CSANG: Continuous Scale Anisotropic Gaussians for Robust Linear Structure Extraction

V. B. Surya Prasath*, Sruthikesh Surineni†, Ke Gao*, Guna Seetharaman‡, Kannappan Palaniappan*

*Computational Imaging and VisAnalysis (CIVA) Lab
Department of Computer Science, University of Missouri-Columbia, MO 65211 USA
Emails: {prasath, palaniappan}@missouri.edu, kg954@mail.missouri.edu
†Networking and Parallel Systems Lab
Department of Computer Science, University of Missouri-Columbia, MO 65211 USA.
Email: ssurineni@mail.missouri.edu
‡US Naval Research Laboratory, Washington D.C., USA

Abstract—Robust estimation of linear structures such as edges and junction features in digital images is an important problem. In this paper, we use an adaptive robust structure tensor (ARST) method where the local adaptation process uses spatially varying adaptive Gaussian kernel that is initialized using the total least-squares structure tensor solution. An iterative scheme is designed with size, orientation, and weights of the Gaussian kernel adaptively changed at each iteration step. Such an adaptation and continuous scale anisotropic Gaussian kernel change for local orientation estimation helps us obtain robust edge and junction features. We consider an efficient graphical processing unit (GPU) implementation which obtained $30 \times$ speed improvement over traditional central processing unit (CPU) based implementations. Experimental results on noisy synthetic and natural images indicate that we obtain robust edge detections and further comparison with other edge detectors shows that our approach obtains better localization and accuracy under noisy conditions.

1. Introduction

Edge detection and feature extraction are important image processing tasks. Digital image edge detection has a long history and has been the object of study for many decades now [1], [2], [3]. Many of these edge detectors are based on image derivatives and gradients and thus require usually pre-smoothing with a smoothing filter for e.g. convolution with a Gaussian kernel. One of the classical and most famous edge detector is due to Canny [3], which is still widely used in many image processing pipelines. Though Canny edge detector includes Gaussian pre-smoothing, it can still obtain spurious edge pixels under noise (see e.g. Figure 1) or double edges, and various improvements have been proposed over the years. Structure tensors are alternative to gradient computations and recently they are advocated in various image processing tasks where edges, orientations are required to be computed accurately and robustly under noisy conditions [4], [5].

Nath and Palaniappan [6] considered an adaptive robust structure tensor (ARST) approach for orientation estimation and image segmentation. The ARST is derived via robust statistics and oriented Gaussian kernel adaption in an iterative manner. By utilizing a spatially varying adaptive Gaussian kernel, improved accuracy in finding edge and corner locations were obtained. Further, robust error function such as Geman-McClure is utilized for estimating ARST in local neighborhoods which leads to better localization of low level linear structure extraction. Due to the nonlinear dependency on the orientation of the patch and iterative nature of the ARST estimation the method requires high computational cost. In this work, we consider utilizing graphical processing unit (GPU) for the main computational iterative scheme of ARST. We show that GPU ARST for fast edge detection with an average $30 \times$ speed improvement over traditional central processing unit (CPU) based implementations. We provide comparative experimental results on edge detection with other well-known edge detectors on synthetic, natural images.

Rest of the paper is organized as follows. In Section 2 we review the adaptive robust structure tensor (ARST) and
corresponding edge detection approach. Section 3 provides experimental results on various synthetic, real images and comparison with other edge detectors. Finally, Section 4 concludes the paper.

2. Adaptive Robust Structure Tensor for Edge Detection

2.1. Robust Estimator for Gradient Estimation

We follow the notations and conventions from [6] and detail the iterative method for the robust estimator based continuous scale anisotropic Gaussians adaptation. Let \( v(x) \) be the true (latent) gradient of an image patch \( \Omega(y) \), centered at pixel \( x \). The norm of the error vector between the estimated gradient \( g(y) \) at location \( y \) and \( v(x) \) is given by \( e(x, y) \) as

\[
||e(x, y)|| = ||g(y) - (g^T(y) v(x))v(x)||.
\]

The aim here is to estimate \( v \), hence the straightforward way is to minimize an error functional \( \rho(||e(x, y)||^2) \) integrated over the image patch \( \Omega \),

\[
e_{LS}(x, y) = \int_{\Omega} \rho(||e(x, y)||^2) W(x, y) \, dy. \tag{2}
\]

