

# Fast Code Detection Using High Speed Time Delay Neural Networks

Hazem M. El-Bakry<sup>1</sup> and Nikos Mastorakis<sup>2</sup>

<sup>1</sup>Faculty of Computer Science & Information Systems,  
Mansoura University, Egypt  
helbakry20@yahoo.com

<sup>2</sup>Department of Computer Science,  
Military Institutions of University Education (MIUE)-Hellenic Naval Academy,  
Greece

**Abstract.** This paper presents a new approach to speed up the operation of time delay neural networks for fast code detection. The entire data are collected together in a long vector and then tested as a one input pattern. The proposed fast time delay neural networks (FTDNNs) use cross correlation in the frequency domain between the tested data and the input weights of neural networks. It is proved mathematically and practically that the number of computation steps required for the presented time delay neural networks is less than that needed by conventional time delay neural networks (CTDNNs). Simulation results using MATLAB confirm the theoretical computations.

## 1 Introduction

Recently, time delay neural networks have shown very good results in different areas such as automatic control, speech recognition, blind equalization of time-varying channel and other communication applications. The main objective of this research is to reduce the response time of time delay neural networks. The purpose is to perform the testing process in the frequency domain instead of the time domain. Our approach was successfully applied for sub-image detection using fast neural networks (FNNs) as proposed in [1,2,3]. Furthermore, it was used for fast face detection [7,9], and fast iris detection [8]. Another idea to further increase the speed of FNNs through image decomposition was suggested in [7].

FNNs for detecting a certain code in one dimensional serial stream of sequential data were described in [4,5]. Compared with conventional neural networks, FNNs based on cross correlation between the tested data and the input weights of neural networks in the frequency domain showed a significant reduction in the number of computation steps required for certain data detection [1,2,3,4,5,7,8,9,11,12]. Here, we make use of our theory on FNNs implemented in the frequency domain to increase the speed of time delay neural networks. The idea of moving the testing process from the time domain to the frequency domain is applied to time delay neural networks. Theoretical and practical results show that the proposed FTDNNs are faster than CTDNNs. In section 2, our theory on FNNs for detecting certain data in one dimensional matrix is described. Experimental results for FTDNNs are presented in section 3.

## 2 Fast Code Detection Using Cross Correlation in the Frequency Domain

Finding a certain code/data in the input one dimensional matrix is a searching problem. Each position in the input matrix is tested for the presence or absence of the required code/data. At each position in the input matrix, each sub-matrix is multiplied by a window of weights, which has the same size as the sub-matrix. The outputs of neurons in the hidden layer are multiplied by the weights of the output layer. When the final output is high, this means that the sub-matrix under test contains the required code/data and vice versa. Thus, we may conclude that this searching problem is a cross correlation between the matrix under test and the weights of the hidden neurons.

The convolution theorem in mathematical analysis says that a convolution of  $f$  with  $h$  is identical to the result of the following steps: let  $F$  and  $H$  be the results of the Fourier Transformation of  $f$  and  $h$  in the frequency domain. Multiply  $F$  and  $H^*$  (conjugate of  $H$ ) in the frequency domain point by point and then transform this product into the spatial domain via the inverse Fourier Transform. As a result, these cross correlations can be represented by a product in the frequency domain. Thus, by using cross correlation in the frequency domain, speed up in an order of magnitude can be achieved during the detection process [1,2,3,4,5,7,8,9,14]. In the detection phase, a sub matrix  $I$  of size  $1 \times n$  (sliding window) is extracted from the tested matrix, which has a size  $1 \times N$ , and fed to the neural network. Let  $W_i$  be the matrix of weights between the input sub-matrix and the hidden layer. This vector has a size of  $1 \times n$  and can be represented as  $1 \times n$  matrix. The output of hidden neurons  $h(i)$  can be calculated as follows:

$$h_i = g \left( \sum_{k=1}^n W_i(k) I(k) + b_i \right) \tag{1}$$

where  $g$  is the activation function and  $b(i)$  is the bias of each hidden neuron (i). Equation 1 represents the output of each hidden neuron for a particular sub-matrix  $I$ . It can be obtained to the whole input matrix  $Z$  as follows:

$$h_i(u) = g \left( \sum_{k=-n/2}^{n/2} W_i(k) Z(u+k) + b_i \right) \tag{2}$$

Eq.2 represents a cross correlation operation. Given any two functions  $f$  and  $d$ , their cross correlation can be obtained by:

$$d(x) \otimes f(x) = \left( \sum_{n=-\infty}^{\infty} f(x+n) d(n) \right) \tag{3}$$

Therefore, Eq. 2 may be written as follows [1]:

$$h_i = g(W_i \otimes Z + b_i) \tag{4}$$

where  $h_i$  is the output of the hidden neuron (i) and  $h_i(u)$  is the activity of the hidden unit (i) when the sliding window is located at position (u) and  $(u) \in [N-n+1]$ .

