On the use of Hough transform for context-based image compression in hybrid raster/vector applications

Pasi Fränti[1], Eugene Ageenko[1], Saku Kukkonen[2] and Heikki Kälviäinen[2]

[1] *Department of Computer Science*
University of Joensuu
P.O. Box 111, FIN-80101 Joensuu, FINLAND
Email: franti,ageenko@cs.joensuu.fi

[2] *Department of Information Technology,*
Lappeenranta University of Technology
P.O. Box 20, FIN-53851 Lappeenranta, FINLAND
Email: Heikki.Kalviainen,Saku.Kukkonen@lut.fi

Abstract: In a hybrid raster/vector system, two representations of the image are stored. Digitized raster image preserves the original drawing in its exact visual form, whereas additional vector data can be used for resolution-independent reproduction, image editing, analysis and indexing operations. We introduce two techniques for utilizing the vector features in context-based compression of the raster image. In both techniques, Hough transform is used for extracting the line features from the raster image. The first technique utilizes the line features to improve the prediction accuracy in the context modeling. The second technique uses a feature-based filter for removing noise near the borders of the extracted line elements. This improves the image quality and results in more compressible raster image. In both cases, we achieve better compression performance.

1. Introduction

In a hybrid raster/vector storage system, both raster and vector representations of the images are encoded and stored [1,2] see Fig. 1. The raster representation provides an exact digitized replica of the original image. The vector representation contains semantic information extracted from the image. It benefits from vector editing capabilities and is suitable for further image processing and semantic analysis [3,4]. The compressed file consists of the extracted line features and the compressed raster image.

The advantage of raster representation is that the images can be easily digitized and stored compactly using latest compression technology. Vector representation, on the other hand, allows better editing capabilities and resolution independent scaling and reproduction. Complete raster-to-vector conversion, however, is not a realistic solution because the conversion systems are of high complexity and they cannot capture all possible vector features reliably without human interaction. Either the file will be filled by huge number of small vector elements, or some of the undetected information will be lost.

We consider the storage problem of hybrid raster/vector systems. In an ideal situation, all background features would be in raster format and all line features in vector format. In practice, hybrid raster/vector representation means that a lot of new data will be stored in the vector format without any saving in the storage of the raster image.

In this paper, we introduce two novel techniques for utilizing the vector features in context-based image compression of the raster image. In both techniques, we use Hough transform [5,6] for extracting the line features from the raster image. The first technique utilizes the line features to improve the prediction accuracy in the context modeling. The compressed file consists of the extracted line features and the compressed raster image.

The second technique uses a feature-based filter for removing the noise near the borders of the extracted line *elements*. This improves the image quality and results in a more compressible raster image. The filtering is based on a simple noise removal procedure using a mismatch image between the original and the feature image. The method is near-lossless because the amount of changes is controlled – only isolated noise pixels are reversed.

2. Context-based compression

Binary images are favorable source for context-based image compression because of the spatial dependencies between neighboring pixels [7,8]. In context-based compression, the pixels are coded on the basis of their probability estimates in respect to the *context*. The context is defined by the combination of the color values of already coded neighboring pixels within the template.

JBIG is the current international standard for compression of the bi-level images in communications [9,10]. In JBIG, the image is coded by default in raster scan order using context-based probability model and adaptive arithmetic coder, namely *QM-coder*. The probability estimation in the QM-coder is derived from the arithmetic coder renormalization [11]. Instead of maintaining pixel counts, the estimation process is implemented as a state automaton consisting of 226 states. The backward-adaptive modeling of JBIG has the advantage that only one pass over the data is required and no overhead (models or code tables) needs to be stored in the compressed file.

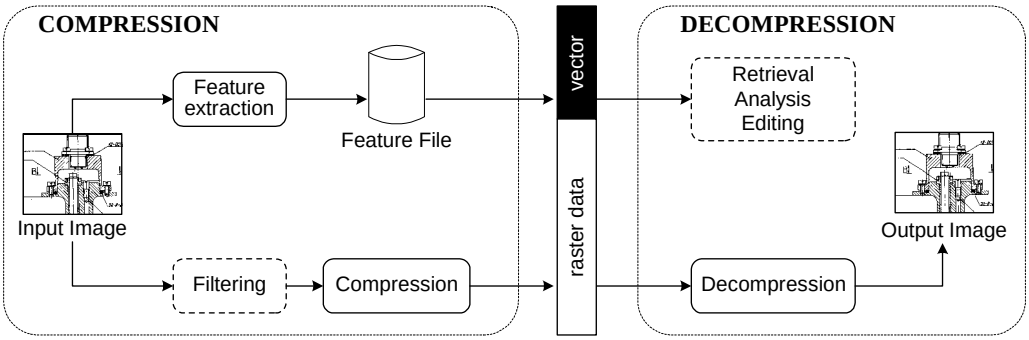

Fig.1. Hybrid raster/vector storage system

The emerging standard JBIG2 [12-14] improves the compression of text images using pattern matching technique for extracting symbols from the image. This enhancement, however, is of limited usage in the case of line-drawing images, as they do not contain large number of text elements.

