Available online at www.sciencedirect.com

SCIENCE DIRECT®

Real-Time Imaging 9 (2003) 289–296

www.elsevier.com/locate/rti

Real-time surface inspection by texture

Topi Mäenpää*, Markus Turtinen, Matti Pietikäinen

Machine Vision Group, Department of Electrical and Information Engineering, Infotech Oulu, P.O. Box 4500, University of Oulu, FIN-90014, Finland

Abstract

In this paper a real-time surface inspection method based on texture features is introduced. The proposed approach is based on the Local Binary Pattern (LBP) texture operator and the Self-Organizing Map (SOM). A very fast software implementation of the LBP operator is presented. The SOM is used as a powerful classifier and visualizer. The efficiency of the method is empirically evaluated in two different problems including textures from the Outex database and from a paper inspection problem.

1. Introduction

The inspection of surface texture is an important part of many industrial quality control applications. Even though color is a commonly used cue, it is not always enough, or even available. For instance in wood surface inspection, texture features can be used to enhance the accuracy of color-based defect detection [1]. Due to the lack of color information, many applications, like textile [2,3], steel [4], and paper [5] inspection, must however be based solely on texture information.

In most industrial inspection systems, speed is a critical issue. The analysis of a digital image must be completed in a tight time frame so that the production system can act based on the measures. Thus, both feature extraction and classification must be performed quickly.

In real-world applications, feature distributions are very seldom "well-behaved", rendering parametric classification inapplicable. On the other hand, the large amount of training data typically needed for a good accuracy renders non-parametric approaches like the k-NN classifier unusable as such. Artificial neural networks are a commonly used solution to this problem. In addition to neural networks, many classifiers commonly used in industrial inspection systems utilize rule-base methods. The idea is to find the threshold values and parameters that produce the best classification accuracy. Usually, human supervision is needed in controlling this procedure, which makes it somehow subjective and prone to human error. Although the classification itself might be fast (depending on the amount of rules and other parameters) the teaching and parameter adjustment phase is laborious.

Until recently, Gabor filtering has been considered the state-of-the-art in texture analysis by many researchers (see e.g. [6,7]). It has also been applied to visual inspection (see e.g. [8]). However, as Kumar and Pang note, the use of a multi-channel filtering scheme in a real-time application calls for additional DSP hardware, which is not a cost-effective solution [3].

In this paper we propose a software-based analysis system capable of processing large images at video rates and above on a standard, low-cost PC. The approach utilizes powerful micro-texton distributions provided by the LBP operator [9], for which we present a very fast software implementation. Feature sets are optimized with the SFFS algorithm [10], and real-time classification performance is achieved through the use of self-organizing maps [11].

2. Optimization for real time

The optimization of a real-time analysis method can happen in a number of application-specific ways. Here, we roughly divide these into three categories, namely feature extraction, feature set, and classification optimization, each of which is handled in the following sections.

*Corresponding author.

E-mail addresses: topiolli@ees2.oulu.fi (T. Mäenpää), dillian@ee.oulu.fi (M. Turtinen), mkp@ee.oulu.fi (M. Pietikäinen).

1077-2014/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S1077-2014(03)00041-X

2.1. Optimized feature extraction

The LBP operator measures locally binarized texture patterns by thresholding a circular neighborhood with the value of the center pixel. In the general definition of the operator, the radius of the neighborhood and the number of samples in it can take arbitrary values [9]. Here we consider only the eight neighbors of a pixel, i.e. a neighborhood with eight samples and a radius of one. Following the convention of Ojala et al. [9], this operator is denoted by $LBP_{8,1}$. Since rotation invariance was not needed in the experiments, and for maximum speed, the bilinear interpolation of diagonal neighbors suggested by Ojala et al. was not used. Instead, the values of the diagonal pixels were used as such.

In Fig. 1, an example of calculating the non-interpolated $LBP_{8,1}$ is shown. The original neighborhood at the left is thresholded by the value of the center pixel, and a binary pattern code is produced by interpreting the neighbors circularly as a binary number. The distribution of pattern codes is used as a texture descriptor.

