Study group
SURF: Feature detection & description

Jacob Toft Pedersen 19983275 jtp@cs.au.dk

Abstract—A technical report on Feature detection and implementing the Speeded-Up Robust Features(SURF) algorithm.

Index Terms—SURF, SIFT, feature detection, feature description, integral image

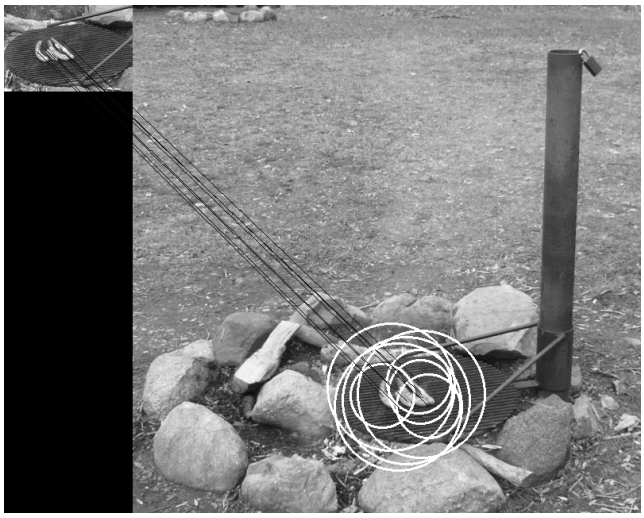

Fig. 1. Detecting common features between two images with different scale and cropping.

1 Introduction

Feature detection is the process where we automatically examine an image to extract features, that are unique to the objects in the image, in such a manner that we are able to detect an object based on its features in different images. This detection should ideally be possible when the image shows the object with different transformations, mainly scale and rotation, or when parts of the object are occluded.

The processes can be divided in to 3 overall steps.

Detection Automatically identify interesting features, *interest points* this must be done robustly. The same feature should always be detected irregardless of viewpoint.

Description Each interest point should have a unique description that does not depend on the features scale and rotation.

Matching Given and input image, determine which objects it contains, and possibly a transformation of the object, based on predetermined interest points.

This report will focus on the details of the first two steps with the SURF algorithm [1].

2 Detection

Scale-Invariant Feature Transform, *SIFT* is a successful approach to feature detection introduced by Lowe [1]. The SURF-algorithm [2] is based on the same principles and steps, but it utilizes a different scheme and it should provide better results, faster.

In order to detect feature points in a scale-invariant manner SIFT uses a cascading filtering approach. Where the Difference of Gaussians, *DoG*, is calculated on progressively downscaled images.

In general the technique to achieve scale invariance is to examine the image at different scales, *scale space*, using Gaussian kernels. Both SIFT and SURF divides the scale space into levels and octaves. An octave corresponds to a doubling of σ, and the the octave is divided into uniformly spaced levels.

1. Speeded-Up Robust Features

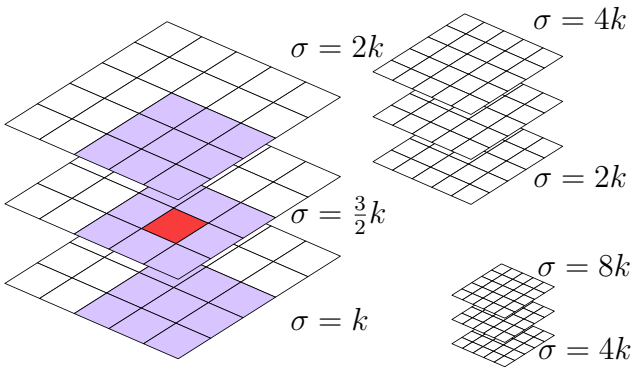

Fig. 2. 3 octaves with 3 levels, The neighborhood for the $3\times3\times3$ non-maximum suppression used to detect features is highlighted.

Both approaches builds a pyramid of response maps, with different levels within octaves. A response map is the result of an operation on the image. The interest points are the points that are the extrema among 8 neighbors in the current level and its 2×9 neighbors in the level below and above. This is a non-maximum suppression in a $3\times3\times3$ neighborhood, the relation between levels, octaves and neighborhood is illustrated in Figure 2.

2.1 Hessian matrix interest points

SURF uses a hessian based blob detector to find interest points. The determinant of a hessian matrix expresses the extent of the response and is an expression of the local change around the area [3].

$$\mathcal{H}(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix} \quad (1)$$

where

$$L_{xx}(\mathbf{x},\sigma) = I(\mathbf{x}) * \frac{\partial^2}{\partial x^2} g(\sigma) \quad (2)$$

$$L_{xy}(\mathbf{x},\sigma) = I(\mathbf{x}) * \frac{\partial^2}{\partial xy} g(\sigma) \quad (3)$$

$L_{xx}(\mathbf{x},\sigma)$ in equation 2 is the convolution of the image with the second derivative of the Gaussian. The heart of the SURF detection is non-maximal-suppression of the determinants of the hessian matrices. The convolutions is very costly to calculate and it is approximated and speeded-up with the use of integral images and approximated kernels.