The context modeling of JBIG can also be improved using variable-size context template [15,16]. The contexts are stored in the leaves of a variable-depth binary tree, referred as *context tree*. The use of variable-size context model enables selective context expansion and utilizes larger context templates without overwhelming the learning cost.

Another way to improve compression is to filter the image for noise removal. Filtering reduces irregularities in the image caused by noise, and in this way, makes the image more compressible without degrading the image quality. Noise appears in the image as randomly scattered noise pixels (additive noise), and as content-dependent noise distorting the contours of printed objects (lines, characters) by making them ragged.

Several methods have been considered in literature for image pre-processing before the compression [17-20]. These filtering methods work by analyzing local pixel neighborhood defined by a filtering template and include logical smoothing, variations of median filtering, isolated pixel removal, and morphological filters [21]. Recent research in mathematical morphology have shown that morphological filtering can be used as an efficient tool for pattern restoration in environment of heavy additive noise [22-25]. Such approaches, however, are not necessarily suitable for filtering content-dependent quantization noise. Another problem is that the filtering may destroy fine image structures carrying crucial information if the amount of filtering is not controlled.

3. Feature extraction using Hough transform

Hough transform is used for extracting the vector features from the image [5,26,27], as summarized in Fig. 2. The motivation is to find rigid fixed length straight lines in the image. The extracted line segments are represented as their end-points. A feature image is reconstructed from the line segments and it is utilized in the compression phase. The extracted line segments are also stored in the compressed file.

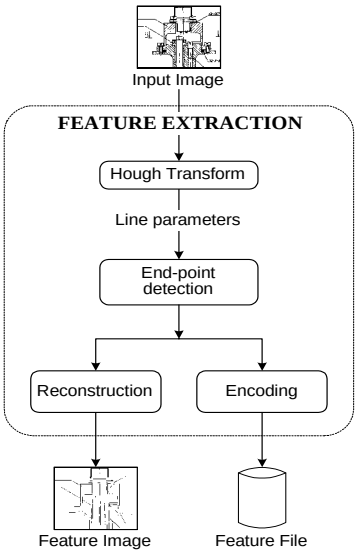

Fig. 2. Block diagram of the feature extraction.

3.1. Hough transform

The lines are first detected by the Hough transform (HT) as follows:

1. Create a set of coordinates from the black pixels in the image.
2. Transform each coordinate (x, y) into parameterized curve in the parameter space.

3. Increment the cells in the parameter space determined by the parametric curve.
4. Detect local maxima in the accumulator array. Each local maximum may correspond to a parametric curve in the image space.
5. Extract the curve segments using the knowledge of the maximum positions.

The parameter space is a $k \times k$ accumulator array where k can be tuned according to the image size, e.g. k = the size of the image. The slope-intercept (ρ,θ) parameterization is used. The accumulation matrix is quantized with equal intervals.

3.2. End-point detection

The Hough transform is capable to determine the location of a line (as a linear function) but it cannot resolve the end-points of the line. In fact, HT does not even guarantee that there exists any finite length line in the image but it only indicates that the pixels (x, y) along $y = a \cdot x + b$ may represent a line. The existence of a line segment must therefore be verified. The verification is performed by scanning the pixels along the line and checking whether they meet certain criteria. We use the scanning width, the minimum number of pixels, and the maximum gap between pixels in a line as the selection criteria. If predefined threshold values are met, a line segment is detected and its end-points are recorded.

3.3. Reconstruction of the feature image

A feature image of equal size is created from the extracted line segments to approximate the input image. The image is constructed by drawing one-pixel width straight lines using the end-points of the line features. The Hough transform does not determine the widths of the lines but wider lines are represented by a bunch of collinear line segments. The line segments may also be deviated from their original direction and/or have one-pixel positional error because of the quantization of the accumulation matrix. Therefore we do not utilize the feature image directly but process it first by consequent operations of morphological *dilation* and *closing* [23]. These operations make the lines one pixel thicker in all directions (dilation) and fill gaps between the line segments (closing). We apply a symmetric 3×3 structure element (*Block*) for the dilation, and a 3×3 cross structure element (*Cross*) for the closing. The cross element is chosen to minimize the distortion in line intersections caused by closing.

