Box filtering for real-time curvature scale-space computation

M Hennig and B Mertsching
GET Lab, Department of Electrical Engineering and Information Technology, Paderborn University, 33098 Paderborn, Germany
E-mail: hennig@get.upb.de

Abstract. Curvature scale-space (CSS) analysis is an important technique for contour-based object recognition in digital images. To compute the CSS for a given contour, it is systematically convolved (smoothed) with Gaussians with increasing standard deviation. The convolutions are computationally expensive, especially for large and high resolution contours, but can be approximated using box filtering (also known as mean and average filtering). Together with running sums, the convolutions can be accelerated by 2–3 magnitudes without significant loss of precision. Nonetheless, box filtering has not been systematically investigated in connection with CSS computation. In this work, we present a theoretical and experimental analysis of different box-filtering techniques in this context and conclude which is the most efficient implementation. Based on this, the CSS of a contour can be computed in real time with high precision.

1. Introduction
Object contours are important features for object recognition in digital images because they outline the shape of objects, their distinct parts and regions, and separate them from the background and other objects. The importance is also underlined by the fact that human observers can easily identify many different objects only based on their contour. In digital images, object contours can be detected even in case of challenging illumination conditions and complex textures. The methods for this purpose range from gradient based edge detection [1] to advanced deep-learning based segmentation techniques [2]. For object recognition, the detected contours can be analyzed, described and compared (measuring the similarity) in many different ways [3, 4]. In general, it can be distinguished between methods which work directly on contours (e.g., by extracting interest points or contour segments) and region-based methods (e.g., by extracting enclosed regions). CSS-based methods work directly on contours by computing a multi-scale representation of the contour curvature [5].

This work addresses the computation of the CSS for a given contour in its most general form, without any pre- and post-processing steps (such as edge detection or length-normalization). Note that in the literature, CSS often refers to a specific method where the zero-crossings of length-normalized contours are used to form a global shape descriptor [5]. CSS analysis is interesting because it can be used for a range of different applications. For example, to detect meaningful contour (object) structures in an automatic and scale-invariant manner (see example in figure 1, right), and for multi-scale edge detection and contour segmentation [5]. Moreover, CSS analysis can be used for many different kinds of data, not only images. In the field of aircraft systems, CSS-based methods can be interesting to analyze and detect objects in the air.
Figure 1. CSS analysis example (contour of a RSK MiG-35), where curvature extrema are traced to extract local scale-invariant contour features. The algorithm used is an extension of the method described in [7].

and on the ground based on different kinds of sensor data. A somewhat related work in this field is presented in [6], where a scale-space based method is used to analyze high-resolution range profiles of radar targets.

2. Curvature scale-space
The first step to compute the CSS for a discrete contour \(\Gamma\) with length \(N_\Gamma\) is to parameterize it by its integer arc length \(u \in [0, N_\Gamma - 1]\):

\[
\Gamma = (x(u), y(u)).
\]  

Next, \(\Gamma\) is systematically convolved with 1D Gaussians \(g\) with increasing standard deviation (scale parameter) \(\sigma \in [\sigma_{\text{start}}, \sigma_{\text{end}}]\), where \(\sigma\) is incremented by \(\sigma_{\text{step}}\):

\[
g(u, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{u^2}{2\sigma^2}\right).
\]  

As the Gaussians have an infinite support, they have to be truncated. For this purpose, we use an odd number of \(N_g = 2\lceil 4.5\sigma \rceil + 1\) samples around the mean. This leads to an accurate approximation. The number is odd to center the Gaussians on each contour pixel. The discrete convolutions are computed as follows (star denotes convolution):

\[
x_{\sigma}(u, \sigma) = x(u) \ast g(u, \sigma), \quad y_{\sigma}(u, \sigma) = y(u) \ast g(u, \sigma).
\]  