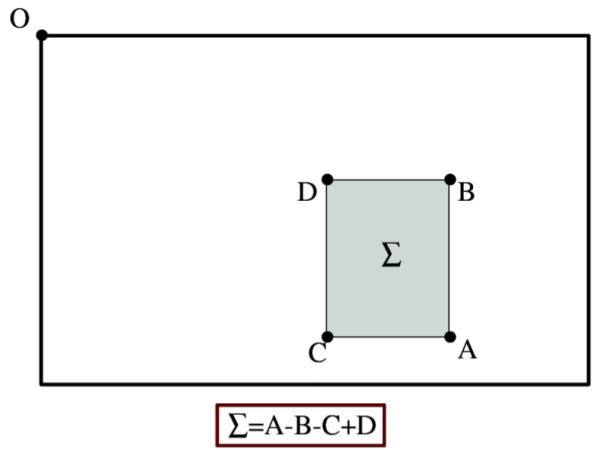

Fig. 3. 4 memory look ups is sufficient to calculate the sum of an rectangular area with an integral image

An Integral image[2] $I(\mathbf{x})$ is an image where each point $\mathbf{x} = (x,y)^\mathsf{T}$ stores the sum of all pixels in a rectangular area between origo and $\mathbf{x}$ (See equation 4).

$$I(\mathbf{x}) = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} I(x,y) \quad (4)$$

The use of integral images enables calculating the response in a rectangular area with arbitrary size using 4 look-ups as illustrated in Figure 3.

The second order Gaussian kernels $\frac{\partial^2}{\partial y^2}g(\sigma)$ used for the hessian matrix must be discretized and cropped before we can apply them, a 9×9 kernel is illustrated in Figure 4. The SURF algorithm approximates these kernels with rectangular boxes, box filters. In the illustration grey areas corresponds to 0 in the kernel where as white are positive and black are negative. This way it is possible to calculate the approximated convolution effectively for arbitrarily sized kernel utilizing the integral image.

$$Det(\mathcal{H}_{approx}) = D_{xx}D_{xy} - (wD_{xy})^2 \quad (5)$$

The approximated and discrete kernels are referred to as D_{yy} for $L_{yy}(\mathbf{x},\sigma)$ and D_{xy} for

2. It is also another name for SAT, summed area tables

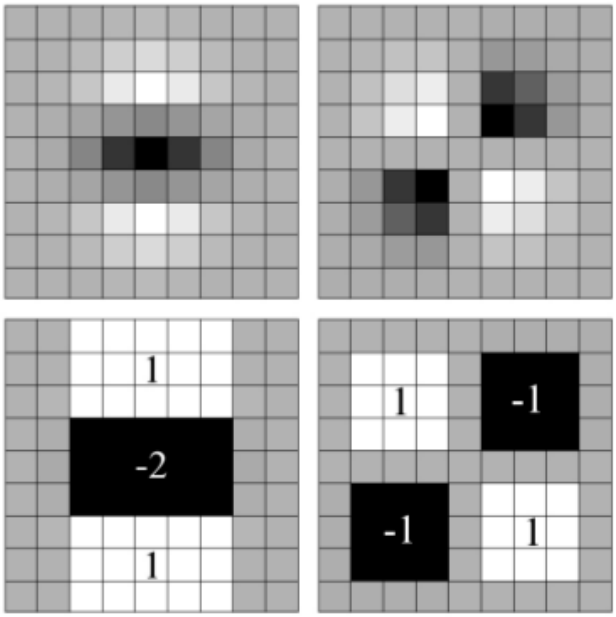

Fig. 4. $L_{yy}(\mathbf{x}, \sigma)$ and $L_{xy}(\mathbf{x}, \sigma)$ Discretized Gaussians and the approximations D_{yy} and D_{xy}

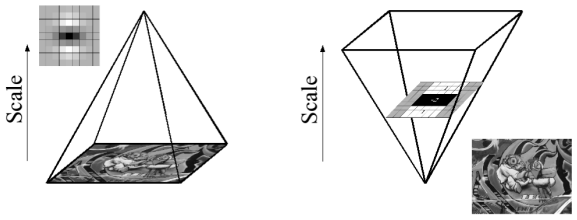

Fig. 5. Where SIFT(left) downscales the image, SURF(right) uses larger and larger filters

$L_{xy}(\mathbf{x}, \sigma)$. The illustrated kernels corresponds to a σ of 1.2 and are the lowest scale that the SURF algorithm can handle. When using the approximated kernels to calculate the determinant of the Hessian matrix - we have to weight it with w in equation 5, this is to assure the energy conservation for the Gaussians. The w term is theoretically sensitive to scale but it can be kept constant at 0.9 [2].

To detect features across scale we have to examine several octaves and levels, where SIFT scales the image down for each octave and use progressively larger Gaussian kernels, the integral images allows the SURF algorithm to calculate the responses with arbitrary large kernels(Figure 5).

.