3.4. Storing the line segments

The extracted line segments are stored as $\{(x_1,y_1), (x_2,y_2)\}$ representing the end-points of the line. A single coordinate value takes $\lceil \log_2 n \rceil$ bits where n is the dimension of the image. For example, a line in an image of 4096×4096 pixels takes $4\times12 = 48$ bits in total. A more compact representation could be achieved if the line segments are sorted according to their first coordinate x_1. Instead of storing the absolute value, we could store the difference between two subsequent x_1's. Most of the differences are very small (about 40 % of them are in the range 0..2). An improvement of about 7 bits (from 12 to 5 bits) was estimated when entropy coding was applied to these difference values. In the present implementation, this idea was not applied.

4. Feature-based context modeling

There are two basic approaches for utilizing the feature image: (1) lossless compression of the residual between the original and the feature image, (2) compression of the original image using the feature image as side information. The first approach does not work in practice because taking the residue destroys spatial dependencies near the borders of the extracted line features. The residual image is therefore not any easier to compress than the original one. On the other hand, the effectiveness of the second approach has been proven in practice in the case of text images [10,12]. We thus adopt the same idea here for line drawing images.

The compression method denoted further as *HTC* is outlined in Fig. 3. The original image is compressed using the JBIG-like technique, which uses previously coded neighboring pixel as context. The context is determined by combining index out of the neighboring pixel values and accessing to the model using a look-up table. Additional context pixels are taken from the feature image. An important point is that any pixel in the feature image can be utilized, even the current pixel that is to be compressed. Here we use ten pixels from the original image as in three-line JBIG modeling and five pixels from the feature image, see Fig. 4. The actual coding is performed by QM-coder, the binary arithmetic coder of JBIG [9]. The line features are also stored in the compressed file.

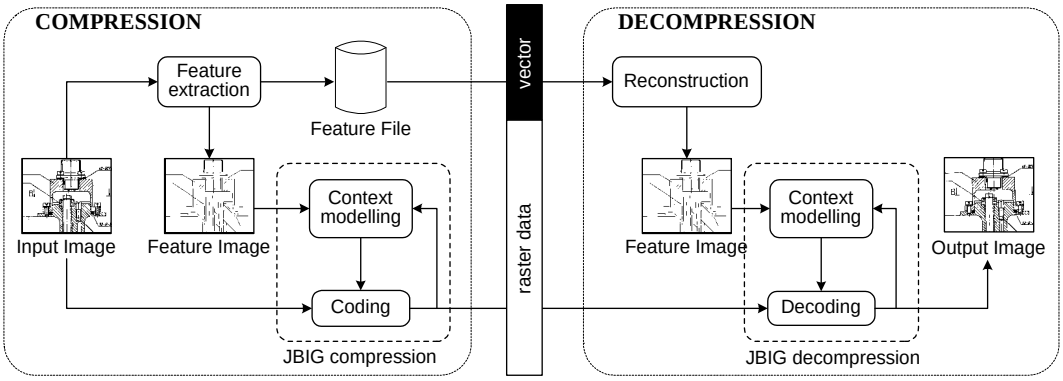

Fig. 3. Block diagram of the new hybrid compression system.

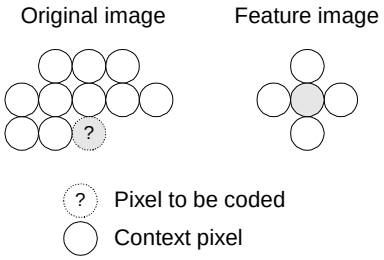

Fig. 4. Illustration of the two-level context template.

5. Feature-based filtering

The compression method utilizing feature-based filtering and denoted further as *HTF-JBIG* is outlined in Fig. 5. The image is preprocessed by a feature-dependent filtering for improving the image quality. The filtering removes noise by the restoration of the line contours and therefore it results in better compression performance. The line features are used only in the compression phase and therefore they need not be stored in the compressed file. The filtered image is compressed by the JBIG without any modifications. Decompression is exactly the same as the JBIG.

The filtering is based on a simple noise removal procedure, as shown in Fig. 6. A difference (mismatch) image between the original and the feature image is constructed. Isolated mismatch pixels (and groups of two mismatch pixels defined by 8-connectivity) are detected and the corresponding pixels in the original image are reversed. This removes random noise and smoothes edges along the detected line segments. The method is near-lossless because the amount of changes is controlled – only isolated noise pixels are reversed. Undetected objects (such as text characters) are left untouched allowing their lossless reconstruction.

The noise removal procedure is successful if the feature image is accurate. However, the feature extraction of HT does not always provide exact width of the lines. The noise removal procedure is therefore iterated three times as shown in Fig. 7. The first stage applies the feature image as such, but the feature image is dilated in the 2nd stage and eroded in the 3rd stage before input into the noise removal procedure. This compensates inaccuracies in the width detection.

The stepwise process is illustrated in Fig. 8. Most of the noise is detected and removed in the first phase. However, the rightmost diagonal line in the feature image is too wide and its upper contour is therefore filtered only in the third stage where the feature image is eroded. The results of the entire filtering process are illustrated in Fig. 9. In these examples, pixel-level noise is mainly filtered out but some of the roughness remains along the lines. These consist of larger groups of noise pixels and therefore are not filtered by the method. Symbols and other non-linear elements are not completely detected by Hough transform, and therefore parts of them may have not been processed.