Here, \( W(x, y) \) is a spatially invariant weighting function, for example the Gaussian kernel,

\[
W(x, y) = (2\pi\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(y-x^2)}. \tag{3}
\]

This weighting function emphasizes the gradient at the central pixel within a small neighborhood, and we utilize an adaptive size and orientation based weighting function estimation, see Section 2.2. One of the well-known and basic formulation is based on the method of least-squares which corresponds to setting \( \rho(s) = s^2 \), thus Eqn. (2) becomes,

\[
e_{LS}(x, y) = \int_{\Omega} (g^T g) W(x, y) \, dy
\]

\[
- \int_{\Omega} (v^T (gg^T)v) W(x, y) \, dy, \tag{4}
\]

subject to the condition\(^1\) that \( ||v|| = 1 \). We utilize the Lagrange multipliers to write the unconstrained formulation as,

\[
E_{LS}(x, y) = v^T (\int_{\Omega} (gg^T) W(x, y) \, dy) v + \lambda(1 - v^T v). \tag{5}
\]

Differentiating \( E_{LS}(x, y) \) to find the extremum leads to the standard eigenvalue problem for solving the best estimate of \( v \), given by \( \hat{v} \).

\[
J(x, W) v = \lambda v, \tag{6}
\]

where

\[
J(x, W) = \int_{\Omega} (gg^T) W(x, y) \, dy.
\]

We note that this is the least-squares structure tensor at position \( x \) using the weighting kernel \( W \). The maximum eigenvector solution of Eqn. (6) gives the least-squares estimate for the gradient at pixel \( x \) using the surrounding gradient information\(^2\).

In [6], robust error measures are advocated instead of the least squares due to their noise tolerant property. Robust error functions impose smaller penalties on outliers [7]. One of the important example for robust error functions is the Geman-McClure function [7] and defined as,

\[
\rho(||e(x, y)||, m) = \frac{||e(x, y)||^2}{m^2 + ||e(x, y)||^2} = 1 - \frac{m^2}{m^2 + ||e(x, y)||^2}, \tag{7}
\]

where, \( m \) is a parameter that determines the amount of penalty imposed on large errors. If we utilize the Geman-McClure robust error function in Eqn. (2) we get,

\[
e_{GM}(x, y) = \int_{\Omega} W(x, y) \, dy
\]

\[
- \int_{\Omega} \frac{m^2}{(g^T g - v^T (gg^T)v + m^2)} W(x, y) \, dy, \tag{8}
\]

Using Lagrange multipliers we can write the unconstrained version as follows,

\[
E_{GM}(x, y) = \int_{\Omega} \frac{m^2}{(g^T g - v^T (gg^T)v + m^2)} W(x, y) \, dy + \lambda(1 - v^T v). \tag{9}
\]

Differentiating \( E_{GM}(x, y) \), with respect to \( v \), and setting it to zero gives

\[
J(x, v, W) v = \lambda v, \tag{10}
\]

where,

\[
J(x, v, W) = \int_{\Omega} \frac{m^2}{(g^T g - v^T (gg^T)v + m^2)^2} (gg^T) W(x, y) \, dy, \tag{11}
\]

is the Geman-McClure robust structure tensor. The following iterative equation,

\[
J(x, v, W) v_{i+1} = \lambda v_{i+1}, \tag{12}
\]

is a fixed-point functional iteration scheme for numerically solving \((\lambda, v)\) in Eqn. (10) that usually converges to a local minimum [8]. We use total least-squares solution to initialize the iterative process in Eqn. (12).

2.2. Continuous Scale Anisotropic Gaussians

Following [6] we use a soft Gaussian convolution function as the weighting function \( W(x, y) \) in our iterative formulation (12) with adaptive size and orientation within \( \Omega \).

Note that, \( v^T(x) \) could be determined by using the minimum eigenvector, however for an ideal edge, the smaller eigenvalue will be zero. Thus, it is numerically more reliable to estimate the maximum eigenvector.

\(1\) Both \( ||v|| = 1 \) and \( ||g|| = 1 \) as these are direction vectors.