The resulting functions represent evolved (smoothed) versions of \(\Gamma\), as shown in figure 1 (left). For each \(\sigma\), the curvature is then computed as follows:

\[
\kappa(u, \sigma) = \frac{x_{\sigma}(u, \sigma) y_{\sigma}(u, \sigma) - x_{\sigma}(u, \sigma) y_{\sigma}(u, \sigma)}{(x_{\sigma}^2(u, \sigma) + y_{\sigma}^2(u, \sigma))^{3/2}}.
\]  

The functions in this equation are discrete first and second order derivatives with respect to \(u\). We compute these derivatives as described in [8] so that they have scale-space properties. The corresponding CSS is the result of arranging the \(\kappa-u\)-planes consecutively with increasing scale parameter \(\sigma\) (see figure 1, center; determining the signatures shown is another step).

Because convolution commutes with differentiation, and the resulting functions from equation (3) are differentiated in equation (4), there are three principal variants to compute \(\kappa\):
Figure 2. Comparison of different variants to compute $\kappa$ for a parametrized contour; BFC = box filter complexity for one convolution and $k$ box-filtering steps (see text for details).

(i) differentiating the convolved signals, (ii) convolving the signals with differentiated Gaussians, or (iii) convolving the differentiated signals. For further analyses, we have summarized these variants in figure 2. Note that the second order derivatives $\ddot{x}_\sigma$ and $\ddot{y}_\sigma$ can always be computed by differentiating $\dot{x}_\sigma$ and $\dot{y}_\sigma$, respectively. This is not explicitly considered here. The box-filter complexity refers to the construction complexity of a box filter with $k$ filtering steps to approximate a Gaussian or its derivatives. For example, the Gaussian $g$ in (i) can be approximated by a single box so that $k = 1$. The first Gaussian derivative $\dot{g}$ in (ii) can be approximated by two boxes so that $k = 2$ (one box for the negative and one for the positive part of the derivative), and $\ddot{g}$ by three boxes so that $k = 3$.

Which of the three variants should be used to compute $\kappa$ based on box filtering? We argue for variant (i): while all variants should theoretically lead to the exact same results, there are characteristic differences in the computation process. In particular, the number of convolutions is lowest in variant (i), and convolutions are computationally expensive. Furthermore, it is more easy to approximate Gaussians directly instead of their derivatives, as it would be required in variant (ii). In particular, the construction process has a higher complexity (BFC). In variant (iii), the signals are first differentiated, which can lead to numerical inaccuracies for discrete input signals in equation (4). In variant (i), the signals are first smoothed so that this is not an issue. In summary, variant (i) is most efficient and robust. Therefore, we use variant (i) for the following experiments.
3. Box filtering

Gaussian convolution can be accelerated and approximated in many different ways, in particular by recursive infinite impulse response (IIR) filters, binomial filters, discrete Fourier transform, and pyramid-based methods [10, 14]. Box filters, as shown in figure 3, are interesting because they work at fixed cost per pixel when used together with running sums, independent of the size of the Gaussian, and are easy to implement.

![Box filtering approximations](image)

Running sums (also known as summed area tables and integral images) have become popular in the literature for 2D images in the context of fast face detection [15] and feature extraction [16] and are adapted for parameterized contours here. The principle is shown in figure 4. The first step is to compute the running sum of the contour as follows:

\[ x_{\sum}(\gamma) = \sum_{u=0}^{\gamma} x(u). \]  

In the same manner, the running sum \( y_{\sum}(\gamma) \) has to be computed for \( y(u) \). Then, the sum of an arbitrary part of the signal can be computed by a single subtraction. As can be seen in figure 3, such signal sums are the basis of box filters. Normal convolution with a filter with \( N_f \) coefficients requires \( N_f \) multiplications and summations for each signal sample. Reducing this to a single subtraction (and one division to consider the height of the box filter) leads to a significant acceleration of the filtering process, even when using \( k \) filtering steps (\( k \) is typically...
small, e.g. 3). Especially large contours require large filter sizes to achieve smoothing so that box filters are particularly efficient here.