It is noted that the noise removal procedure does not guarantee the retention of connectivity of the lines. It is therefore possible that very thin lines may be broken apart because of a pixel removal. Although the situation is rare, the retention of connectivity can be important in some application. In this case, an additional procedure must be applied to check whether pixel removal would break connectivity. For example, the method in [30] can be implemented by a simple look-up table consisting of the pixels within the 3×3 neighboring.

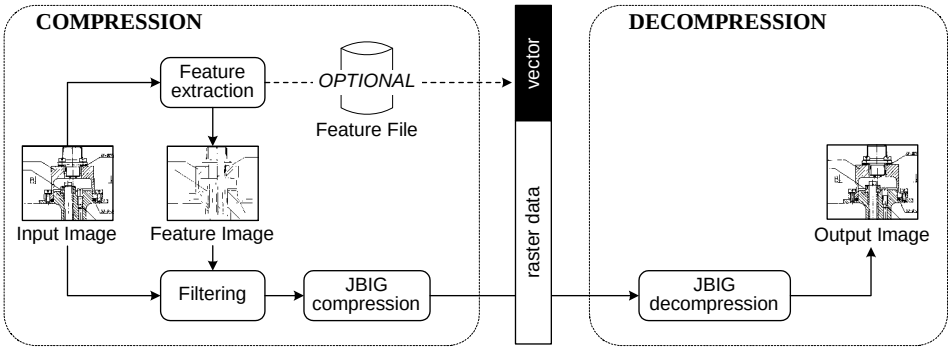

Fig. 5. Block diagram of the near-lossless compression system.

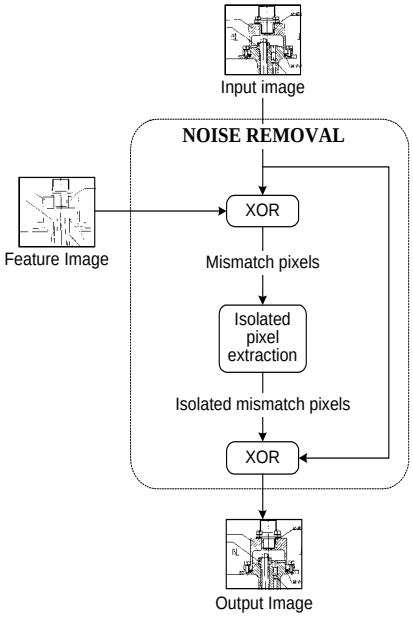

Fig. 6. Block diagram of the noise removal procedure.

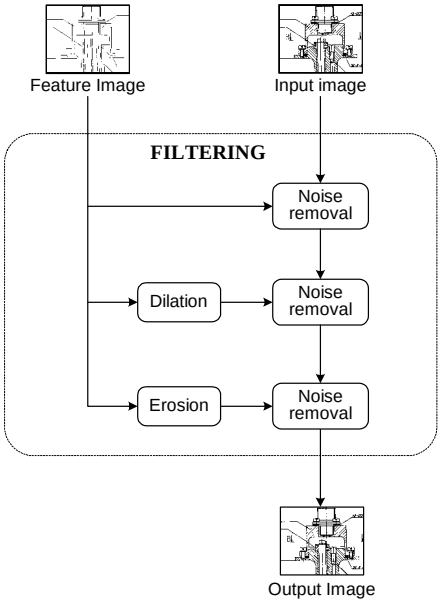

Fig. 7. Block diagram of the three-stage filtering procedure.

6. Test results

The performance of the proposed methods is tested by compressing the set of test images of Fig. 10. Three different feature sets were constructed from each image with different amount of line segments (sets 1, 2, and 3). The number of extracted lines was controlled by varying the parameters in the Hough transform.

The effect of the feature-based context modeling on the file size is shown in Fig. 11. The feature-based context modeling improves the compression of the raster image of about 1 to 10 %, depending on the image and the number of extracted line elements. The amount of saving, however, is too small to compensate the overhead required by the feature file. For example, the vector data requires 6.3 kilobytes for the image *Bolt*, and the size of the raster data can be reduced from 12.7 to 11.2 kilobytes. In total, the file takes 17.5 kilobytes.

The second method (HTF-JBIG) applies feature-based filtering for noise removal and standard JBIG for image compression. In this case, the number of extracted line features does not affect the file size because the features are not stored. It is therefore better to use as many features as can be reliably detected. In our case, the set 3 (most line segments) gives the best results among the three tested sets. The method improves the compression performance of about 12 % on average, in comparison to JBIG. For example, the image *Bolt* requires 10.3 kilobytes in comparison to 12.7 of JBIG, or 17.5 of the hybrid compression.