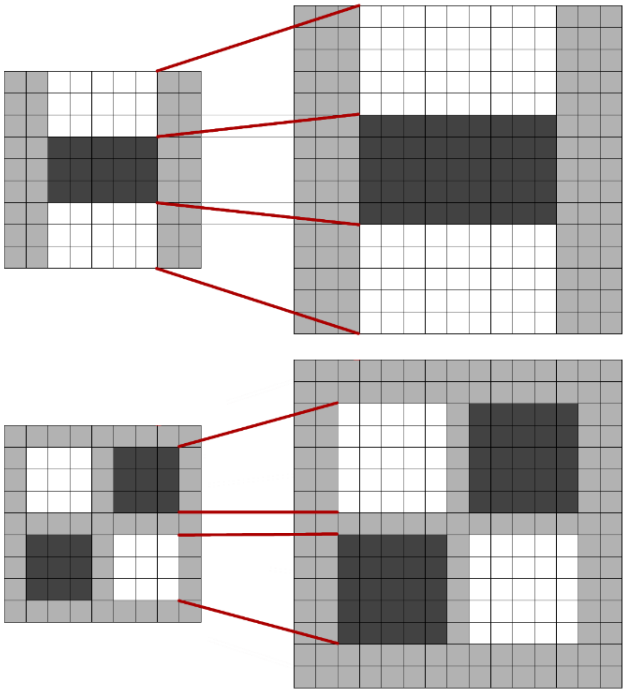

Fig. 6. Increasing the size of kernels, and keeping the lobes correctly scaled

This does pose two challenges, how to scale the approximated kernels and how this influences the possible values for σ. The kernels has to have an uneven size to have a central pixel and the rectangular areas, the *lobes*, has to have the same size(Figure 6).

The SURF paper [2] goes into detail with these considerations. The result is that division of scale space into levels and octaves becomes fixed as illustrated in Figure 7. The filter size will be large and if the convolution were to be done with a regular Gaussian kernel this would be prohibitively expensive. The use of integral images not only makes this feasible - it also does it fast, and without the need to downscale the image. It should be noted that this approach with large box filters can preserve and be sensitive to high frequency noise.

When finding a extrema at one of the higher octaves the area covered by the filter is rather large and this introduces a significant error for the position of the interest point. To remedy this the exact location of the interest point are interpolated by fitting a 3D quadratic in scale space [4]. An interest point is located in

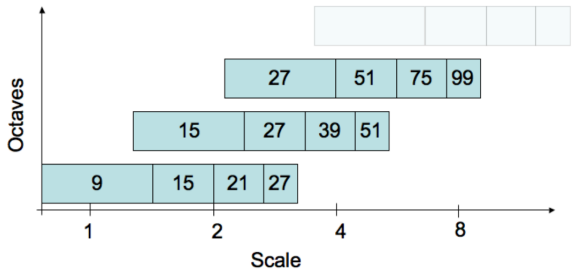

Fig. 7. The scale (σ) the filter sizes and octaves on a logarithmic scale

scale space by (x, y, s) where x, y are relative coordinates($x, y \in [0; 1]$) and s is the scale.

3 DESCRIPTION

The purpose of a descriptor is to provide a unique and robust description of a feature, a descriptor can be generated based on the area surrounding a interest point. The SURF descriptor is based on Haar wavelet responses and can be calculated efficiently with integral images. SIFT uses another scheme for descriptors based on the Hough transform. Common to both schemes is the need to determine the orientation. By determining a unique orientation for a interest point, it is possible to achieve rotational invariance. Before the descriptor is calculated the *interest area* surrounding the interest point are rotated to its direction.

The SURF descriptors are robust to rotations and an upright version, U-SURF, should be robust for rotations $\pm 15°$, without performing an orientation assignment [2]. I have implemented the upright version, and will not go into further detail on orientation assignment.

The SURF descriptor describes an interest area with size $20s$. The interest area is divided into 4×4 subareas that is described by the values of a wavelet response in the x and y directions. The wavelet response in the x and y direction is refered to as dx and dy respectively, the wavelets used to calculate the response is illustrated in Figure 9. The interest area are weighted with a Gaussian centered at the interest point to give some robustness for deformations and translations. The components involved in the calculations is illustrated in Figure 8.

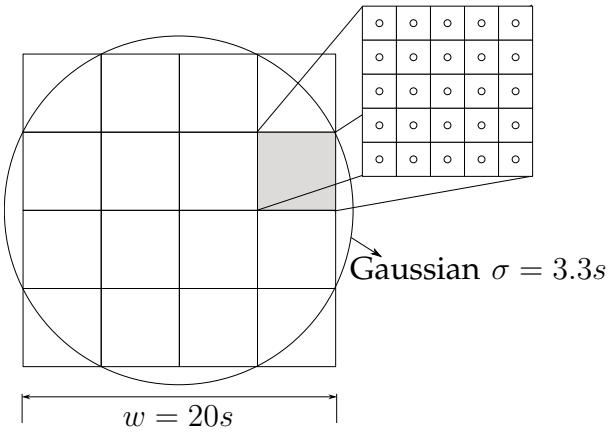

Fig. 8. A $20s$ areas is divided into 4×4 subareas that are sampled 5×5 times to get the wavelet response(Figure 9)

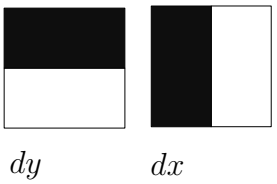

Fig. 9. The wavelets response. Black and white areas corresponds to a weight -1 and 1 for the Haar kernels. They are used with a filter size of $2s$

$$v = \{\sum dx, \sum |dx|, \sum dy, \sum |dy|\} \qquad (6)$$

For each subarea a vector v (Equation 6) is calculated, based on 5×5 samples. The descriptor for a interest point is the 16 vectors for the subareas concatenated. Finally the descriptor is normalized, to achieve invariance to contrast variations that will represent themselves as a linear scaling of the descriptor.

Several schemes varying the size, number of samples and wavelet function has been tested. This setup has been experimentally found as optimal, taking performance and precision into account [2].

