ANALOGIC COMPUTERS LTD.

Aladdin V1.3

CNN SOFTWARE LIBRARY FOR ACE4K CHIP
(TEMPLATES AND ALGORITHMS)

VERSION 1.0

Budapest

2000

TABLE OF CONTENTS

1. Templates/Instructions

1.1. BASIC IMAGE PROCESSING

GradientIntensityEstimation: ***Estimation of the gradient intensity in a local neighborhood***

Old names: AVERGRAD

A =

0	0	0
0	0	0
0	0	0

B =

b	***b***	***b***
b	0	***b***
b	***b***	***b***

z = 0

where $\boldsymbol{b} = |v_{u_{ij}} - v_{u_{kl}}| / 8$.

I. Global Task

Given: static grayscale image **P**

Input: **U(t) = P**

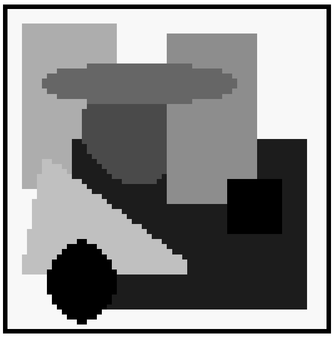
Initial State: **X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)

Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

Output: **Y(t)⇒Y(∞)** = Grayscale image representing the estimated average gradient intensity in a local neighborhood in **P**.

II. Examples

Example 1: image name: avergra2.bmp, image size: 64x64; template name: avergrad.tem.



input

output

III. ACE4K implementation

Implementation method: optimization simplification .

GradIntEstimation_ACE4K: (Full-range model, ACE4K)

Horidiffc.tem

A =

0	0	0
0	0	0
0	0	0

B =

0	0	0
0	1	-3
0	0	0

z_1 = 0 z_2 = 2.45

Verdiffc.tem

A =

0	0	0
0	0	0
0	0	0

B =

0	-2.5	0
0	0.8	0
0	0	0

z_1 = 0 z_2 = 1.45

NegLAMLLM.tem

A =

0	0	0
0	3	0
0	0	0

B =

0	0	0
0	3	0
0	0	0

z_1 = 0 z_2 = 1.1

AbsVal.tem

A =

0	0	0
0	-1	0
0	0	0

B =

0	0	0
0	-.9	0
0	0	0

z_1 = 1.7 z_2 = 0

AverHor.tem

A =

0	0	0
0	0	0
0	0	0

B =

0	0	0
0	.5	0
0	0	0

z_1 = .8 z_2 = 0

AverVer.tem

A =

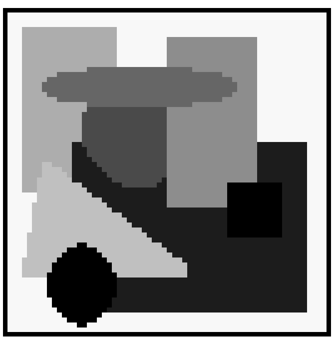
0	0	0
0	1	0
0	0	0

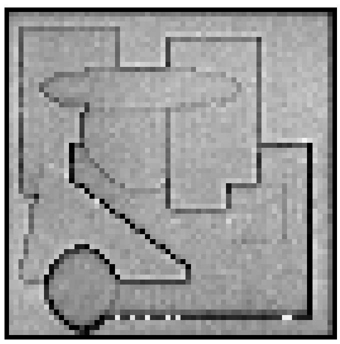
B =

0	0	0
0	1,75	0
0	0	0

z_1 = 1 z_2 = 0

Example 1 (resolution: 64x64): image name: avergra2.bmp, macro code: gradint_ace4k.amc.


input


output

Remarks:

- Masking works much better with DTCNN;
- The current values has to be set peculiarly for all the template operations;
- Central and non-central elements in the B templates behave different way…
- I used a simplified form of the original template: the diagonal directions are the sum of the horizontal and vertical gradients and so the diagonal template elements could be omitted;
- In this chip implementation averaging is omitted, because the algorithm has a bit superior performance, than the original nonlinear template (However, it can be included into the aververc.tem and averhorc.tem templates…).
- The 2nd order gradient can be computed much easier (for the above test images provide almost the same result): Antagonistic Center-surround template and an Absolute Value template.