The storage sizes of the two proposed methods are summarized in Table 1. In a hybrid compression both raster and vector data are included in the compressed file. The column “vector” contain the storage size required by the vector features. The two columns “raster” refers to the two alternatives for compressing the raster image: using JBIG and using

the proposed HTC method with feature-based context modeling. The table includes also results (last column) where the new techniques have both been utilized at the same time. In the case of image *Bolt*, the size of the raster decreases to 9.1 kilobytes but in this case, the vector data must also be stored.

The running times of the proposed methods in total for the test set using Pentium-200 machine are shown in Fig. 12. The feature extraction dominates the running time in the compression phase and makes it an order of magnitude slower than JBIG. The method is therefore suitable only for applications where compression can be made off-line. Decompression of filtered images in HTF-JBIG is performed using the standard JBIG routines and therefore is as fast as JBIG. In the hybrid HTC compression, the decompression phase is about 35 % slower because of the processing of the vector features.

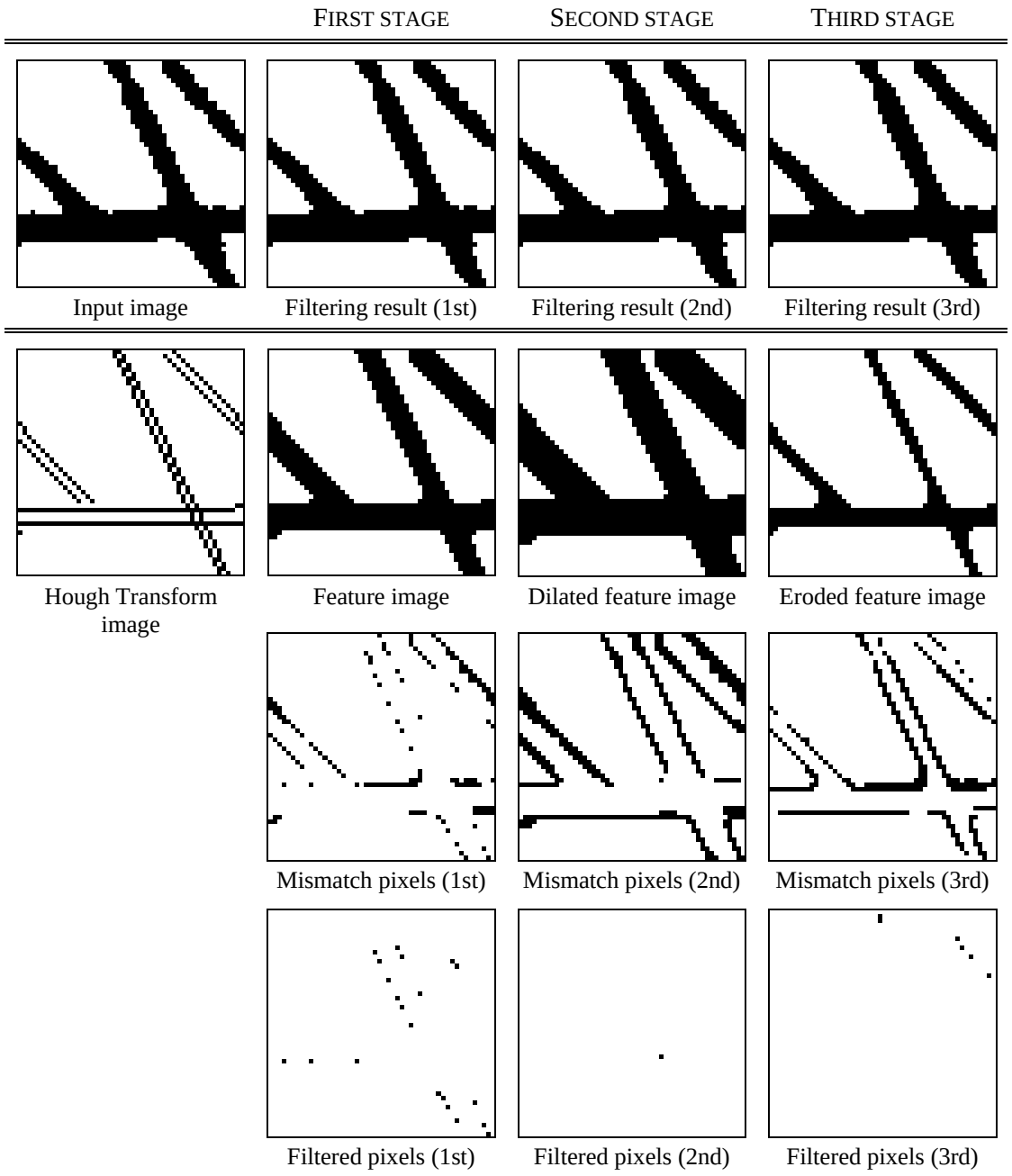

Fig. 8. Illustration of the three-stage filtering procedure.