Now, the above cross correlation can be expressed in terms of one dimensional Fast Fourier Transform as follows [1]:

$$W_i \otimes Z = F^{-1} \left( F(Z) \bullet F^*(W_i) \right) \quad (5)$$

Hence, by evaluating this cross correlation, a speed up ratio can be obtained comparable to conventional neural networks. Also, the final output of the neural network can be evaluated as follows:

$$O(u) = g \left( \sum_{i=1}^q W_o(i) h_i(u) + b_o \right) \quad (6)$$

where  $q$  is the number of neurons in the hidden layer.  $O(u)$  is the output of the neural network when the sliding window located at the position  $(u)$  in the input matrix  $Z$ .  $W_o$  is the weight matrix between hidden and output layer.

The complexity of cross correlation in the frequency domain can be analyzed as follows:

1- For a tested matrix of  $1 \times N$  elements, the 1D-FFT requires a number equal to  $N \log_2 N$  of complex computation steps [13]. Also, the same number of complex computation steps is required for computing the 1D-FFT of the weight matrix at each neuron in the hidden layer.

2- At each neuron in the hidden layer, the inverse 1D-FFT is computed. Therefore,  $q$  backward and  $(1+q)$  forward transforms have to be computed. Therefore, for a given matrix under test, the total number of operations required to compute the 1D-FFT is  $(2q+1)N \log_2 N$ .

3- The number of computation steps required by FNNs is complex and must be converted into a real version. It is known that, the one dimensional Fast Fourier Transform requires  $(N/2) \log_2 N$  complex multiplications and  $N \log_2 N$  complex additions [13]. Every complex multiplication is realized by six real floating point operations and every complex addition is implemented by two real floating point operations. Therefore, the total number of computation steps required to obtain the 1D-FFT of a  $1 \times N$  matrix is:

$$\rho = 6((N/2) \log_2 N) + 2(N \log_2 N) \quad (7)$$

which may be simplified to:

$$\rho = 5N \log_2 N \quad (8)$$

4- Both the input and the weight matrices should be dot multiplied in the frequency domain. Thus, a number of complex computation steps equal to  $qN$  should be considered. This means  $6qN$  real operations will be added to the number of computation steps required by FNNs.

5- In order to perform cross correlation in the frequency domain, the weight matrix must be extended to have the same size as the input matrix. So, a number of zeros =  $(N-n)$  must be added to the weight matrix. This requires a total real number of computation steps =  $q(N-n)$  for all neurons. Moreover, after computing the FFT for the weight matrix, the conjugate of this matrix must be obtained. As a result, a real number of computation steps =  $qN$  should be added in order to obtain the conjugate of

the weight matrix for all neurons. Also, a number of real computation steps equal to  $N$  is required to create butterflies complex numbers ( $e^{-jk(2\pi n/N)}$ ), where  $0 < K < L$ . These  $(N/2)$  complex numbers are multiplied by the elements of the input matrix or by previous complex numbers during the computation of FFT. To create a complex number requires two real floating point operations. Thus, the total number of computation steps required for FNNs becomes:

$$\sigma = (2q+1)(5N \log_2 N) + 6qN + q(N-n) + qN + N \quad (9)$$

which can be reformulated as:

$$\sigma = (2q+1)(5N \log_2 N) + q(8N-n) + N \quad (10)$$

6- Using sliding window of size  $1 \times n$  for the same matrix of  $1 \times N$  pixels,  $q(2n-1)(N-n+1)$  computation steps are required when using CTDNNs for certain code detection or processing  $(n)$  input data. The theoretical speed up factor  $\eta$  can be evaluated as follows:

$$\eta = \frac{q(2n-1)(N-n+1)}{(2q+1)(5N \log_2 N) + q(8N-n) + N} \quad (11)$$

### 3 Simulation Results

Time delay neural networks accept serial input data with fixed size  $(n)$ . Therefore, the number of input neurons equals to  $(n)$ . Instead of treating  $(n)$  inputs, our new approach is to collect all the input data together in a long vector (for example  $100 \times n$ ). Then the input data is tested by time delay neural networks as a single pattern with length  $L$  ( $L=100 \times n$ ). Such a test is performed in the frequency domain as described in section II. Complex-valued neural networks have many applications in fields dealing with complex numbers such as telecommunications, speech recognition and image processing with the Fourier Transform [6,10]. Complex-valued neural networks mean that the inputs, weights, thresholds and the activation function have complex values. In this section, formulas for the speed up ratio with different types of inputs will be presented. The special case of only real input values (i.e. imaginary part=0) will be considered. Also, the speed up ratio in the case of a one and two dimensional input matrix will be concluded. The operation of FNNs depends on computing the Fast Fourier Transform for both the input and weight matrices and obtaining the resulting two matrices. After performing dot multiplication for the resulting two matrices in the frequency domain, the Inverse Fast Fourier Transform is calculated for the final matrix. Here, there is an excellent advantage with FNNs that should be mentioned. The Fast Fourier Transform is already dealing with complex numbers, so there is no change in the number of computation steps required for FNNs. Therefore, the speed up ratio in the case of complex-valued time delay neural networks can be evaluated as follows:

#### 1) In case of real inputs

A) For a one dimensional input matrix

Multiplication of  $(n)$  complex-valued weights by  $(n)$  real inputs requires  $(2n)$  real operations. This produces  $(n)$  real numbers and  $(n)$  imaginary numbers. The addition

of these numbers requires  $(2n-2)$  real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(2n-1)(N-n+1) \quad (12)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(2n-1)(N-n+1)}{(2q+1)(5N \log_2 N) + q(8N-n) + N} \quad (13)$$

The theoretical speed up ratio for searching short successive ( $n$ ) data in a long input vector ( $L$ ) using complex-valued time delay neural networks is shown in Tables 1, 2, and 3. Also, the practical speed up ratio for manipulating matrices of different sizes ( $L$ ) and different sized weight matrices ( $n$ ) using a 2.7 GHz processor and MATLAB is shown in Table 4.

**Table 1.** The theoretical speed up ratio for time delay neural networks (1D-real values input matrix,  $n=400$ )

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	4.6027e+008	4.2926e+007	10.7226
40000	1.8985e+009	1.9614e+008	9.6793
90000	4.2955e+009	4.7344e+008	9.0729
160000	7.6513e+009	8.8219e+008	8.6731
250000	1.1966e+010	1.4275e+009	8.3823

**Table 2.** The theoretical speed up ratio for time delay neural networks (1D-real values input matrix,  $n=625$ )

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	7.0263e+008	4.2919e+007	16.3713
40000	2.9508e+009	1.9613e+008	15.0452
90000	6.6978e+009	4.7343e+008	14.1474
160000	1.1944e+010	8.8218e+008	13.5388
250000	1.8688e+010	1.4275e+009	13.0915

**Table 3.** The theoretical speed up ratio for time delay neural networks (1D-real values input matrix,  $n=900$ )

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
10000	9.823 e+008	4.2911e+007	22.8933
40000	4.2206e+009	1.9612e+008	21.5200
90000	9.6176e+009	4.7343e+008	20.3149
160000	1.7173e+010	8.8217e+008	19.4671
250000	2.6888e+010	1.4275e+009	18.8356

**Table 4.** Practical speed up ratio for time delay neural networks (1D-real values input matrix)

Length of input matrix	Speed up ratio (n=400)	Speed up ratio (n=625)	Speed up ratio (n=900)
10000	17.88	25.94	35.21
40000	17.19	25.11	34.43
90000	16.65	24.56	33.59
160000	16.14	24.14	33.05
250000	15.89	23.76	32.60

B) For a two dimensional input matrix

Multiplication of  $(n^2)$  complex-valued weights by  $(n^2)$  real inputs requires  $(2n^2)$  real operations. This produces  $(n^2)$  real numbers and  $(n^2)$  imaginary numbers. The addition of these numbers requires  $(2n^2-2)$  real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(2n^2 - 1)(N - n + 1)^2 \tag{14}$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(2n^2 - 1)(N - n + 1)^2}{(2q + 1)(5N^2 \log_2 N^2) + q(8N^2 - n^2) + N} \tag{15}$$

The theoretical speed up ratio for detecting  $(nxn)$  real valued submatrix in a large real valued matrix  $(NxN)$  using complex-valued time delay neural networks is shown in Tables 5, 6, 7. Also, the practical speed up ratio for manipulating matrices of different sizes  $(NxN)$  and different sized weight matrices  $(n)$  using a 2.7 GHz processor and MATLAB is shown in Table 8.