In this work, we consider four common box-filter approximations as shown in figure 3. The simplest method is SBF, where the signal is filtered $k$ times with the same filter in an iterative manner. The filter size required to achieve the equivalent $\sigma$ of the Gaussian can be determined analytically [10]. One problem of this method is that the ideal filter size, which is typically a real valued number, has to be rounded to an odd number to center the filter on each contour pixel. As this rounding leads to inaccurate filter results, the method can be improved by using two filters with different sizes, which are also applied in an iterative manner ($k$ times in total). This method is known as FAG. The SB method uses $k$ box filters in a stacked manner to resemble the shape of the Gaussian. The box filters can be constructed based on minimizing a cost function which computes the deviation of the resembled version from the original Gaussian [11]. EBF can be seen as a mix of iterated filtering and stacked boxes. In this case, two boxes are stacked and applied $k$ times in an iterative manner. Similar to SBF and EBF, the filter coefficients are determined analytically.

4. Experiments and results

In our evaluation, we analyze the computation time and accuracy of the different box-filter approximations for $k = 3$ and 5 filtering steps. We have chosen $k$ like this because for smaller values, the approximations become inaccurate, and for larger values there is no significant accuracy gain. We have used our own C++ implementations of the box-filter methods. The parameter values for our experiments are summarized in table 1, where $\sigma_{\text{end}}$ is only used to analyze the computation time, not the accuracy (there we use a fixed value to consider different contours).

![Figure 4. Principle of running sums: based on the running sum $x_\Sigma(\gamma)$, the sum of an arbitrary part of the signal can be computed by a single subtraction.](image)

| Parameter | Description | Value |
|-----------|-------------|-------|
| $N_g$     | Number of samples of the Gaussian | $2[4.5\sigma] + 1$ |
| $\sigma_{\text{start}}$ | Min. scale parameter | 8.0 |
| $\sigma_{\text{end}}$ | Max. scale parameter | $0.2N_\Gamma$ |
| $\sigma_{\text{step}}$ | Increment of the scale parameter | 2.0 |
| $k$       | Number of box filter steps | 3, 5 |

To analyze the computation time, we have used an artificial contour in the form of a straight line, because only the number of operations is relevant here, not the form of the contour. The results for contour lengths $N_\Gamma$ from 100 to 10000 px are shown in figure 5 (Hardware: Intel i7-8086K CPU @ 4.00 GHz). The computation times have been measured for the convolutions to compute $x_\sigma$ and $y_\sigma$ for all values of $\sigma$, as specified in table 1. In other words, the computation of the complete CSS is considered. As can be seen, the box filters work 2–3 magnitudes faster...
than the normal convolutions, and FAG is always the fastest method. The CSS computation can be further accelerated by using different values for $\sigma_{\text{step}}$, e.g., depending on $\sigma$.

To analyze the accuracy of the different box-filter approximations, we have used five test shapes from the MPEG-7 CE-Shape-1 data set [17] and computed their contours using the Canny edge detector. The contours are shown in figure 6. Since the contours are closed, boundary padding is not required. Next, we have computed $\kappa(u, \sigma)$ as reference according to

![Figure 5. Evaluation results for $k = 3$ and 5 main filtering steps (the mean RMSE is computed for $\kappa$, see text for details).](image)

![Figure 6. Test contours from MPEG-7 CE-Shape-1 data set used to analyze the accuracy of the different box-filter approximations.](image)
equation (4) using the truncated Gaussian with \( N_g \) samples as specified in table 1. Finally, we have computed \( \kappa(u, \sigma) \) using the different box-filter approximations and computed the root-mean-square error (RMSE) of \( \kappa(u, \sigma) \) compared to the reference along the complete contour, and then taken the mean value of this error among the five contours. The results are shown in figure 5. As can be seen, except for SBF, the results are close together, and concerning \( k \), the error is smaller for \( k = 5 \) (as expected). The rising slopes in the center of the graphs disappear when the contours crown and device-6 are not considered (not explicitly shown here). In this case, the error decreases with increasing \( \sigma \). Most likely, this is a result of the sharp transitions along these contours in combination with the denominator of equation (4), where small changes can have a large impact on \( \kappa(u, \sigma) \). Larger values for \( k \) might reduce the error, but the total error is already very small. With a larger test data set, this behavior will presumably be less or disappear.