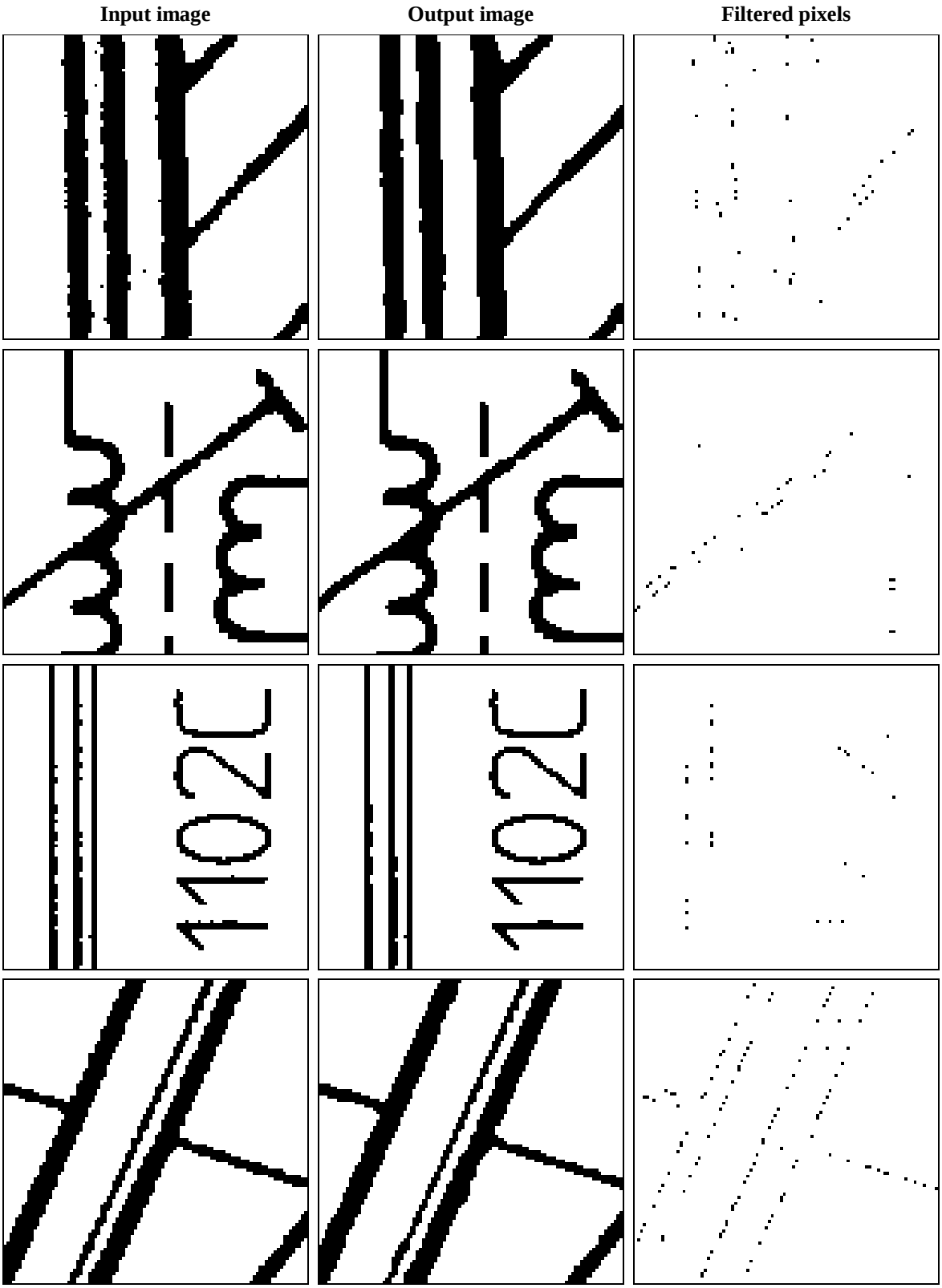

Fig. 9. Filtering examples from left to right: sample from the original image, from the filtered image, and their difference.