When implementing the LBP operator for a general-purpose CPU, there are a number of issues affecting the computational performance that must be addressed. A decent microprocessor can nowadays perform hundreds of millions of instructions in a second. The full capability cannot however be utilized if proper care is not taken. In CISC processors, like the present PC processors, the number of micro-operations per instruction varies—some instructions consume more clock cycles than others. Also, to exploit the full capability of the pipeline, the number of conditional jumps in the code must be minimized. Finally, the number of memory accesses should be kept as low as possible. At least it must be ensured that only very few memory references miss the first-level cache.

An apparent problem with the LBP operator is that each pixel value in a neighborhood must be compared to the value of the center. With P neighbors this causes, on average, $P/2$ pipeline stalls per pixel, because the comparison result cannot be predicted. The operator can however be implemented without conditional jumps, as will be explained in the following. Also, the CISC instructions used are all one-cycle operations, with the exception of memory-to-register and register-to-memory transfers.

The trick used in eliminating conditional branching is based on processors' internal representation of integer values, called two's complement. A negative number is created by inverting each bit of the corresponding positive number (one's complement) and then adding one to the result, ignoring possible overflows. The result of this is that the MSB of a binary number tells the sign of the number—negative numbers have their MSB set to one.

To build an LBP code, one needs to set a bit in a binary number depending on whether a value in a neighborhood is larger than the center pixel. Instead of comparing these values, one can utilize their difference. When the value of a neighbor is subtracted from the value of the center, the MSB of the result can be directly used in building the LBP code. Care must however be taken to ensure no overflows or underflows occur when calculating the difference. That is, the calculation must be performed with a sufficiently large number of bits. Furthermore, one must be subtracted from the difference so that a neighbor value equal to the center produces the sign bit.

Fig. 2 shows an example of calculating a four-bit LBP code. The neighborhood on the left is traversed counter-clockwise, starting at "3". The signs of the differences are extracted by a logical AND operation with 1000_2. The sign is then shifted to the right 3—*Index* times. Finally, the LBP code is obtained with a logical OR operation of the shifted signs.

The C implementation of the $LBP_{8,1}$ operator (with no interpolation) listed in the appendix was used in all subsequent experiments. On a 1 GHz processor, the implementation achieves video rates (24 images/s) with images as large as 960×960 pixels, if the images have been prefetched into memory. It takes only 20 ms to process a prefetched paper image, used in experiment #2. Processing an Outex image, used in experiment #1, takes 0.03 ms.

2.2. Feature selection

Good features are crucial to classification in terms of both classification accuracy and computational performance. With a good subset of a large number of features, it is possible to reduce both redundancy and the amount of calculation needed. In some cases, a good subset can achieve a better classification accuracy than the original, large feature set [12]. Our intent is not to

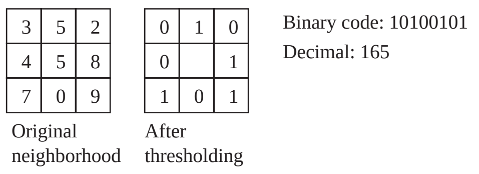
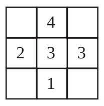


Fig. 1. Calculating the non-interpolated $LBP_{8,1}$. The binary code is read counter-clockwise starting at "8".



Index	Difference	Binary	Shifted sign
0	3 – 3 – 1 = –1	1111	0001
1	3 – 4 – 1 = –2	1110	0010
2	3 – 2 – 1 = 0	0000	0000
3	3 – 1 – 1 = 1	0001	0000
		$LBP_{4,1}$ (logical OR):	0011

Fig. 2. Optimized calculation of $LBP_{4,1}$.

develop new ways of reducing the number of features, but to use previously developed algorithms to accomplish this task. In a comparative study of feature selection algorithms, Jain and Zongker [13] found that the SFFS algorithm of Pudil et al. [10] was superior to the others tested. Consequently, it was selected for the experiments. The use of a feature selection algorithm was motivated by the fact that LBP distributions have been successfully pruned with the beam search [12].