**Table 5.** The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=20)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	3.1453e+008	4.2916e+007	7.3291
200x200	1.5706e+009	1.9610e+008	8.0091
300x300	3.7854e+009	4.7335e+008	7.9970
400x400	6.9590e+009	8.8203e+008	7.8898
500x500	1.1091e+010	1.4273e+009	7.7711

**Table 6.** The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=25)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	4.3285e+008	4.2909e+007	10.0877
200x200	2.3213e+009	1.9609e+008	11.8380
300x300	5.7086e+009	4.7334e+008	12.0602
400x400	1.0595e+010	8.8202e+008	12.0119
500x500	1.6980e+010	1.4273e+009	11.8966

**Table 7.** The theoretical speed up ratio for time delay neural networks (2D-real values input matrix, n=30)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	5.4413e+008	4.2901e+007	12.6834
200x200	3.1563e+009	1.9608e+008	16.0966
300x300	7.9272e+009	4.7334e+008	16.7476
400x400	1.4857e+010	8.8201e+008	16.8444
500x500	2.3946e+010	1.4273e+009	16.7773

**Table 8.** Practical speed up ratio for time delay neural networks (2D-real values input matrix)

Size of input matrix	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	17.19	22.32	31.74
200x200	17.61	22.89	32.55
300x300	16.54	23.66	33.71
400x400	15.98	22.95	34.53
500x500	15.62	22.49	33.32

### 2) In case of complex inputs

A) For a one dimensional input matrix

Multiplication of (n) complex-valued weights by (n) complex inputs requires (6n) real operations. This produces (n) real numbers and (n) imaginary numbers. The addition of these numbers requires (2n-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(4n-1)(N-n+1) \quad (16)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(4n-1)(N-n+1)}{(2q+1)(5N \log_2 N) + q(8N-n) + N} \quad (17)$$

**Table 9.** The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=400)

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	9.2111e+008	4.2926e+007	21.4586
200x200	3.7993e+009	1.9614e+008	19.3706
300x300	8.5963e+009	4.7344e+008	18.1571
400x400	1.5312e+010	8.8219e+008	17.3570
500x500	2.3947e+010	1.4275e+009	16.7750

The theoretical speed up ratio for searching short complex successive (n) data in a long complex-valued input vector (L) using complex-valued time delay neural

networks is shown in Tables 9, 10, and 11. Also, the practical speed up ratio for manipulating matrices of different sizes (L) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 12.

**Table 10.** The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=625)

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.4058e+009	4.2919e+007	32.7558
200x200	5.9040e+009	1.9613e+008	30.1025
300x300	1.3401e+010	4.7343e+008	28.3061
400x400	2.3897e+010	8.8218e+008	27.0883
500x500	3.7391e+010	1.4275e+009	26.1934

**Table 11.** The theoretical speed up ratio for time delay neural networks (1D-complex values input matrix, n=900)

Length of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.9653e+009	4.2911e+007	45.7993
200x200	8.4435e+009	1.9612e+008	43.0519
300x300	1.9240e+010	4.7343e+008	40.6410
400x400	3.4356e+010	8.8217e+008	38.9450
500x500	5.3791e+010	1.4275e+009	37.6817

**Table 12.** Practical speed up ratio for time delay neural networks (1D-complex values input matrix)

Length of input matrix	Speed up ratio (n=400)	Speed up ratio (n=625)	Speed up ratio (n=900)
10000	37.90	53.58	70.71
40000	36.82	52.89	69.43
90000	36.34	52.47	68.69
160000	35.94	51.88	68.05
250000	35.69	51.36	67.56

B) For a two dimensional input matrix

Multiplication of (n<sup>2</sup>) complex-valued weights by (n<sup>2</sup>) real inputs requires (6n<sup>2</sup>) real operations. This produces (n<sup>2</sup>) real numbers and (n<sup>2</sup>) imaginary numbers. The addition of these numbers requires (2n<sup>2</sup>-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta = 2q(4n^2 - 1)(N - n + 1)^2 \tag{18}$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(4n^2 - 1)(N - n + 1)^2}{(2q + 1)(5N^2 \log_2 N^2) + q(8N^2 - n^2) + N} \tag{19}$$



The theoretical speed up ratio for detecting (nxn) complex-valued submatrix in a large complex-valued matrix (NxN) using complex-valued neural networks is shown in Tables 13, 14, and 15. Also, the practical speed up ratio for manipulating matrices of different sizes (NxN) and different sized weight matrices (n) using a 2.7 GHz processor and MATLAB is shown in Table 16.