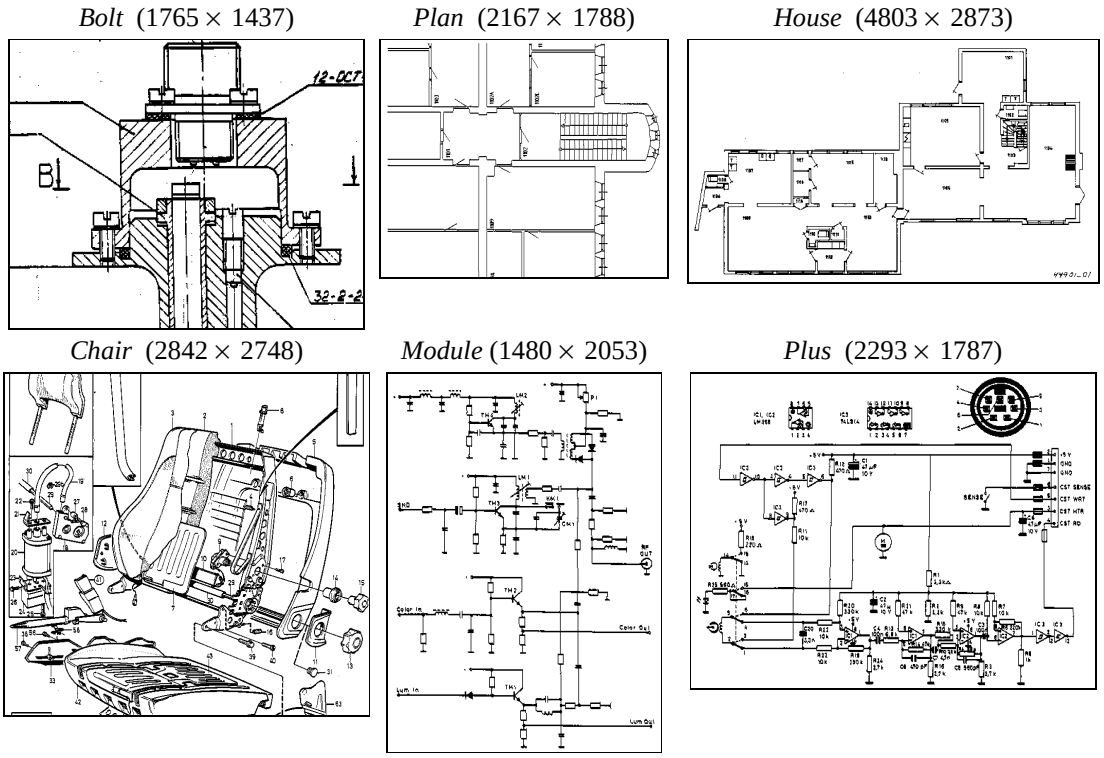

Fig. 10. Set of test images.

Table 1. Summary of the storage sizes of the different methods (in bytes).

Image	Hybrid compression			Filtering only	Filtering + Hybrid
	vector	raster (JBIG)	raster (HTC)	(HTF-JBIG)	(HTF-HTC)
BOLT	6,438	12,966	11,514	10,536	9,287
PLAN	2,370	5,098	4,578	4,325	3,786
HOUSE	13,398	15,688	13,961	13,336	11,553
CHAIR	16,710	52,384	50,140	51,529	48,023
MODULE	3,468	7,671	7,222	6,431	6,057
PLUS	5,268	17,609	17,132	16,273	15,739
TOTAL	47,652	111,416	104,547	102,430	94,445

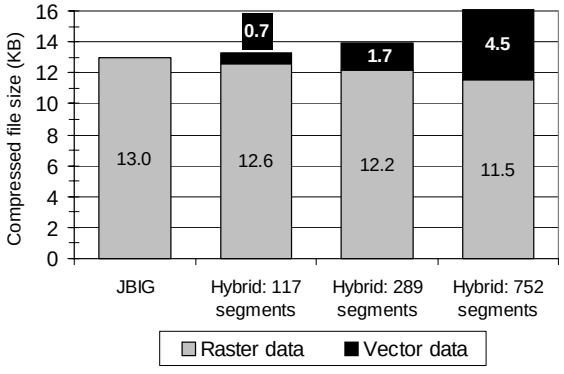

Fig. 11. Illustration of the compressed file sizes for *Bolt* with variable amount of line elements.

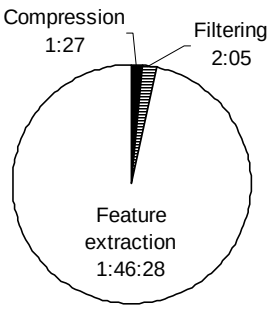

Fig. 12. Running times of the HT-based compression.

7. Conclusions

Two methods were introduced for improving compression performance in hybrid raster/vector applications. The first method uses the feature image as side information but the improvement is found to be too small to compensate the overhead required by the feature file.

The second method applies feature-based filtering for removing noise from the original image. It improves the image quality and results in about 12 % improvement in the compression. At the same time, the quality of the decompressed images is visually the same (or even better) because the reversed pixels are mainly random noise or scanning noise near the line segments.

Overall, the benefit of utilizing feature-based information was moderate at best, and cannot compensate the increase in the storage size caused by the inclusion of vector features. We therefore conclude that we must either give up the requirement of preserving exact replica of the original image, or improve the quality of the vectorizing drastically if the we want to store the hybrid file structure efficiently.