The SFFS algorithm is an enhanced version of the forward search. On each iteration, all remaining features are added to the feature set in turn, and the one that resulted in the largest performance increase is selected. In a backward step, each of the already selected features is disabled in turn, and the performance of the remaining features is measured to see if removing the feature could increase performance.

As a performance criterion for the SFFS algorithm, the result of a 3-NN classification of a set of training samples was used. With the paper images, the classification was made with the leave-one-out principle. With Outex textures, a hold-out test was performed by dividing the training samples to two even sets. Hold-out was used to speed up the feature reduction with the large number of training samples. The final results were obtained by classifying an independent validation set. Two criteria were used in determining whether to stop the feature reduction. The algorithm was stopped if

(1) the training set was faultlessly classified
(2) the trend of the classification accuracy, calculated as a moving average of five successive results, rose less than 0.2 percentage units per iteration.

Criterion (2) is later referred to as a "saturation condition". Its intent is to find the point where adding new features no longer produces a performance increase large enough to compensate the increased computational burden. The value of the saturation limit is an application-specific factor that is used in balancing the speed versus accuracy trade-off.

As a statistical measure of image texture, the LBP distribution needs special attention when features are pruned. To ensure the sum of each feature vector remains unity, an extra bin was appended to the histograms. Another important point is that the time consumed in extracting the LBP distribution is not affected by the number of enabled features.

2.2.1. *SOM-based classification*

The computational burden of non-parametric classifiers like the k-NN classifier is linearly related to the number of training samples. This poses serious efficiency problems to many visual inspection applications as a large amount of training data is needed. In this work, self-organizing maps are used as fast classifiers.

SOM consists of a finite amount of nodes representing the original high-dimensional data in a low-dimensional node grid [11]. The code vectors on the map are adapted and labeled with training data. Each node is labeled according to the most common class among the training samples closest to its code vector. In classification, the SOM works as a simple vector quantizer: an unknown sample is classified according to the code vector closest to it.

Typically, the size of a SOM is selected so that the number of training samples is 5–20 times the number of nodes. As a result, the time consumed in classifying an unknown sample with the SOM is approximately 5–20 times shorter than that of the k-NN classifier. With a relatively small number of training samples, the speed difference may however not be that large as the size of the SOM cannot be made arbitrarily small without destroying its learning capabilities. The type of input data affects the size of the SOM. If the inspected data are complex and the features have no ability to discriminate them correctly, a larger SOM is needed. A number of methods of optimizing SOM-based classification exist [14]. Also, methods for making k-NN classification faster have been presented [15]. Since the speed finally comes down to the number of code vectors or training samples, the optimization methods are not considered here. Furthermore, the sizes of the self-organizing maps used are so small that the optimization methods may be useless anyway.

In training a SOM and in classification, the Euclidean distance has been conventionally used. In the experiments reported here, the simplified G statistic (log-likelihood measure) suggested for the LBP by Ojala et al. [9] was also employed:

$$L(S, M) = -\sum_{b=1}^{B} S_b \log M_b, \tag{1}$$

where S and M denote sample and model distributions, respectively. B is the number of bins in the distributions. With the k-NN classifier, this measure was used exclusively.

Fig. 3 shows a block diagram of the proposed system. Dotted lines represent the training phase, which is performed only once. The solid lines represent the on-line classification. The role of a user is to assign class labels to the training data by either inspecting each individual sample or by determining class boundaries on the SOM adapted to the training data. Despite training the SOM, the training images are used in tuning the feature set with the SFFS algorithm after they have been given class labels.