\(2\) This is due to the fact that the structure tensor is a quadratic form of the gradient.
Spatially varying adaptation of the kernel (local neighborhood shape and coefficients) is beneficial for improving the estimation of oriented image structures. The neighborhood \( \Omega \) is initialized as a circular region and subsequently adapted to be an oriented elliptical region. Figure 2 shows the adaptive algorithm at an ideal edge. In this figure, the dashed-line elliptical region is oriented along the gradient while the solid-line elliptical region (that is scaled and rotated by 90°) is oriented along the edge. The spatially varying kernel \( W_i(x,y) \) that is used with Eqn. (11) is defined as,

\[
W_i(x,y) = K e^{-\frac{1}{2}((y-x)^T U_i^{-1} \lambda_i^{-2} U_i^{-1}(y-x))^2},
\]

where \( K \) is a scaling factor associated with the Gaussian function. We initialize the kernel \( W_0(x,y) \) as an isotropic Gaussian with \( R_0^1 = R_0^2 = R_0 \). A fairly large number is chosen (typically \( R_0 = 8 \)), in order to reduce the influence of noise when evaluating the structure tensor. The columns of \( U_1 \) are the eigenvectors \((e_1^i, e_2^i)\), with the columns of \( U_{i-1} \) initialized as the co-ordinate axes. Let \( \lambda_1^i \) and \( \lambda_2^i \) (with \( \lambda_1^i > \lambda_2^i \)) be the eigenvalues of the structure tensor \( J_i(x, v_{i-1}^1, W_i) \) at the \( i \)th iteration. Scaled versions of these eigenvalues are used to update the semi-major and semi-minor axes for the \((i+1)\)th iteration as

\[
R_{i+1}^1 = \frac{\lambda_1^i}{\lambda_1^i + \lambda_2^i} R_i^1 \quad \text{and} \quad R_{i+1}^2 = \frac{\lambda_2^i}{\lambda_1^i + \lambda_2^i} R_i^2.
\]

The eigenvectors obtained from the current iteration, along with \( (R_1^i, R_2^i) \), are used to update the kernel as follows

\[
W_{i+1}(x,y) = K e^{-\frac{1}{2}((y-x)^T U_i^{-1} \lambda_i^{-2} U_i^{-1}(y-x))^2},\]

\[
\Lambda_{i+1} = \begin{bmatrix}
\sqrt{2}R_{i+1}^1 & 0 \\
0 & \sqrt{2}R_{i+1}^2
\end{bmatrix}.
\]

This kernel is used to compute a new structure tensor \( J_i(x, v_{i+1}^i, W_{i+1}) \) as per Eqn. (11). To account for the spatially varying Gaussian kernel, Eq. 12 is modified to the following form,

\[
J_i(x, v_{i+1}^i, W_{i+1}) v_{i+1}^i = \lambda v_{i+1}^i.
\]

We experimentally determined that two or three iterations were sufficient to achieve good results.

### 2.3. Edge Detection with CSANG

**Algorithm 1 ARST - edge detection**

**Input:** \( u \), image of size \( m \times n \); \( t_{hi}, t_{lo} \), hysteresis thresholds \( (t_{hi} > t_{lo}) \).

**Returns:** a binary edge image of size \( m \times n \).

1. **procedure** INITIALIZATIONS
2. Compute the iterative solution \( v \), See Eqn. (16)
3. Create maps:
   4. \( E_{mag}(x,y) : m \times n \rightarrow \mathbb{R} \)
   5. \( E_{nms}(x,y) : m \times n \rightarrow \mathbb{R} \)
   6. \( E_{bin}(x,y) : m \times n \rightarrow \{0,1\} \)
4. for all image coordinates \((x,y) \in m \times n\) do
5. \( E_{mag}(x,y) \leftarrow |v| \)
6. \( E_{nms}(x,y) \leftarrow 0 \)
7. \( E_{bin}(x,y) \leftarrow 0 \)
8. **procedure** NONMAXIMUM SUPPRESSION
9. for \( x \leftarrow 1, \ldots, m-2 \) do
10. for \( x \leftarrow 1, \ldots, n-2 \) do
11. \( e_x^+, e_x^- \) eigenvectors of structure tensor
12. \( s \leftarrow \text{GetOrientationSector}(e_x^+, e_x^-) \)
13. If \( \text{IsLocalMax}(E_{mag}, x, y, s, t_{hi}) \) then
14. \( E_{nms}(x,y) \leftarrow E_{mag}(x,y) \)
15. **procedure** HYSTERESIS THRESHOLDING
16. for \( x \leftarrow 1, \ldots, m-2 \) do
17. for \( x \leftarrow 1, \ldots, n-2 \) do
18. If \( (E_{nms}(x,y) \geq t_{hi} \wedge E_{bin}(x,y) = 0) \) then
19. TraceAndThreshold(\( E_{nms}, E_{bin}, x, y, t_{lo} \))
20. return \( E_{bin} \)