References

1. Wilson D.J., S.E.A. Your paper. Scan Edit Archive Initiative, 1999. http://www.sea-initiative.org/
2. Fränti P., Ageenko E.I., Kälviäinen H., Kukkonen S., Compression of line drawing images using Hough transform for exploiting global dependencies, *Proc. 4th Joint Conf. on Information Sciences (JCIS'98)*, RTP, USA, IV: 433-436, 1998.
3. Kasturi R., et al., A system for interpretation of line drawings. *IEEE Trans. on Pattern Analysis, Machine Intelligence* 12(10): 978-992, 1990.
4. Kasturi R., O'Gourman L., Document image analysis: A bibliography. *Machine Vision, Applications* 5: 231-243, 1992.
5. Hough P.C.V., Methods and means for recognizing complex patterns. U.S. Patent 3,069,654, 1962
6. Kälviäinen H., Hirvonen P., Xu L., Oja E., Probabililistic, non-probabilistic Hough transforms: overview and comparisons. *Image, Vision Computing* 13: 239-251, 1995.
7. Langdon G.G., Rissanen J., Compression of black-white images with arithmetic coding. *IEEE Trans. Communications* 29 (6): 858-867, 1981.
8. Tisher P.E., Worley R.T., Maeder A.J., Goodwin M., Context-based lossless image compression. *The computer Journal* 36: 68-77, 1993.
9. ISO/IEC International Standard 11544. *ISO/IEC/JTC1/SC29/WG9*; also ITU-T Recommendation T.82. Progressive Bi-level Image Compression, 1993.
10. Witten I.H., Moffat A., and Bell T.C., *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.
11. Pennebaker W.B., Mitchell J.L., Probability estimation for the Q-coder. *IBM Journal of Research, Development* 32 (6): 737-759, 1998.
12. Howard P.G., Text image compression using soft pattern matching. *Comp. Journal* 40: 146-156, 1997.
13. Howard P.G., Kossentini F., Martins B., Forchammer S., and Rucklidge W. J., The emerging JBIG2 standard. *IEEE Trans. Circuits, Systems for Video Technology* 8 (7): 838-848, 1998.
14. JBIG2 Working Draft, 1999. http://www.jpeg.org/public/jbigpt2.htm
15. Fränti P. and Ageenko E.I., On the use of context tree for binary image compression. *IEEE Proc. Int. Conf. on Image Processing* (*ICIP'99*), Kobe, Japan, 1999.
16. Martins B. and Forchhammer S., Bi-level image compression with tree coding. *IEEE Transactions on Image Processing*, 7 (4): 517-528, 1998.
17. Ting D. and Prasada B., Digital processing techniques for encoding of graphics. *Proc. of the IEEE* 68 (7): 757-769, 1980.
18. Bernstein R., Adaptive nonlinear filters for simultaneous removal of different kinds of noise in images. *Proc. IEEE Tran. on Circuits and Systems* CAS-34 (11): 1275-1291, 1987.
19. Algazi V.R., Kelly P.L., and Estes R.R., Compression of binary facsimile images by preprocessing and color shrinking. *IEEE Trans. on Communications* 38 (9): 1592-1598, 1990.
20. Zhang Q. and Danskin J.M., Bitmap reconstruction for document image compression. *SPIE Proc. Multimedia Storage, Archiving Systems*, Boston, MA, Vol. 2916: 188-199, 1996.
21. Serra J., Image Analysis and Mathematical morphology. London: Academic Press, 1982.
22. Schonfeld D. and Goutsias J., Optimal morphological pattern restoration from noisy binary images. *IEEE Trans. on Pattern Analysis, Machine Intelligence* 13 (1): 14-29, 1991.
23. Heijmans H.J.A.M., Morphological image operators. Boston: Academic Press, 1994.
24. Koskinen L. and Astola J., Soft morphological filters: A robust morphological filtering method. *Journal of Electronic Imaging* 3: 60-70, 1994.
25. Dougherty E.R. and Astola J. (eds), Nonlinear Filters for Image Processing, SPIE Optical Engineering Press, 1997.
26. Leavers V.F., Survey: Which Hough Transform. *CVGIP Image Understanding* 58 (2): 250-264, 1993.
27. Parker J.R. *Algorithms for image processing and computer vision*. John Willey & Sons, 1996
28. Zhang and J.M. Danskin, Bitmap reconstruction for document image compression. *SPIE Proc. Multimedia Storage and Archiving Systems*, Boston, MA, Vol. 2916: 188-199, 1996.
29. Pennebaker and J.L. Mitchell, *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.
30. Yokoi S., Toriwaki J., and Fukumura T., Topological properties in digitized binary pictures. *Systems Computer Controls*. Vol. 4: 32-39, 1973.