When pre-labeled data are not available, the visualization capabilities of the SOM can be utilized. The idea is to feed on-line images to feature extraction and classification and to visually determine class boundaries on the SOM [16]. There is no need to assign labels to

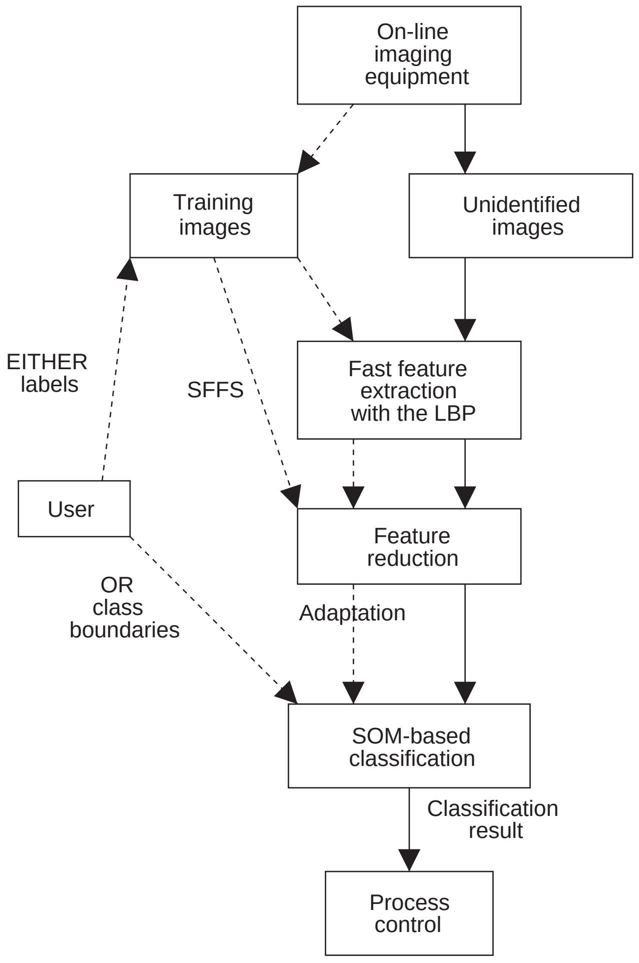


Fig. 3. Block diagram of the real-time inspection system.

individual samples, and the resulting map can be used in classification as such. After determining the class boundaries, the training data must be classified with the SOM with all features enabled to obtain class labels for the samples. Only after this can the feature reduction step be performed.

3. Experiments

To demonstrate the performance of the proposed system, two empirical experiments were arranged. First, a set of natural textures imaged in a highly controlled laboratory environment were utilized. The images were obtained from the Outex database which contains over 300 different textures, each captured under three different light sources, six spatial resolutions, and nine rotation angles [17]. This set of textures is well suited to showing the applicability of the methods to the discrimination of diverse textures in general. Furthermore, these textures provide a good basis for comparing the classification performance to other texture analysis methods. Second, a real-time paper inspection problem was used as an application study to show the performance of the system in a real-world problem.

3.1. Experiment #1—Outex textures

In the first experiment, a preselected set of 24 textures from the Outex database [17] was used. At the Outex site, this set has the test suite ID Outex_TC_00002. Although the images are not from an industrial inspection system, they contain textures that are likely to occur in one. The textures, shown in Fig. 4, contain 18 different canvases, four carpets and two ceramic tiles. The suite is produced by taking the maximum possible number of non-overlapping 32×32 pixel sub-images from each gray-scale image. This results in a total number of 8832 images, and 368 samples per texture. These are randomly divided into two even sets, one for training and the other for testing. The random division is performed 100 times. As a classification result, the mean and standard deviation of the classification accuracy over the 100 trials are reported.

Table 1 summarizes the results of the experiments. A "baseline" result for the test suite was obtained with a 3-NN classifier, using the $LBP_{8,1}$ (without interpolation) as a feature vector. The classification accuracy was on average 88.6%. This falls just a few percentage units short of the best result reported so far for this suite. With Gabor filtering, an accuracy of 92.6% has been obtained [17]. With the multi-resolution LBP proposed in [9] ($LBP^{u2}_{8,1} +^{u2}_{16,2} +^{u2}_{24,3}$), the accuracy was 91.1%. These numbers indicate that the accuracy of the fast $LBP_{8,1}$ is fairly close to the current state-of the art. The computational burden is, however, significantly smaller [18].

