regions.bmp
Detecting MSER regions (starting timer)
Detected 270 (upper) + 89 (lower) = 359 MSER regions [0.31secs]
Saving image ../outputData/images/ValbonneChurch_009_mserMata02MergedDelta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 152 (upper) + 35 (lower) = 187 MSER regions [0.31secs]
Saving image ../outputData/images/ValbonneChurch_009_mserMata02MergedHalfMeanDelta20_regions.bmp
Detecting MSER regions (starting timer)
Detected 66 (upper) + 21 (lower) = 87 MSER regions [0.40secs]
Saving image ../outputData/images/bikes3_mser_regions.bmp
Detecting MSER regions (starting timer)
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.52secs]
Saving image ../outputData/images/bikes1_mser_regions.bmp
Detecting MSER regions (starting timer)
Detected 460 (upper) + 440 (lower) = 900 MSER regions [1.30secs]
Saving image ../outputData/images/akropolh2002_6_mser_regions.bmp
Detecting MSER regions (starting timer)
Detected 481 (upper) + 428 (lower) = 909 MSER regions [1.23secs]
Saving image ../outputData/images/akropolh2002_7_mser_regions.bmp
Detecting MSER regions (starting timer)
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.46secs]
Saving image ../outputData/images/boat1_mser_regions.bmp
Detecting MSER regions (starting timer)
Detected 481 (upper) + 326 (lower) = 807 MSER regions [0.37secs]
Saving image ../outputData/images/boat6_mser_regions.bmp
Detected 460 (upper) + 440 (lower) = 900 MSER regions [1.30secs]
Calculating SURF-128 descriptors took: [2.41secs]
Detected 481 (upper) + 428 (lower) = 909 MSER regions [1.20secs]
Calculating SURF-128 descriptors took: [1.82secs]
Correspondences: 15 (upper) + 29 (lower) = 44. 0 are correct. Matching: [0.14secs]
Saving image ../outputData/images/akropolh2002_6_match67.bmp
Saving image ../outputData/images/akropolh2002_7_match67.bmp
Detected 208 (upper) + 71 (lower) = 279 MSER regions [0.42secs]
Calculating SURF-128 descriptors took: [0.53secs]
```

```
Detected 218 (upper) + 86 (lower) = 304 MSER regions [0.39secs]
Calculating SURF-128 descriptors took: [0.48secs]
Correspondences: 24 (upper) + 3 (lower) = 27. 25 are correct. Matching: [0.00secs]
Saving image ../outputData/images/graf1_match12.bmp
Saving image ../outputData/images/graf2_match12.bmp
Detected 208 (upper) + 71 (lower) = 279 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [0.56secs]
Detected 265 (upper) + 69 (lower) = 334 MSER regions [0.39secs]
Calculating SURF-128 descriptors took: [0.67secs]
Correspondences: 9 (upper) + 2 (lower) = 11. 9 are correct. Matching: [0.00secs]
Saving image ../outputData/images/graf1_match13.bmp
Saving image ../outputData/images/graf3_match13.bmp
Detected 208 (upper) + 71 (lower) = 279 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [0.56secs]
Detected 211 (upper) + 108 (lower) = 319 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [0.55secs]
Correspondences: 6 (upper) + 2 (lower) = 8. 5 are correct. Matching: [0.00secs]
Saving image ../outputData/images/graf1_match14.bmp
Saving image ../outputData/images/graf4_match14.bmp
Detected 208 (upper) + 71 (lower) = 279 MSER regions [0.41secs]
Calculating SURF-128 descriptors took: [0.56secs]
Detected 237 (upper) + 117 (lower) = 354 MSER regions [0.42secs]
Calculating SURF-128 descriptors took: [0.63secs]
Correspondences: 4 (upper) + 2 (lower) = 6. 0 are correct. Matching: [0.00secs]
Saving image ../outputData/images/graf1_match15.bmp
Saving image ../outputData/images/graf5_match15.bmp
Detected 208 (upper) + 71 (lower) = 279 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [0.53secs]
Detected 259 (upper) + 123 (lower) = 382 MSER regions [0.42secs]
Calculating SURF-128 descriptors took: [0.61secs]
Correspondences: 3 (upper) + 0 (lower) = 3. 0 are correct. Matching: [0.00secs]
Saving image ../outputData/images/graf1_match16.bmp