Having calculated the descriptors finding a match between two descriptors is a matter of testing if the distance between the two vectors is sufficiently small. The SURF algorithm does add another detail to speed up matching, that

is the sign of Laplacian.

$$\nabla^2 L = tr(\mathcal{H}) = L_{xx}(\mathbf{x}, \sigma) + L_{yy}(\mathbf{x}, \sigma) \quad (7)$$

The laplacian is the trace of the hessian matrix (Equation 7) and when calculating the determinant of the hessian matrix these values are available. It is a matter of storing the sign. The reason to store the sign of the Laplacian is that distinguishes between bright blobs on dark backgrounds and vice versa. It is only necessary to compare the full descriptor vectors if the have the same sign, which can lower the computational cost of matching.

4 Implementation

In the previous sections I have outlined the theoretical background for the SURF algorithm, in this section I will go into details with the implementation I have made.

I have used the `openCV` framework, this is a `C++` framework for computer vison. It ships with its own implementation of SURF and SIFT and several other computer vision algorithms. It was chosen as it provides a good low level rutines for working with images and easy loading and saving of different image formats.

When implementing i have studied both an imageJ plugin[3] for SURF and openSURF [4] for comparison and inspiration. Evans that initiated openSURF has published an technical article detailing the individual steps in the algorithm [5].

The focus when implementing has been on getting to a state where it was possible to start automated testing, such that the effects of additional refinements could be examined. It has not been a priority to optimize for speed or memory usage.

4.1 Detection

The example image used throughout the report is the barbecue setting (Figure 1), but during implementation testing other images have been used to verify the results[5].

3. www.labun.com/imagej-surf

4. http://www.chrisevansdev.com/

5. All images have been processed as 64bit greyscale floating point, as the input has been normal 3 channel color images this is overkill, but working with double precision arithmetic reduces precision errors.

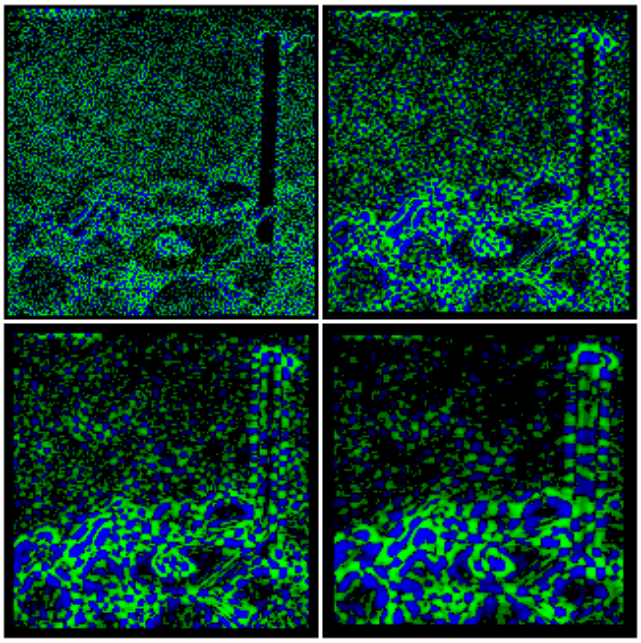

Fig. 10. The hessian determinant ($Det(\mathcal{H}_{approx})$) response maps for the 2'th octave. Positive values are green and negative values blue

The first step is to create an integral image, this is done using an built-in `opencv` routine. Then the response maps for 4 levels in 4 octaves are created. The size of the filters are fixed and when calculating the response maps the exact size of the *lobes* and their offsets are calculated based on the filter size(see Figure 6).

boxfilters are used to find D_{xx}, D_{yy} and D_{xy} and then calculate the approximated Hessians and Laplacian. Before using $D_{??}$ they are normalized - such that the responses does not depend on the filter size.

The dimension of the response maps are equal within an octave, and is halved when going up an octave, this is implemented by increasing the distance between samples.

When implementing this part it was critical to have some meaningful debugging output. Figure 10 and 11 shows a montage of some of this debugging output[6]. This type of output coupled with simple test images proved to be a valuable tool to inspect the behavior of the detector.

After having built the response maps the

6. Images using the green/blue coloring have been preprocessed with histogram equalization, as this presents the data better for visual inspection.

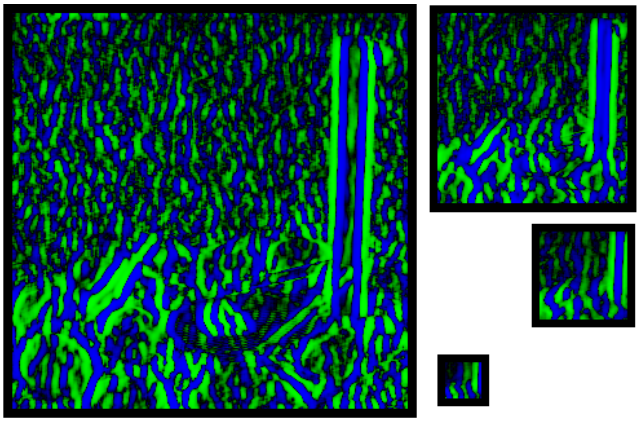

Fig. 11. The D_{xx} response map for the largest filter in each octave. The images are to scale, it can be observed that not only is the images smaller also the filters are larger, thus removing more noise