The solution of the iterative procedure (16) \( v \) is used to obtain an edge map by applying non-maximal suppression and hysteresis thresholding steps, see Algorithm 1. These are standard steps in obtaining binary edge maps with single pixel width and originally used by Canny edge detector [3]. We next details the graphical processing unit (GPU) based implementation details of the core iterative procedure.

### 2.4. GPU Implementation Details

#### 2.4.1. Parallelization approach
Figure 3. Our GPU implementation involves the following. (a) iterative GPU execution pipeline, and (b) thread block scheduling.

- **Structure Tensor:** This is one time executed kernel which loads input image into shared memory. First, we compute the gradients on shared memory data and from the computed gradients structure tensor matrix is estimated.

- **Robust Estimator:** In this kernel, we compute the GM robust estimation for smoothness and texture terms from gradients and structure tensor. All the computation equations are shown in the pseudo code. Shared memory is used for computing 2nd derivatives similar to normal gradients. Since GM robust estimation is same for both smoothness and texture we implemented it as a device function. GM robust function there is a floating point division which is very slow to make it faster we approximate division by multiplication with reciprocal.

- **Input for solver:** Inputs to Thomas solver are prepared using a couple of kernels. Inputs to these kernels are from Robust estimator and they are transposed for row-wise gradients. In the original source code, there is a data dependency on the previous value. To avoid this data dependency and make the algorithm parallel we load the input data into shared memory and re-compute the previous value so that data dependency is eliminated. The overhead for re-computation of previous value is quite minimal.

- **CuSparse batched gtsv:** Thomas solver is used from CuSparse gtsv library function. If we consider a matrix, this solver solves a column each instance. So, to make the column elements coalesced we do a matrix transpose on inputs. For multiple columns to be solved at the same time we use batched gtsv solver but this requires a considerable amount of global memory (batchCount \( \times (4?m+2048) \times \) sizeof(\( \text{type}_G \))). In some cases we ran out of memory, to solve this issue we run the solver multiple iterations with fewer batches(columns) considering the free memory available at the launch of the kernel. Gradients are updated by taking the average of row wise and column wise gradient estimation.

- **Orientation Estimation:** Finally, once all the iterations for computing the gradients are done we compute the orientation of the gradients. Orientation is estimated using atan2f, an inbuilt math function.

- **Matrix Transpose:** This is the most instanced kernel in the whole algorithm. Implementing this optimally is critical for performance. This kernel is optimized for square block size. First, we load the input matrix into shared memory. A different mapping is used to read data from shared memory and store data into output coalesced.

**Figure 3(a) shows the GPU execution pipeline discussed above.**

**2.4.2. Thread scheduling and shared memory utilization.** Details about thread scheduling, shared memory utilization are described below, see Figure 3(b).

- **Thread Scheduling:** Input image could be of any resolution and thread blocks need to be traversed over the whole image. Though we have a 2D grid of blocks we convert this 2D grid into a 1D value in terms of the image width. From this image width, we compute the current iteration \( X \) and \( Y \). Shown below is the image considering grid dimensions of \( < 3, 2 > \). Thread blocks first walk the image width and once we hit the boundary of the image then we move to next row(vertical) and so on.

- **Shared memory utilization:** Shared memory is used in most of the kernels. Shared memory in matrix transpose is straight forward. Shared memory for computing gradients is shown below. For computing a gradient in \( X \) we need previous and next value in input, gradient in \( Y \) we need row above and row below value. To optimize all the overlaps we load the data into shared memory and for all the boundary elements we need an extra load. To reduce the warp divergence the threads with threadIdx.y==0, threadIdx.x==0 load two extra elements. For example, if we have a block of \( < 8, 8 > \) we load 10 \( \times \) 10 data into shared memory. Since we use __sync_threads() for barrier synchronization we make sure all the threads in a block are alive even though they might be out of the image.