Saving image ../outputData/images/graf6_match16.bmp
Saving image ../outputData/images/Graph_grafimages_repeatability.bmp
Saving image ../outputData/images/Graph_grafimages_locationaccuracy12.bmp
Saving image ../outputData/images/Graph_grafimages_locationaccuracy14.bmp
Saving image ../outputData/images/Graph_grafimages_recall.bmp
Saving image ../outputData/images/Graph_grafimages_oneprecision.bmp
Detected 679 (upper) + 544 (lower) = 1223 MSER regions [0.86secs]
Calculating SURF-128 descriptors took: [1.96secs]
Detected 638 (upper) + 503 (lower) = 1141 MSER regions [0.64secs]
Calculating SURF-128 descriptors took: [1.94secs]
Correspondences: 20 (upper) + 22 (lower) = 42. 16 are correct. Matching: [0.11secs]
Saving image ../outputData/images/wall1_match12.bmp
Saving image ../outputData/images/wall2_match12.bmp
Detected 679 (upper) + 544 (lower) = 1223 MSER regions [0.86secs]
Calculating SURF-128 descriptors took: [1.98secs]
Detected 639 (upper) + 501 (lower) = 1140 MSER regions [0.64secs]
Calculating SURF-128 descriptors took: [1.76secs]
Correspondences: 29 (upper) + 20 (lower) = 49. 14 are correct. Matching: [0.10secs]
Saving image ../outputData/images/wall1_match13.bmp
Saving image ../outputData/images/wall3_match13.bmp
Detected 679 (upper) + 544 (lower) = 1223 MSER regions [0.86secs]
```

```
Calculating SURF-128 descriptors took: [1.98secs]
Detected 636 (upper) + 514 (lower) = 1150 MSER regions [0.65secs]
Calculating SURF-128 descriptors took: [1.72secs]
Correspondences: 30 (upper) + 27 (lower) = 57. 6 are correct. Matching: [0.10secs]
Saving image ../outputData/images/wall1_match14.bmp
Saving image ../outputData/images/wall4_match14.bmp
Detected 679 (upper) + 544 (lower) = 1223 MSER regions [0.85secs]
Calculating SURF-128 descriptors took: [1.98secs]
Detected 691 (upper) + 588 (lower) = 1279 MSER regions [0.61secs]
Calculating SURF-128 descriptors took: [1.95secs]
Correspondences: 21 (upper) + 16 (lower) = 37. 2 are correct. Matching: [0.12secs]
Saving image ../outputData/images/wall1_match15.bmp
Saving image ../outputData/images/wall5_match15.bmp
Detected 679 (upper) + 544 (lower) = 1223 MSER regions [0.90secs]
Calculating SURF-128 descriptors took: [2.05secs]
Detected 681 (upper) + 594 (lower) = 1275 MSER regions [0.62secs]
Calculating SURF-128 descriptors took: [1.92secs]
Correspondences: 22 (upper) + 9 (lower) = 31. 1 are correct. Matching: [0.11secs]
Saving image ../outputData/images/wall1_match16.bmp
Saving image ../outputData/images/wall6_match16.bmp
Saving image ../outputData/images/Graph_wallimages_repeatability.bmp
Saving image ../outputData/images/Graph_wallimages_locationaccuracy12.bmp
Saving image ../outputData/images/Graph_wallimages_recall.bmp
Saving image ../outputData/images/Graph_wallimages_oneprecision.bmp
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.42secs]
Calculating SURF-128 descriptors took: [2.33secs]
Detected 629 (upper) + 369 (lower) = 998 MSER regions [0.43secs]
Calculating SURF-128 descriptors took: [2.55secs]
Correspondences: 60 (upper) + 38 (lower) = 98. 80 are correct. Matching: [0.08secs]
Saving image ../outputData/images/boat1_match12.bmp
Saving image ../outputData/images/boat2_match12.bmp
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.39secs]
Calculating SURF-128 descriptors took: [2.34secs]
Detected 512 (upper) + 338 (lower) = 850 MSER regions [0.52secs]
Calculating SURF-128 descriptors took: [2.38secs]
Correspondences: 56 (upper) + 30 (lower) = 86. 59 are correct. Matching: [0.07secs]
Saving image ../outputData/images/boat1_match13.bmp
Saving image ../outputData/images/boat3_match13.bmp
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [2.35secs]
Detected 366 (upper) + 201 (lower) = 567 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [1.66secs]