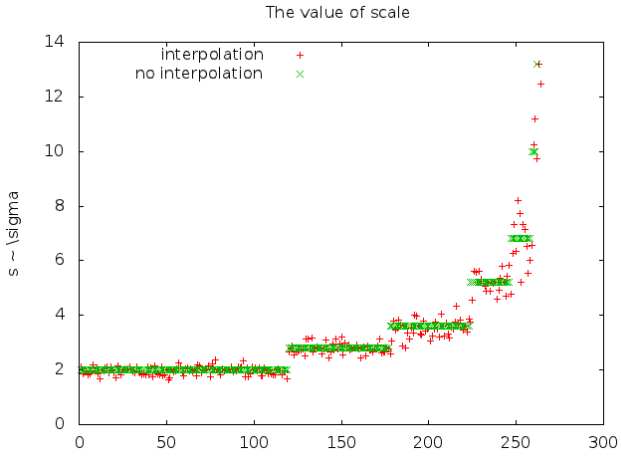

Fig. 12. Interest point and at which scale s they are detected

next step is to do a non-maximum suppression on the hessian determinant values ($Det(\mathcal{H}_{approx})$). Neubeck et al [6] provides practical considerations for an efficient algorithm, which weighs the theoretical bounds against real life access times. Time did not permit optimizing this part. The neighborhood for non-maximal suppression is illustrated in Figure 2. To filter out noise only are thresholded before considering them.

Having identified a maximum, the next step is to interpolate the position in scale space using the method from Lowe [4], Evans [5] provides good notes on the details of this step. In essence we have to fit a 3D quadratic expressing the Hessian as a function $H(x, y, \sigma)$ byusing a Taylor expansion(8) for finding the extrema by setting the derivative to zero and solving the equation(9) to find $\hat{\mathbf{x}} = (x, y, \sigma)$

$$H(\mathbf{x}) = H + \frac{\partial H^{\mathsf{T}}}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathsf{T}}\frac{\partial^2 H^{\mathsf{T}}}{\partial \mathbf{x}^2}\mathbf{x} \tag{8}$$

$$\hat{\mathbf{x}} = \frac{\partial^2 H^{-1}}{\partial^2 \mathbf{x}}\frac{\partial H}{\partial \mathbf{x}} \tag{9}$$

The derivatives are calculated from finite differences in the response maps. And the inverse is found using an `opencv` function with an SVD-decomposition. This step proved to be an exercise in rigor and indexing discipline. It was implemented such that it is possible to compile with or without interpolation, Figure 12 shows the scales at which interest points are found.

Unfortunately the results are consistently better without interpolation. The interpolation is the result of fitting a 3D quadratic in scale space and the interpolation itself might give wrong results, that is results that are unreasonable far from the initial point. When studying the source for openSURF and the imageJ plugin it was obvious that the exact criteria for an bad result where up to interpretation. The result from the interpolation is relative offsets from the original point, and to weed the bad results out, interpolation results with any component > 0.5 discards the offset. [7]

4.2 Description

FiXme Note: better figure?

In the previous section I have described how to identify interest points, this section will go into detail about describing these points. The overall and theoretical background is in Section 3.

I have implemented the U-SURF variant, that is i do not assign an orientation to the interest point. This was chosen as U-SURF should be fairly rotational invariant and orientation assignment could be implemented at later stage to improve the performance.

7. This is similar to the same tactic used in openSURF, the imageJ plugin checks whether the point is still inside *the scale space pyramid*.

Fig. 13. The interest points detected in the barbecue setting

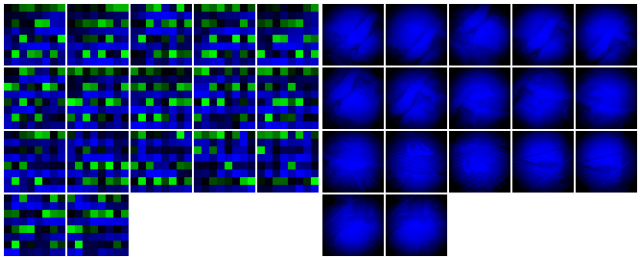

Fig. 14. An image representation of the descriptors, together with the areas described. The areas are weighted by a Gaussian as described in section 3.

To calculate the descriptors, each of the 16 subareas surrounding the interest point are sampled to get the wavelet response dx and dy together with their absolute values using the scheme illustrated in Figure 8. The wavelet response is calculated using the integral image with box filters with filter size $2s$. For the case dx this is the value of square to the right minus the value of a square to the left of the interest point(See Figure 9). This response are then weighted by a Gaussian ($\sigma = 3.3s$) centered at the interest point.

The vector consisting of the concatenated values of all 64 descriptors are normalized as a last step. The interest points can be saved to a basic ASCII list where precision does matter. It

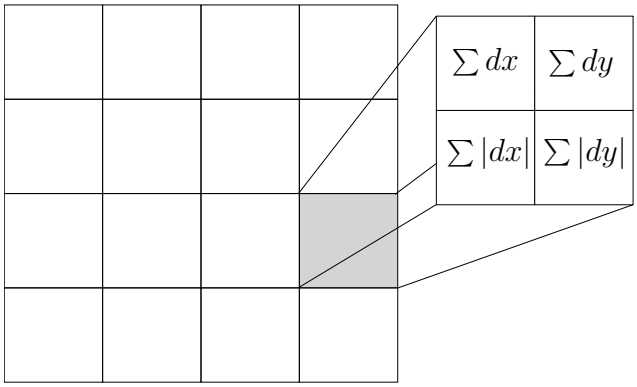

Fig. 15. The layout of the image representation of an desciptor (Figure 14)