3. Experimental Results

3.1. Setup and timings

We utilized synthetic images and natural images from the Berkeley segmentation data set (BSDS500) which con-
Algorithm 2  Nonmaximum suppression and hysteresis thresholding used in Algorithm 1

\begin{algorithm}
\begin{procedure}{\text{GetOrientationSector}((v_x, v_y))}
\text{Returns the orientation sector $s_\theta$ for the 2D vector $(v_x, v_y)^T$.}
\begin{align*}
\begin{pmatrix}
  v'_x \\
  v'_y
\end{pmatrix} &= \begin{pmatrix}
  \cos (\pi/8) & -\sin (\pi/8) \\
  \sin (\pi/8) & \cos (\pi/8)
\end{pmatrix} \cdot \begin{pmatrix}
  v_x \\
  v_y
\end{pmatrix} \\
\text{if } v'_y < 0 & \text{ then} \\
  v'_x & \left\{ \begin{array}{ll}
  0 & \text{if } (v'_x \geq 0) \land (v'_y \geq v'_y) \\
  1 & \text{if } (v'_x \geq 0) \land (v'_y < v'_y) \\
  2 & \text{if } (v'_x < 0) \land (v'_x < v'_y) \\
  3 & \text{if } (v'_x < 0) \land (v'_x \geq v'_y)
\end{array} \right.
\end{align*}
\end{procedure}
\begin{procedure}{\text{IsLocalMax}(E_{mag}, x, y, s, t_{lo})}
\text{Determines if $E_{mag}$ is a local maximum at position $(x, y)$ in direction $s_\theta \in \{0, 1, 2, 3, 4\}$.}
\begin{align*}
m_C & \leftarrow E_{mag}(x, y) \\
\text{if } m_C & \geq t_{lo} \text{ then} \quad \text{return } \text{false} \\
\text{else} & \\
(m_L, m_R) & \leftarrow \\
\begin{pmatrix}
  E_{mag}(x-1, y) , E_{mag}(x+1, y)
\end{pmatrix} & \text{if } s_\theta = 0 \\
\begin{pmatrix}
  E_{mag}(x-1, y-1) , E_{mag}(x+1, y+1)
\end{pmatrix} & \text{if } s_\theta = 1 \\
\begin{pmatrix}
  E_{mag}(x, y-1) , E_{mag}(x, y+1)
\end{pmatrix} & \text{if } s_\theta = 2 \\
\begin{pmatrix}
  E_{mag}(x-1, y+1) , E_{mag}(x+1, y-1)
\end{pmatrix} & \text{if } s_\theta = 3 \\
\text{return } (m_L \leq m_C) \land (m_C \geq m_R).
\end{align*}
\end{procedure}
\begin{procedure}{\text{TraceAndThreshold}(E_{nms}, E_{bin}, x_0, y_0, t_{lo})}
\text{Reversely collects and marks all pixels of an edge that are 8-connected to $(x, y)$ and exhibit a gradient magnitude above $t_{lo}$.}
\begin{align*}
E_{bin}(x_0, y_0) & \leftarrow 1 \\
x_L & \leftarrow \max(x_0 - 1, 0) \\
x_R & \leftarrow \min(x_0 + 1, m - 1) \\
y_T & \leftarrow \min(y_0 - 1, 0) \\
y_B & \leftarrow \min(y_0 + 1, n - 1) \\
\text{for } x & \leftarrow x_L, \ldots, x_R \text{ do} \\
\text{for } y & \leftarrow y_T, \ldots, y_B \text{ do} \\
\text{if } (E_{nms}(x, y) \geq t_{lo}) \land (E_{bin}(x, y) = 0) & \text{ then} \quad \text{TraceAndThreshold}(E_{nms}, E_{bin}, x, y, t_{lo}) \quad \text{return}
\end{align*}
\end{algorithm}

Table 1. Runtime in seconds for MATLAB, C, CUDA and CUDA pre opt, for different image resolution and iterations.