Correspondences: 23 (upper) + 20 (lower) = 43. 9 are correct. Matching: [0.05secs]
Saving image ../outputData/images/boat1_match14.bmp
Saving image ../outputData/images/boat4_match14.bmp
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [2.38secs]
Detected 276 (upper) + 147 (lower) = 423 MSER regions [0.48secs]
Calculating SURF-128 descriptors took: [1.28secs]
Correspondences: 18 (upper) + 24 (lower) = 42. 8 are correct. Matching: [0.03secs]
Saving image ../outputData/images/boat1_match15.bmp
Saving image ../outputData/images/boat5_match15.bmp
Detected 534 (upper) + 373 (lower) = 907 MSER regions [0.40secs]
```

```
Calculating SURF-128 descriptors took: [2.30secs]
Detected 481 (upper) + 326 (lower) = 807 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [2.45secs]
Correspondences: 18 (upper) + 12 (lower) = 30. 0 are correct. Matching: [0.07secs]
Saving image ../outputData/images/boat1_match16.bmp
Saving image ../outputData/images/boat6_match16.bmp
Saving image ../outputData/images/Graph_boatimages_repeatability.bmp
Saving image ../outputData/images/Graph_boatimages_locationaccuracy12.bmp
Saving image ../outputData/images/Graph_boatimages_locationaccuracy14.bmp
Saving image ../outputData/images/Graph_boatimages_recall.bmp
Saving image ../outputData/images/Graph_boatimages_oneprecision.bmp
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.60secs]
Calculating SURF-128 descriptors took: [1.10secs]
Detected 150 (upper) + 25 (lower) = 175 MSER regions [0.43secs]
Calculating SURF-128 descriptors took: [0.66secs]
Correspondences: 31 (upper) + 15 (lower) = 46. 36 are correct. Matching: [0.00secs]
Saving image ../outputData/images/bikes1_match12.bmp
Saving image ../outputData/images/bikes2_match12.bmp
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.56secs]
Calculating SURF-128 descriptors took: [1.04secs]
Detected 66 (upper) + 21 (lower) = 87 MSER regions [0.39secs]
Calculating SURF-128 descriptors took: [0.46secs]
Correspondences: 18 (upper) + 11 (lower) = 29. 22 are correct. Matching: [0.00secs]
Saving image ../outputData/images/bikes1_match13.bmp
Saving image ../outputData/images/bikes3_match13.bmp
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.60secs]
Calculating SURF-128 descriptors took: [1.10secs]
Detected 70 (upper) + 17 (lower) = 87 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [0.47secs]
Correspondences: 14 (upper) + 7 (lower) = 21. 12 are correct. Matching: [0.00secs]
Saving image ../outputData/images/bikes1_match14.bmp
Saving image ../outputData/images/bikes4_match14.bmp
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.53secs]
Calculating SURF-128 descriptors took: [1.07secs]
Detected 61 (upper) + 17 (lower) = 78 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [0.41secs]
Correspondences: 14 (upper) + 5 (lower) = 19. 13 are correct. Matching: [0.00secs]
Saving image ../outputData/images/bikes1_match15.bmp
Saving image ../outputData/images/bikes5_match15.bmp
Detected 239 (upper) + 84 (lower) = 323 MSER regions [0.56secs]
Calculating SURF-128 descriptors took: [1.04secs]
Detected 41 (upper) + 16 (lower) = 57 MSER regions [0.36secs]
Calculating SURF-128 descriptors took: [0.30secs]
Correspondences: 14 (upper) + 6 (lower) = 20. 6 are correct. Matching: [0.00secs]
Saving image ../outputData/images/bikes1_match16.bmp
Saving image ../outputData/images/bikes6_match16.bmp
Saving image ../outputData/images/Graph_bikesimages_repeatability.bmp
Saving image ../outputData/images/Graph_bikesimages_locationaccuracy12.bmp
Saving image ../outputData/images/Graph_bikesimages_locationaccuracy14.bmp
Saving image ../outputData/images/Graph_bikesimages_recall.bmp
Saving image ../outputData/images/Graph_bikesimages_oneprecision.bmp
Detected 155 (upper) + 141 (lower) = 296 MSER regions [0.45secs]
Calculating SURF-128 descriptors took: [0.46secs]
```

```