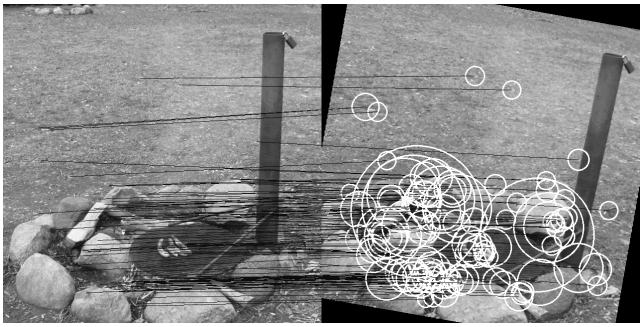

Fig. 16. Matching with a 10° rotated version. The circles has a diameter of $10s$

was interesting to note that with the default precision saving and loading a list and testing against the same image would not give a 100% match. Figure 14 shows the area sample together with the descriptors according to the layout in Figure 15.

The program together with instructions on compiling and running i is available in Appendix A.

4.3 Futher work

Focus and time has been put at getting a good quality of the results. There is still things that could be optimized in this regard, just as there are some optimisations that could be implemented that would increase performance. At the moment the structure of the code is such that the response maps for each octave are build in isolation, as can be seen in Figure 6 several filter sizes are used across octaves, reusing the results should be possible. Further more the calculation of the response maps may

be done in parallel as suggested by Gossow et al [7] and the non-maximal suppression could also be done in parallel. In general there is a potential to get a considerable performance gain by using GPU computations[8].

5 RESULTS

It was very satisfying when the program showed the first matching of interest points as illustrated in Figure 1, which incidentally is the same test data used for the first match. From there the next step were to setup automated testing, enabling testing and measuring how changes and additions affected the quality.

5.1 Testing

When testing the algorithm there are several key factors to take into account. Gossow et al [7] has published a report comparing several open source Implementations of the SURF algorithm, they introduce 3 different performance criteria. *Repeatability*, *precision* and *recall*. Repeatability is the how good the detector is at detecting the same features under different transformations of the images. As some transformations, for example an rotation, may move interest points out of the image it must take this into account. Thus repeatability is defined as = $\frac{\#correspondences}{min(n_1,n_2)}$, where $n_{1,2}$ is the number of interest point in the images to be compared and correspondence whether the points are close to the correct result. [9].

Precision measures how many correct matches are found, and *recall* is the relation between correct matches and correspondences. Gossow et al uses a fixed pool of images and they have calculated the transformation between them, so they are able to calculate correspondence and verify if a match is correct

As i have implemented the U-SURF variant I have mainly focused the testing invariance to scale. This also removed the need to calculate the position of interest points after the transformations, as the position is stored as relative coordinates, and correspondence is linear in with respect to the distance between the match and the original point, when there is no skewing involved.

5.2 Setup

The tests have been carried out on a Intel core i5 processor with 8GB ram running a 64 bit Ubuntu Linux *natty narwhale*, using the `GCC` compiler version 4.5. with optimisation level 2 and `openCV` version 2.2.

I created a script the takes as input a single image and rotates and scales this image[10], and then tries to match interest points from the original image to the transformed versions(Instructions in Appendix A). It is a synthetic case, where the actual implementation of the scaling could influence the results. It is on the other hand a reasonable test setup and it is easy to rerun the test to investigate the effects of different parameters. In the future a setup where different images are tested automatically would be beneficial as the image tested does influence how many interest points are found.

The statistics gathered are the number of interest points found and how they match interest points from the original image. Matching is performed based on simple thresholding of the distance between the descriptor vectors. To investigate this threshold the minimal distance is found and the values are put into bins of size 0.1, a interest point are placed into all bins when the distance is below the bins threshold, to create histograms.

Furthermore the interest points are verified, the distance between the relative coordinates are compared[11], and this is considered as correct matches which can be used to determine the precision of the implementation. Since the verification only looks a relative coordinates it is not meaningful for rotations.

The SURF paper [2] states that the U-SURF is robust $\pm 15^\circ$ when looking at Figure 17 this would suggest that a threshold of 0.4 or 0.5

8. Debugging GPU algorithms is notoriously difficult, and as focus has been on quality this was not considered a reasonable starting point for this study group. There exists an open source GPU implementation http://www.d2.mpi-inf.mpg.de/surf.

9. Correspondence is defined based on the overlapping area between interest regions [7]

10. using imagemagicks `convert` utility.

11. I use a basic threshold .01, ideally it should have taken scale s and the image dimensions into account

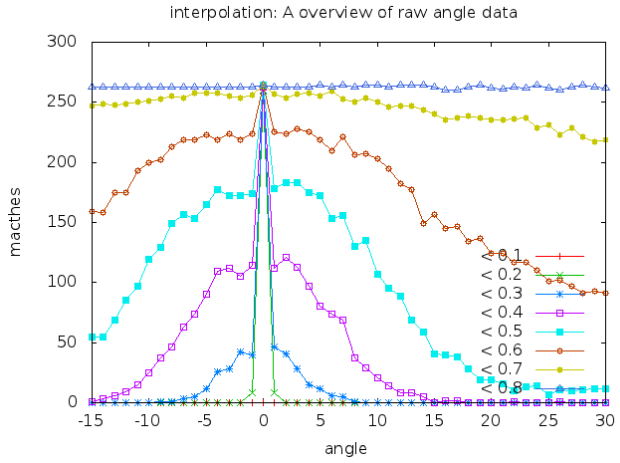

Fig. 17. Distance between descriptor matches for different thresholds.