|                | 512x512-1 | 512x512-50 | 1920x1080-1 | 1920x1080-50 | 6600x4400-1 | 6600x4400-50 |
|----------------|-----------|------------|-------------|-------------|-------------|-------------|
| MATLAB         | 0.13      | 4.8        | 1.19        | 50          | 16.7        | 646         |
| C              | 0.21      | 7.2        | 1.28        | 46          | 20          | 780         |
| CUDA           | 1.39      | 1.75       | 1.4         | 3.4         | 2.24        | 23          |
| CUDA pre opt   | 1.82      | 2.3        | 1.85        | 5.4         | -           | -           |

We use a synthetic image \textit{Fish} of size $512 \times 334$ for testing the accuracy of different edge detectors under different noise levels [6]. We compare the edge detection results on noise-free, and noisy (obtained by adding Gaussian noise of standard deviation $\sigma_n = 30$) images with other edge detectors [1], [2], [3] with default parameters. As can be seen from the Figure 4 bottom row, under noisy conditions all the traditional edge detectors pick up false positive edges while our CSANG obtained a robust result.

3.2. Comparison with other edge detectors

We compare our fast CSANG with traditional edge detectors due to Prewitt [1], Sobel [2], and Canny [3].

3.2.1. Synthetic images. We use a synthetic image \textit{Fish} of size $518 \times 334$ for testing the accuracy of different edge detectors under different noise levels [6]. We compare the edge detection results on noise-free, and noisy images with other edge detectors [1], [2], [3] with default parameters. As can be seen from the Figure 5(b), and Canny edge detection results were obtained with default parameters. As can be seen by comparing the edge detection results obtained with our CSANG (Figure 5(c)) we obtained better results. In Canny edge detection results we can observe lots of spurious edges while our approach obtained less spurious edges, however edge connectivity is lost in some places.

4. Conclusions

In this paper, we utilized a fast adaptive robust structure tensor for edge detection in digital images. By utilizing continuous scale anisotropic gaussians via Geman-McClure robust error estimator function we obtained efficient edge detection for noisy images. We provide an efficient GPU implementation using CUDA, and our implementation can obtain $30 \times$ speed improvement over traditional CPU implementations. Compared to different edge detectors from the
Figure 4. Comparison of different edge detectors on noise-free and noisy images. (a) Input images, (b) Prewitt [1], (c) Sobel [2], (d) Canny [3], (e) ARST. (Top row) Noise-free, (bottom row) noisy image obtained with adding Gaussian noise of standard deviation $\sigma_n = 30$.

Figure 5. Comparison on Berkeley segmentation data set (BSDS500). (a) Input images, (b) ground truth (GT), and edge detection results with (d) Canny edge detector [3], (e) our CSANG.
literature, our ARST based edge detector provides robust and accurate results. Experimental results on a variety of noisy images indicate we obtain better edge detection results and compared to other standard edge detectors. Testing other robust error estimator functions and applying the fast ARST to large scale image segmentation defines our future work.

References

[1] J. M. S. Prewitt, “Object enhancement and extraction,” in Picture processing and Psychopictorics. New York, NY, USA: Academic Press, 1970, pp. 75–149.

[2] I. E. Sobel, “Camera models and machine perception,” Ph.D. dissertation, Stanford University, USA, August 1970.

[3] J. F. Canny, “A computational approach to edge detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679–698, 1986.

[4] V. B. S. Prasath, D. Vorotnikov, R. Pelapur, S. Jose, G. Seetharaman, and K. Palaniappan, “Multiscale Tikhonov-total variation image restoration using spatially varying edge coherence exponent,” IEEE Transactions on Image Processing, vol. 24, no. 12, pp. 5220–5235, 2015.

[5] V. B. S. Prasath, R. Pelapur, G. Seetharaman, and K. Palaniappan, “Multiscale structure tensor for improved feature extraction and image regularization,” Submitted, 2017.

[6] S. Nath and K. Palaniappan, “Adaptive robust structure tensors for orientation estimation and image segmentation,” Lecture Notes in Computer Science (ISVC), vol. 3804, pp. 445–453, 2005.

[7] M. J. Black and A. Rangarajan, “On the unification of line processes, outlier rejection, and robust statistics,” International Journal of Computer Vision, vol. 19, no. 1, pp. 57–91, 1996.

[8] R.van den Boomgaard and J.van de Weijer, “Robust estimation of orientation for texture analysis,” in 2nd Intl. Work. Text. Anal. Synt., Copenhagen, Denmark, 2002, pp. 135–138.