Detected 155 (upper) + 103 (lower) = 258 MSER regions [0.40secs]
Calculating SURF-128 descriptors took: [0.36secs]
Correspondences: 32 (upper) + 8 (lower) = 40. 35 are correct. Matching: [0.00secs]
Saving image ../outputData/images/leuven1_match12.bmp
Saving image ../outputData/images/leuven2_match12.bmp
Detected 155 (upper) + 141 (lower) = 296 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [0.46secs]
Detected 128 (upper) + 72 (lower) = 200 MSER regions [0.44secs]
Calculating SURF-128 descriptors took: [0.23secs]
Correspondences: 19 (upper) + 10 (lower) = 29. 22 are correct. Matching: [0.00secs]
Saving image ../outputData/images/leuven1_match13.bmp
Saving image ../outputData/images/leuven3_match13.bmp
Detected 155 (upper) + 141 (lower) = 296 MSER regions [0.43secs]
Calculating SURF-128 descriptors took: [0.46secs]
Detected 124 (upper) + 53 (lower) = 177 MSER regions [0.41secs]
Calculating SURF-128 descriptors took: [0.19secs]
Correspondences: 10 (upper) + 7 (lower) = 17. 11 are correct. Matching: [0.00secs]
Saving image ../outputData/images/leuven1_match14.bmp
Saving image ../outputData/images/leuven4_match14.bmp
Detected 155 (upper) + 141 (lower) = 296 MSER regions [0.47secs]
Calculating SURF-128 descriptors took: [0.45secs]
Detected 141 (upper) + 39 (lower) = 180 MSER regions [0.34secs]
Calculating SURF-128 descriptors took: [0.23secs]
Correspondences: 7 (upper) + 8 (lower) = 15. 7 are correct. Matching: [0.00secs]
Saving image ../outputData/images/leuven1_match15.bmp
Saving image ../outputData/images/leuven5_match15.bmp
Detected 155 (upper) + 141 (lower) = 296 MSER regions [0.46secs]
Calculating SURF-128 descriptors took: [0.43secs]
Detected 95 (upper) + 30 (lower) = 125 MSER regions [0.32secs]
Calculating SURF-128 descriptors took: [0.10secs]
Correspondences: 8 (upper) + 10 (lower) = 18. 4 are correct. Matching: [0.00secs]
Saving image ../outputData/images/leuven1_match16.bmp
Saving image ../outputData/images/leuven6_match16.bmp
Saving image ../outputData/images/Graph_leuvenimages_repeatability.bmp
Saving image ../outputData/images/Graph_leuvenimages_locationaccuracy12.bmp
Saving image ../outputData/images/Graph_leuvenimages_recall.bmp
Saving image ../outputData/images/Graph_leuvenimages_oneprecision.bmp
```

References

[AMLB09] Online reference for the Ánoq SML Basis Library (0.8.2), by Ánoq of the Sun, 2009-05-18: `http://www.hardcoreprocessing.com/pro/anoqsmlbasis/`

[Brow02] M. Brown, D. Lowe. "Invariant Features from Interest Point Groups". In *BMVC*. 2002.

[Brow04] M. Brown, R. Szeliski and S. Winder. "Multi-image Matching using Multi-scale Oriented Patches". In *IEEE Computer Society Conference of Computer Vision and Pattern Recognition (CVPR 2005), (San Diego, CA), Volume I*, p. 510-517. June 2005.

[BayH06] Herbert Bay, Tinne Tuytelaars, Luc Van Gool. "SURF: Speeded Up Robust Features". 2006.

[Case08] Vincent Caselles and Pascal Monasse. *Geometric Description of Topographic Maps and Applications to Image Processing (preprint)*. September 1, 2008.

[Chum05] O. Chum and J. Matas. "Matching with PROSAC - PROgressive SAmple Consensus". In *IEEE Computer Society Conference of Computer Vision and Pattern Recognition (CVPR 2005), (San Diego, CA), Volume I*, p. 220-226. June 2005.

[Corm90] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. *An Introduction to Algorithms*. The MIT Press, 1990.

[Crow84] Franklin C. Crow. "Summed-Area Tables for Texture Mapping". In *SIGGRAPH Conference Proceedings of Computer Graphics Volume 18, 3, July 1984*. ACM, 1984.

[Cyga09] Boguslaw Cyganek, J. Paul Siebert. *An Introduction to 3D Computer Vision Techniques and Algorithms*. John Wiley & Sons Ltd, 2009.

[Dufo02] Yves Dufournaud, Cordelia Schmid and Radu Horaud, *Image Matching with Scale Adjustment*. Scientific report, INRIA, 2002.