**Table 13.** The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=20)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	6.2946e+008	4.2916e+007	14.6674
200x200	3.1431e+009	1.9610e+008	16.0281
300x300	7.5755e+009	4.7335e+008	16.0040
400x400	1.3927e+010	8.8203e+008	15.7894
500x500	2.2197e+010	1.4273e+009	15.5519

**Table 14.** The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=25)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	8.6605e+008	4.2909e+007	20.1836
200x200	4.6445e+009	1.9609e+008	23.6856
300x300	1.1422e+010	4.7334e+008	24.1301
400x400	2.1198e+010	8.8202e+008	24.0333
500x500	3.3973e+010	1.4273e+009	23.8028

**Table 15.** The theoretical speed up ratio for time delay neural networks (2D-complex values input matrix, n=30)

Size of input matrix	Number of computation steps required for classic complex-valued neural networks	Number of computation steps required for fast complex-valued neural networks	Speed up ratio
100x100	1.0886e+009	4.2901e+007	25.3738
200x200	6.3143e+009	1.9608e+008	32.2021
300x300	1.5859e+010	4.7334e+008	33.5045
400x400	2.9722e+010	8.8201e+008	33.6981
500x500	4.7904e+010	1.4273e+009	33.5640

**Table 16.** Practical speed up ratio for time delay neural networks (2D-complex values input matrix)

Size of input matrix	Speed up ratio (n=20)	Speed up ratio (n=25)	Speed up ratio (n=30)
100x100	38.33	46.99	62.88
200x200	39.17	47.79	63.77
300x300	38.44	48.86	64.83
400x400	37.92	47.23	65.99
500x500	37.32	46.89	64.89

## 4 Conclusion

New FTDNNs have been presented. Theoretical computations have shown that FTDNNs require fewer computation steps than conventional ones. This has been achieved by applying cross correlation in the frequency domain between the input data and the input weights of time delay neural networks. Simulation results have confirmed this proof by using MATLAB. This algorithm can be successfully applied to any application that uses time delay neural networks.

## References

- [1] El-Bakry, H. M., Zhao, Q.: A Modified Cross Correlation in the Frequency Domain for Fast Pattern Detection Using Neural Networks. *International Journal of Signal Processing* **1** (2004) 188-194
- [2] El-Bakry, H. M., Zhao, Q.: Fast Object/Face Detection Using Neural Networks and Fast Fourier Transform. *International Journal of Signal Processing* **1** (2004) 182-187
- [3] El-Bakry, H. M., Zhao, Q.: Fast Pattern Detection Using Normalized Neural Networks and Cross Correlation in the Frequency Domain. accepted and under publication in the *EURASIP Journal on Applied Signal Processing*
- [4] El-Bakry, H. M., Zhao, Q.: A Fast Neural Algorithm for Serial Code Detection in a Stream of Sequential Data. *International Journal of Information Technology* **2** (2005) 71-90
- [5] El-Bakry, H.M., Stoyan, H.: FNNs for Code Detection in Sequential Data Using Neural Networks for Communication Applications. *Proc. of the First International Conference on Cybernetics and Information Technologies, Systems and Applications: CITSA 2004 Orlando, Florida, USA, Vol. IV, 150-153.*
- [6] Hirose, A.: *Complex-Valued Neural Networks Theories and Applications. Series on innovative Intellegence* **5** (2003)
- [7] El-Bakry, H.M.: Face detection using fast neural networks and image decomposition. *Neurocomputing Journal*, **48** (2002) 1039-1046
- [8] El-Bakry, H.M.: Human Iris Detection Using Fast Cooperative Modular Neural Nets and Image Decomposition. *Machine Graphics & Vision Journal (MG&V)* **11** (2002) 498-512
- [9] El-Bakry, H.M.: Automatic Human Face Recognition Using Modular Neural Networks," *Machine Graphics & Vision Journal (MG&V)* **10** (2001) 47-73
- [10] Jankowski, S., Lozowski, A., Zurada, M.: Complex-valued Multistate Neural Associative Memory. *IEEE Trans. on Neural Networks* **7** (1996) 1491-1496
- [11] El-Bakry, H.M., Zhao, Q.: Fast Pattern Detection Using Neural Networks Realized in Frequency Domain. *Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, (2005) 89-92*
- [12] El-Bakry, H.M., Zhao, Q.: Sub-Image Detection Using Fast Neural Processors and Image Decomposition. *Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, (2005) 85-88*
- [13] Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. *Math. Comput* **19** (1965) 297-301
- [14] Klette, R., Zamperon: *Handbook of image processing operators. John Wiley & Sons Ltd (1996)*