5. Conclusion
The results of our evaluation show that the convolutions can be accelerated by 2–3 magnitudes without significant loss of precision (note the logarithmic scale of the time axes in figure 5). Even large contours (\( N_{\Gamma} = 10,000 \) px) can be processed in real-time when using a suitable parametrization for the CSS computation. In summary, FAG is the most efficient method and should be used with \( k = 5 \) filtering steps in combination with variant (i) from figure 2. If the contours do not contain sharp transitions or a slightly larger error is acceptable, \( k = 3 \) can be used.

References
[1] Li Y, Wang S, Tian Q and Ding X 2015 A survey of recent advances in visual feature detection *Neurocomputing* **149** 736–751
[2] Minae S, Boykov Y, Porikli F, Plaza A, Kehtarnavaz N and Terzopoulos D 2020 Image segmentation using deep learning: A survey (Preprint arXiv:2001.05566)
[3] Yang M, Kpalma K and Ronson J 2008 A Survey of shape feature extraction techniques *Pattern Recognition: Techniques, Technology and Applications* 43–90
[4] Ma J, Jiang X, Fan A, Jiang J and Yan J 2021 Image matching from handcrafted to deep features: A survey *International Journal of Computer Vision* **129** 23–79
[5] Mokhtarian F and Bober M 2003 Curvature scale space representation: theory, applications, and MPEG-7 standardization (Springer Science & Business Media)
[6] Liu J, Fang N, Xie Y J and Wang B F 2016 Scale-space theory-based multi-scale features for aircraft classification using HRRP *Electronics Letters* **52**(6) 475–477
[7] Hennig M and Mertsching B 2016 Contour Analysis in Biological and Artificial Cognitive Systems 5th Interdisciplinary Workshop on Cognitive Systems (KogSys 2016)
[8] Lindeberg T 1993 Discrete derivative approximations with scale-space properties: A basis for low-level feature extraction *Journal of Mathematical Imaging and Vision* **3**(4) 349–376
[9] Wells W M 1986 Efficient synthesis of Gaussian filters by cascaded uniform filters *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(2) 234–239
[10] Kovesi P 2010 Fast almost-gaussian filtering *Proc. of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)* 121–125
[11] Bhatia A, Snyder W E and Bilbro G 2010 Stacked integral image *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)* 1530–1535
[12] Elboher E and Werman M 2012 Efficient and accurate Gaussian image filtering using running sums *Proc. of the 12th International Conference on Intelligent Systems Design and Applications* 897–902
[13] Gwosdek P, Greweinig S, Bruhn A and Weickert J 2011 Theoretical foundations of gaussian convolution by extended box filtering *Proc. of the International Conference on Scale Space and Variational Methods in Computer Vision* 447–458
[14] Getreuer P 2013 A survey of Gaussian convolution algorithms *Image Processing On Line* **2013** 286–310
[15] Viola P and Jones M J 2004 Robust real-time face detection *International Journal of Computer Vision* **57**(2) 137–154
[16] Bay H, Ess A, Tuytelaars T and Van Gool L 2008 Speeded-up robust features (SURF) *Computer Vision and Image Understanding* **110**(3) 346–359
[17] MPEG-7 Core Experiment CE-Shape-1 Test Set (Part B) last checked: 2021-03-27 URL https://dabi.temple.edu/external/shape/MPEG7/dataset.html