[Fole96] Foley, van Dam, Feiner, Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley 1990-1996.

[Harr88] C. Harris and M. Stephens. "A Combined Corner and Edge Detector". In *Alvey Vision Conference*, pages 147-151. 1998.

[Hart03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*, Second Edition. Cambridge University Press, 2003.

[HystWi] Explanation of Hysteresis on Wikipedia, as seen on the 10th of September 2009: `http://en.wikipedia.org/wiki/Hysteresis`

[Koen84] J. Koenderink. "The Structure of Images". In *Biological Cybernetics 50 (1984)*, p. 363-370. 1984.

[Lowe04] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In *International Journal of Computer Vision, 2004*. January 5, 2004.

[Luca81] B. Lucas and T. Kanade. "An Interative Image Registration Technique with an Application to Stereo Vision". In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674-679, 1981.

[Mata02] J. Matas, O. Chum, M. Urban, T. Pajdla. "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions". In *Proc. British Machine Vision Conference BMVC2002*, 2002.

[Miko02] Krystian Mikolajczyk. *Detection of Local Features Invariant to Affine Transformations*. Ph.D. thesis, Institut National Polytechnique de Grenoble, France, 2002.

[Miko04] Krystian Mikolajczyk, Cordelia Schmid. "Scale & Affine Invariant Interest Point Detectors". In *International Journal of Computer Vision 60(1)*, pages 61-83. Kluwer Academic Publishers, 2004.

[Miko05] Krystian Mikolajczyk, Cordelia Schmid. "A Performance Evaluation of Local Descriptors". In *Ieee Transactions on Pattern Analysis and Machine Intelligence, volume 27, No. 10*, p. 1615-1630. October 2005.

[Miko06] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, L. Van Gool. "A Comparison of Affine Region Detectors". In *International Journal of Computer Vision*. Springer Science + Business Media Inc. 2006.

[Moll99] Tomas Möller, Eric Haines. *Real-Time Rendering*. A. K. Peters 1999.

[Mona99] Pascal Monasse. "Contrast Invariant Registration of Images". In *Proc. of Int. Conf. on Acoustics, Speech and Signal Processing* p. 3221-3224, Phoenix Arizona, 1999.

[Mona00] Pascal Monasse, Frédéric Giuchard. "Fast Computation of a Contrast-invariant Image Representation". In *IEEE Transactions on Image Processing* p. 860-872, Volume 9, Issue 5, 2000.

[MLto07] Online reference for the MLton compiler, by Stephen Weeks et al, version 20070826: `http://www.mlton.org`

[Neub06] Alexander Neubeck and Luc Van Gool. "Efficient Non-Maximum Suppression". In *18th International Conference on Pattern Recognition (ICPR) 2006*, volume 3, pages 850-855. 2006.

[Oshe06] Stanley Osher, Nikos Paragios (Editors). *Geometric Level Set Methods in Imaging, Vision and Graphics*. Springer Science+Business Media LLC, 2006.

[Poll00] Marc Pollefeys. *SIGGRAPH 2000 Course Notes 12: Obtaining 3D Models With a Hand-Held Camera*. ACM SIGGRAPH, 2000.

[Prit03] D. Pritchard and W. Heidrich. "Cloth Motion Capture". In *Computer Graphics Forum (Eurographics 2003), 22(3)*, pages 263-271. 2003.

[SMLB04] Online reference for the Standard ML Basis Library (2004 edition), by John Reppy et al: `http://www.standardml.org/Basis/`

[Szel06] Richard Szeliski. *Image Alignment and Stitching: A Tutorial. Foundations and Trends in Computer Graphics and Vision 2:1 (2006)*. now Publishers Inc., 2006.

[Tuyt99] Tinne Tuytelaars, Luc Van Gool. "Content-based Image Retrieval based on Local Affinely Invariant Regions". In *Proc. Third Int'l Conf. on Visual Information Systems*, pages 493-500, 1999.

[Tuyt00] Tinne Tuytelaars, Luc Van Gool. "Wide Baseline Stereo Matching based on Local, Affinely Invariant Regions". In M. Mirmehdi and B. Thomas, editors, *Proc. British Machine Vision Conference BMVC2000*, pages 412-422, London UK, 2000.

[Watt92] Alan Watt, Mark Watt. *Advanced Animation and Rendering Techniques Theory and Practice*. Addison-Wesley 1992.