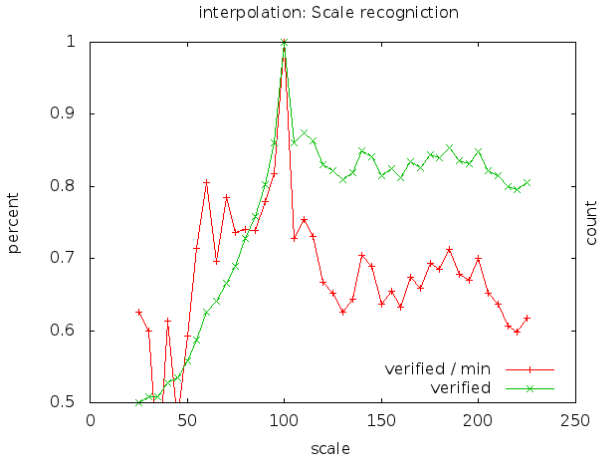

Fig. 18. Repeatability for scaling

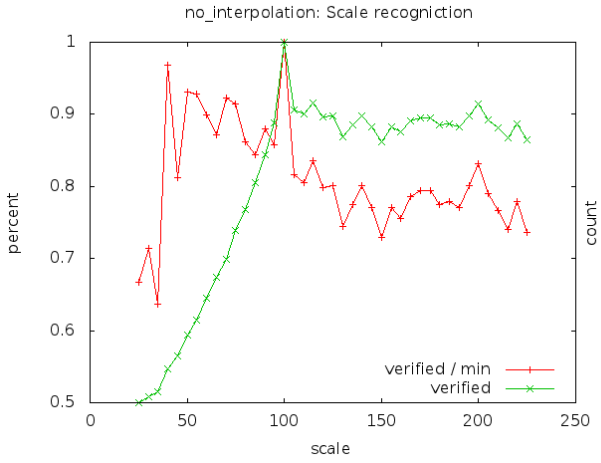

Fig. 19. Repeatability for scaling without interpolation

would be reasonable, Figure 16 shows a matching with a tolerance of 0.4. If the threshold is higher false positives starts occuring, here there are no visual noticeable false positives. This comparison is made using the euclidean distance between the descriptor vectors, it could be interesting to examine the effect of using the Mahalanobis distance[12].

When scaling it is possible to measure the repeatability, this is illustrated in Figure 18. There is a spike at 100% scaling where the recognition of course is 100%. At other scales it hovers around 70% which is a bit worse that the measurements in the SURF-article [2] and Gossow et al [7]. As mentioned in Section 4.1 it is possible to compile without interpolation. The results without interpolation is shown in Figure 19 and it is frustrating to observe that complex calculations to interpolate in scale space apparently gives worse results.

The reason for this discrepancy could be as simple as an implementation bug, it could also be an side effect from the scaling. Without interpolation the subset of possible positions in scale space is fixed and it is entirely possible that the better results simple is the result of the positions *snapping to a grid*. This suspicion is further fueled when inspecting the data with different thresholds in Figure 20 and 21. The spikes could very well be a *snap to grid* side effect. Its most likely a combination of several details.

It may also be a the fault in the descriptors robustness to translations. There is reason to believe that fiddling with the parameters for calculation of the descriptors will influence the results. The reason is that the spikes in the plots were more pronounced before changing the parameters for calculating the descriptors[13]. There are several options that could be explored in more detail, and Gossow et al do mention that algorithms using another interpolation scheme for the descriptors than the original consis-

12. This is suggested by Bay et al [2]

13. Correctly centering the Gaussian on the interest point and not the subarea did improve the results, however both increasing or decreasing σ gave worse results

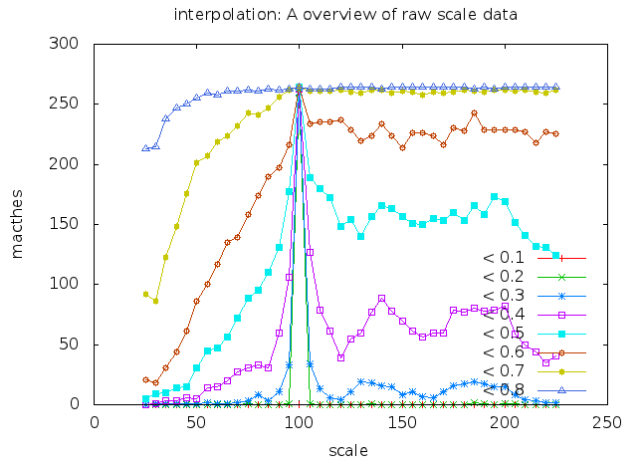

Fig. 20. The scaling data

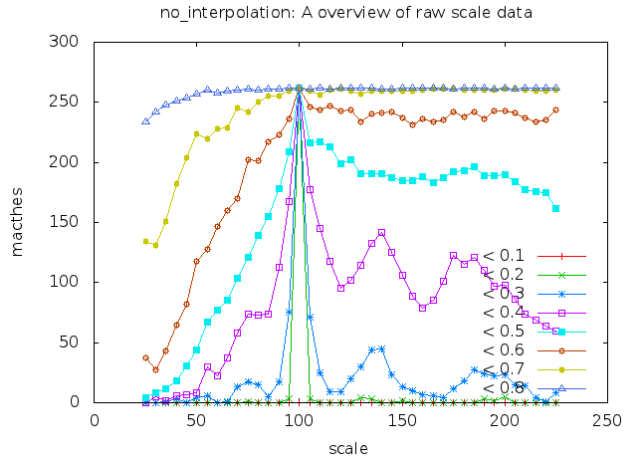

Fig. 21. The scaling data with out interpolation.

tently showed better results[14].