Classification with a k-NN classifier with 4416 training samples is quite time consuming. A seven-fold increase in speed was achieved without a considerable loss of accuracy by a SOM-based classifier. A 25×25 SOM was trained and labeled with the training samples, using the log-likelihood dissimilarity measure. The rest

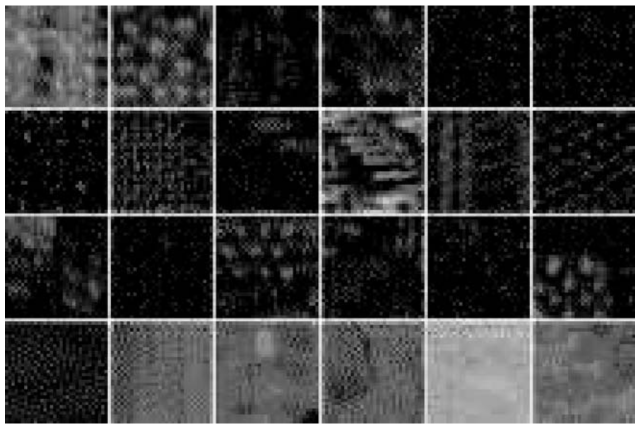


Fig. 4. A 32×32 sample of each Outex texture used in experiment #1.

Table 1
Outex classification results

Method	Features	Speed (ms/sample)	Accuracy (%)	Deviation (%-units)
SOM 25×25	256	52	88.4	0.36
SOM 25×25	$58+1$	11	87.0	0.58
3-NN	256	378	88.6	0.29
3-NN	$58+1$	89	88.7	0.36

of the samples were then classified against the adapted code book of the SOM. The accuracy fell only 0.2 percentage units short of that of the 3-NN classifier. Using the log-likelihood measure instead of the Euclidean distance clearly helps, as the result with the latter was only 81.7%. Consequently, the log-likelihood measure was used exclusively in the experiments.

Classifying Outex textures seemed to be a challenging problem in which a large portion of the LBP codes were needed for good accuracy. When running the SFFS algorithm, the saturation condition was fulfilled after selecting 15 features. In classifying the test set with these features, an accuracy of only 76.7% was achieved. Interestingly, all the codes selected so far were "uniform", as defined by Ojala et al. [9]. Therefore, classification experiments were repeated with all the uniform codes enabled. With $LBP_{8,1}$, there is a total number of 58 uniform codes. With the additional entry added to make the sum of the feature vectors unity, the length of the feature vectors became 59. Considering only these features results in an almost five-fold increase in speed. At the same time, the classification accuracy remains almost the same, as shown in Table 1. The robustness of the shorter feature vector with respect to variations in training and testing data seems to be somewhat worse. Anyhow, the standard deviation of the classification score over the 100 trials is relatively small in all cases.

The results show that an accuracy close to that of the state-of-the-art can be achieved with the proposed system in a general texture discrimination problem. The computational burden is very small: the analysis of an unknown image could be performed in 11 ms.

3.2. Experiment #2—Real-time paper inspection

Paper characterization is an extreme example of a real-time classification application. The paper web moves about 30 m/s, posing tough requirements for an analysis system. The image acquisition itself is a difficult problem which must be considered carefully. Requirements for image analysis are simple: faster is better. With a spatial image width of about 0.6 m, a frame rate of 50 fps is needed to ensure no material is missed. Therefore, taking an image, transferring it to the analysis system, extracting features, and classifying may take only about 20 ms. With paper it probably makes no harm to miss a few centimeters between successive frames, because the overall quality does not change rapidly. Nevertheless, a continuous estimate of paper quality is of high value to the control system.

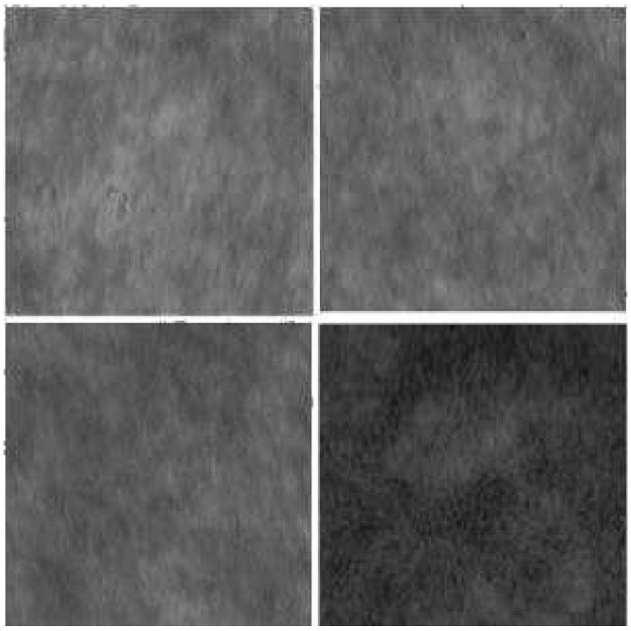


Fig. 5. A sample of each of the four paper quality classes used in experiment #2.

The problem used in judging the performance of the proposed system was to classify paper samples into four quality classes. The testing data contained back-lit paper images in which brighter areas indicate thinner paper. A sample of each quality class is shown in Fig. 5. Three of the classes are practically indistinguishable by the eye, but the fourth one is clearly different. The data to be analyzed were 8-bit gray-scale images with 756×566 pixels.

A hold-out test was used in determining the classification accuracy. The original set of 1004 images was divided into two distinct sets: one for training and the other for testing. A 3-NN classifier was again used in obtaining a "baseline" result. With all features enabled, the classification accuracy was 99.8%, which may make one to suspect that the problem is over-simplified. However, in a recent comparative study, a large number of different texture features were evaluated with the same data. With methods previously used in paper inspection, the best accuracy, 75.7%, was obtained with FFT-based features [19].

As a fast classifier, a SOM with 10×8 nodes was used. The map is small enough to achieve real-time performance while providing good accuracy. Compared to the 3-NN classifier with 502 training samples, the SOM is about six times faster. The classification accuracy is however somewhat weaker.

Table 2
Paper classification results

Method	Features	Speed (ms/sample)	Accuracy (%)
SOM 10×8	256	1.7	98.4
SOM 10×8	$3+1$	0.17	99.2
3-NN	256	11	99.8
3-NN	$3+1$	0.75	99.8

With the paper images, the SFFS algorithm was highly successful. The training set was faultlessly classified with only three enabled LBP codes. Recalling the "distribution normalizer" bin, the total length of the pruned LBP feature vector became four. Hence, the speed improvement compared to the full LBP distribution was 64-fold, without loss of accuracy. The accuracy of a 3-NN classification of the testing data with these three features was 99.8%. An interesting result was that with the SOM, the pruned feature vector even increased classification accuracy. Classification results and the times elapsed in classification are shown in Table 2.

4. Discussion

When designing a real-time visual inspection system, one often needs to balance between speed and accuracy requirements. This is also the case with the proposed system. Shorter feature vectors typically mean somewhat worse classification accuracy, and one needs to find the optimal trade-off. Similarly, the SOM-based classifier is much faster than k-NN, but slightly more inaccurate. Furthermore, the discrimation performance of the $LBP_{8,1}$ is certainly not good enough for all applications. If the processing time requirements can be relaxed, the multi-resolution LBP might be a good candidate for a powerful feature [9]. Even though it results in slower feature extraction and longer feature vectors, these problems can be reduced by the very same methods that were applied here to the $LBP_{8,1}$.