Besides exploring different techniques it could be interesting to explore the parameters for the current implementation in more depth. There are many different parameters that can influence the type and amount of interest points. For instance the thresholding before doing non-maximum suppression, the number of octaves and levels and the response maps initial size could be investigated further.

6 Conclusion

Implementing the SURF algorithm has proven to be a challenge. The results are not optimal, but as preceding sections shows, there are vast possibilities for tweaking and fiddling, and there are many places a subtle bug might hide. Gossow et al [7] shows that the implementation details do matter, and that there is more than one way to implement it.

It has been interesting and time consuming to implement the algorithm from the ground up. If I were to employ the SURF algorithm to a real world problem in the future, this experience will be valuable when adapting a open source implementation to my needs. Having more eyes on the code can help optimize details and assure correct implementations. This would free resources to investigate different variations of parameters and strategies.

In conclusion, I consider this implementation successful. It is able to detect and describe with consistent results and demonstrates the core principles of the SURF algorithms detection and description scheme.

14. openSURF is using another scheme, I have not been able to find the paper they reference: *Agrawal ECCV 08*

Jacob Toft Pedersen Computer Science, Aarhus.

REFERENCES

[1] D. G. Lowe, "Distinctive image features from scale invariant keypoints," *International Journal of Computer Vision*, 2004.
[2] S. U. R. F. (SURF), "Herbet bay, andreas ess, tinne tuytelaars, luc van gool," *Elsevier preprint*, 2008.
[3] Wikipedia, "Blob detection — Wikipedia, the free encyclopedia," 2011, [Online; accessed 14-July-2010]. [Online]. Available: https://secure.wikimedia.org/wikipedia/en/wiki/Blob_detection
[4] M. B. David Lowe, "Invariant features from interest point groups," *BMVC*, 2002.
[5] C. Evans, "Notes on the opensurf library," University of Bristol, Tech. Rep. CSTR-09-001, January 2009. [Online]. Available: http://www.chrisevansdev.com
[6] L. V. G. Alexander Neubeck, "Efficient non-maximum supression," *ICPR*, 2006.
[7] D. P. David Gossow, Peter Decker, "An evaluation of open source surf implementations," Active Vision Group, University of Koblenz-Landau, Tech. Rep., 2009?
[8] Wikipedia, "Feature detection — Wikipedia, the free encyclopedia," 2011, [Online; accessed 14-July-2010]. [Online]. Available: https://secure.wikimedia.org/wikipedia/en/wiki/Feature_detection_%28computer_vision%29

CONTENTS

APPENDIX A PROGRAM

The source code is available at http://cs.au.dk/~jtp/SURF. There is a `Makefile` and it should compile on any linux platform with the `openCV` libraries installed.

The basic usage is:

```
bin$./main needles.png haystack.jpg
```

Which will search `needles.png` for interest points and try to match them to interest points found in `haystack.jpg`, it will then diplay the result as in Figure 16.

The main program accepts several options, the most important are:

-o `--output` *filename* will save the output instead of displaying it.
-c `--create` *filemame* will save description of `needles.png` and not try to match.
-l `--load` *filemame* will load description of interest points and try to match.
-t `--tolerance` *dist* will only consider it as a match if the distance between descriptor vectors is less than *dist*
-v `--verbose` will be verbose, several `v`'s as `-vvvvv` will increase verbosity level.

There are further options to produce debugging output, statistical output or run tests. Theese options are documented in *main.cpp*.

To run automated test there is a bash script `createcase.sh` in the `data` directory. It can be run as

```
data$./createcase.sh bbq.jpg suffix
```

This will create plots the directory `data/bbq/plots`. This was used to generate the plots in the used in Section 5

APPENDIX B RESOURCES

The main resources for doing the implementation has been the SURF paper [2] and Evans [5].

To get a general feel of computer vision and the techniques used Wikipedia does have an good section on feature detection [8]. The SIFT

article [1] and http://www.aishack.in/2010/05/sift-step-1-constructing-a-scale-space/ made it possible to make a rudimentary SIFT - detector, it is not a part of the codebase any longer, as the initial plan to have both algorithms were not feasible.

In general http://www.aishack.in/ does have some interesting articles on computer vision and working with openCV. With descriptions of various concepts: Hough transform, features, corner detection, scale space et cetera.

I was led on a an interesting but none the less *wild goose chase* with optimizing non-maximal suppression. It was an interesting article but it was not an essential part of the algorithm.

The documentation supplied with `openCV` is good and thorough, it was unfortunate that i did not realize that their addressing convention were *col, row* and not *row, col*, this was not noticeable before is causes x and y coordinates to be switched. This particular misunderstanding and rigorously going through the code to fix it were particular annoying.

LIST OF FIGURES