With the Outex textures, reducing features to a small subset with the SFFS algorithm results in a performance drop too large to be acceptable. The "uniform" patterns can however be used in achieving an almost five-fold speed improvement without significantly affecting classification accuracy. With the paper data, a small subset of features seems to carry all the necessary information. With the SOM, the pruned feature vectors perform even better than the original one. The difference in accuracy between the k-NN classifier and the SOM is in most cases so small that it can be justified by the large difference in classification speed. In industrial inspection applications, however, the most important feature of the SOM is its ability to visualize the data and make it easy for the user to train the system.

In applications like paper inspection, feature extraction must be performed very quickly. For this task, the optimized LBP operator is a good candidate, particularly because the LBP itself has been shown to perform well in many applications and comparative studies [4,5,9,18]. An LBP operator implemented with the optimization technique presented in this paper can be used in processing large images at video rates.

The time consumed in determining the quality class of a previously unseen sample of paper was successfully reduced to 20 ms. If image acquisition and transfer can be done in parallel to the analysis task, this means that practically no paper is missed between successive frames. This result was achieved by utilizing all the components of the system, depicted in Fig. 3.

The relative speeds of the feature extraction and classification with respect to each other heavily depend on the number of texture classes, features and the size of the input images. In experiment #1, a large SOM must be used due to the large number of different classes. The input images are small, but a relatively large number of features must be used. Consequently, feature extraction is much faster than classification. In experiment #2, the situation is just the opposite—the time used in classifying the extracted and reduced features is negligible. Even though the SOM-based classifier is clearly faster than k-NN, the difference is not very important because feature extraction takes much more time. The performance of the SOM becomes more prominent with a larger number of training samples.

5. Conclusions

This paper presented a framework for real-time visual inspection with texture. The proposed system is based on a very fast implementation of the LBP texture operator, the SFFS feature selection algorithm, and a SOM-based classifier with which a log-likelihood dissimilarity measure is used. The performance of the system was evaluated with a set of natural textures imaged in a laboratory environment, and in a real-time paper inspection application. Fast feature extraction, feature reduction, and SOM-based classification were combined to achieve real-time performance in a very demanding real-time application. The obtained throughput was 50 analyzed images per second, with an image size of 756×566 pixels.

Acknowledgements

The financial support provided by the Academy of Finland, the national Graduate School in Electronics, Telecommunication, and Automation, and Nokia Foundation is gratefully acknowledged.

Appendix. Optimized C code for calculating the LBP

```
#include <string.h>
#include <stdlib.h>
#define compab_mask_inc(ptr,shift) \
{ value |= ((unsigned int)(cntr - *ptr) & 0x80000000) \
   >> (31-shift); ptr++; }
/**
* Calculate a LBP8,1 feature vector for an image array.
* This function does not use interpolation. The input
* image is presented as a linear array, in raster-scan
* order. As a result, a newly allocated array of 256
* integers is returned.
** /
int* LBP8(const int* data, int rows, int columns)
{
   const int
      *p0 = data,
      *p1 = p0 + 1,
      *p2 = p1 + 1,
      *p3 = p2 + columns,
      *p4 = p3 + columns,
      *p5 = p4 - 1,
      *p6 = p5 - 1,
      *p7 = p6 - columns,
      *center = p7 + 1;
   int r,c,cntr;
   unsigned int value;
   int* result = (int*)malloc(256*sizeof(int));
   memset(result, 0, 256*sizeof(int));
   for (r=0;r<rows-2;r++)
      {
         for (c=0;c<columns-2;c++)
            {
               value = 0;
               cntr = *center - 1;
               compab_mask_inc(p0,0);
               compab_mask_inc(p1,1);
               compab_mask_inc(p2,2);
               compab_mask_inc(p3,3);
               compab_mask_inc(p4,4);
               compab_mask_inc(p5,5);
               compab_mask_inc(p6,6);
               compab_mask_inc(p7,7);
               center++;
               result[value]++;
            }
         p0 += 2;
         p1 += 2;
         p2 += 2;
         p3 += 2;
         p4 += 2;
         p5 += 2;
         p6 += 2;
         p7 += 2;
         center += 2;
      }
   return result;
}
```

References

[1] Mäenpää T, Viertola J, Pietikäinen M. Optimizing color and texture features for real-time visual inspection. Pattern Analysis and Applications 6, in press.

[2] Cohen F, Fan Z, Attali S. Automated inspection of textile fabrics using textural models. IEEE Transactions on Pattern Analysis and Machine Intelligence 1991;13(8):803–8.

[3] Kumar A, Pang G. Defect detection in textured materials using Gabor filters. IEEE Transactions on Industry Applications 2002;38(2):425–40.

[4] Xu Z, Pietikäinen M, Ojala T. Defect classification by texture in steel surface inspection. In: International Conference on Quality Control by Artificial Vision, Le Creusot, Burgundy, France, 1997. p. 179–84.

[5] Iivarinen J. Unsupervised segmentation of surface defects with simple texture measures. In: Pietikäinen M. editor. Machine perception and artificial intelligence, Vol. 40. Singapore: World Scientific; 2000. p. 231–8.

[6] Manjunath B, Ma W. Texture features for browsing and retrieval of image data. IEEE Transactions on Pattern Analysis and Machine Intelligence 1996;18(8):837–42.

[7] Grigorescu S, Petkov N, Kruizinga P. Comparison of texture features based on Gabor filters. IEEE Transactions on Image Processing 2002;11(10):1160–7.

[8] Bodnarova A, Bennamoun M, Latham S. Textile flaw detection using optimal Gabor filters. In: 15th International Conference on Pattern Recognition, Vol. 4, Barcelona, Spain, 2000. pp. 799–802.

[9] Ojala T, Pietikäinen M, Mäenpää T. Multiresolution gray scale and rotation invariant texture analysis with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence 2002;24(7):971–87.

[10] Pudil P, Novovičová J, Kittler J. Floating search methods in feature selection. Pattern Recognition Letters 1994;15:1119–25.

[11] Kohonen T. Self-organizing maps. Berlin, Germany: Springer; 1997.

[12] Mäenpää T, Ojala T, Pietikäinen M, Soriano M. Robust texture classification by subsets of local binary patterns. In: 15th International Conference on Pattern Recognition, Vol. 3, Barcelona, Spain, 2000. p. 947–50.

[13] Jain A, Zongker D. Feature selection: evaluation, application and small sample performance. IEEE Transactions on Pattern Analysis and Machine Intelligence 1997;19(2):153–8.

[14] Niskanen M, Kauppinen H, Silvén O. Real-time aspects of SOM-based visual surface inspection. In: Proceedings SPIE Machine Vision Applications in Industrial Inspection X, Vol. 4664, San Jose, CA, USA, 2002. p. 123–34.

[15] Dasarathy B. editor. Nearest neighbor (NN) norms: NN pattern classification techniques. Los Alamitos, CA, IEEE Computer Society Press; 1991.

[16] Silven O, Niskanen M, Kauppinen H. Wood inspection with non-supervised clustering. Machine Vision and Applications 2003;3(5–6):275–85.

[17] Ojala T, Mäenpää T, Pietikäinen M, Viertola J, Kyllönen J, Huovinen S. Outex—new framework for empirical evaluation of texture analysis algorithms. In: 16th International Conference on Pattern Recognition, Vol. 1, Québec, Canada, 2002. p. 701–6. http://www.outex.oulu.fi/.

[18] Ojala T, Mäenpää T, Viertola J, Kyllönen J, Pietikäinen M. Empirical evaluation of MPEG-7 texture descriptors with a large-scale experiment. In: The Second International Workshop on Texture Analysis and Synthesis, Copenhagen, Denmark, 2002. p. 99–102.

[19] Turtinen M, Pietikäinen M, Silven O, Mäenpää T, Niskanen M. Texture-based paper characterization using non-supervised clustering. In: 6th International Conference on Quality Control by Artificial Vision, (SPIE Vol. 5132) Gatlinburg, TN, USA, 2003, p. 178–188.