DiagonalHoleDetection: ***Detects the number of diagonal holes from each diagonal line*** **[6]**

Old names: CCD_DIAG (Chua-Yang model)

A =

1	0	0
0	2	0
0	0	-1

B =

0	0	0
0	0	0
0	0	0

z = 0

I. Global Task

Given: static binary image **P**

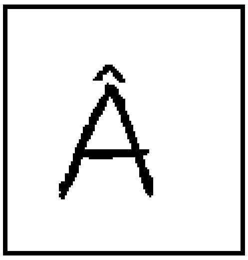
Input: **U(t) =** Arbitrary or as a default U(t)=0

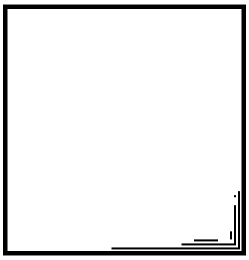
Initial State: **X(0) = P**

Boundary Conditions: Fixed type, y_{ij} = 0 for all virtual cells, denoted by [Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image that shows the number of diagonal holes in each diagonal line of image **P**.

II. Example: image name: a_letter.bmp, image size: 117x121; template name: ccd_diag.tem .


input


output

III. ACE4K implementation

Implementation method:

DiagonalHoleDetection_ACE4K: (Full-range model, ACE4K)

A =

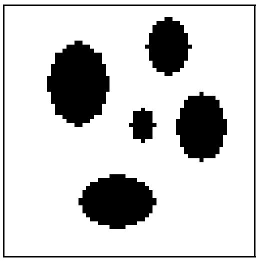
2.3	0	0
0	3	0
0	0	-2.3

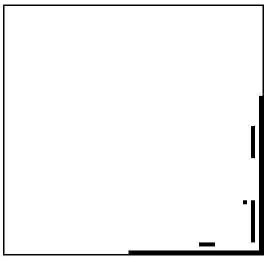
B =

0	0	0
0	0	0
0	0	0

z_1 = 0 z_2 = 0

Example 1 (resolution: 64x64): image name: sc_09.bmp, template name: *ccd_diag_se_ace4k.tem.*


input


output

Remarks:

- Image should be loaded into a LAM;
- Repeat template operation a few times.

CenterPointDetector: ***Center point detection*** **[21]**

<u>*Old names*</u>*: CENTER*

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \quad \mathbf{B}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = \boxed{-1}$$

$$\mathbf{A}_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad \mathbf{B}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_2 = \boxed{-1}$$

$$\mathbf{A}_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 4 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \mathbf{B}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_3 = \boxed{-1}$$

$$\mathbf{A}_4 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 6 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \mathbf{B}_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_4 = \boxed{-1}$$

. . .

$$\mathbf{A}_8 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 6 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{B}_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_8 = \boxed{-1}$$

I. Global Task

Given: static binary image **P**

Input: **U(t) =** Arbitrary or as a default U(t)=0

Initial State: **X(**0**) = P**

Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image where a black pixel indicates the center point of the object in **P**.

Remark:

The algorithm identifies the center point of the black-and-white input object. This is always a point of the object, halfway between the furthermost points of it. Here a DTCNN template sequence is given, each element of it should be used for a single step. It can easily be transformed to a continuous-time network:

CENTER1:

$$\mathbf{A}_1 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_1 = \begin{array}{|c|c|c|}\hline 1 & 0 & 0 \\ \hline 1 & 4 & -1 \\ \hline 1 & 0 & 0 \\ \hline\end{array} \qquad z_1 = \boxed{-1}$$

CENTER2:

$$\mathbf{A}_2 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_2 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 6 & 0 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \qquad z_2 = \boxed{-1}$$

CENTER3:

$$\mathbf{A}_3 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_3 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 4 & 0 \\ \hline 0 & -1 & 0 \\ \hline\end{array} \qquad z_3 = \boxed{-1}$$

CENTER4:

$$\mathbf{A}_4 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_4 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 6 & 1 \\ \hline -1 & 0 & 1 \\ \hline\end{array} \qquad z_4 = \boxed{-1}$$

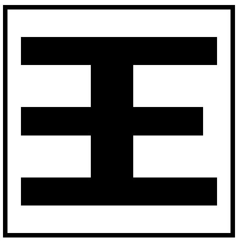
. . .

CENTER8:

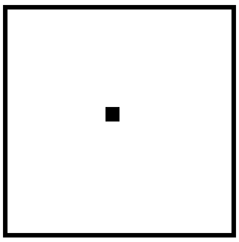
$$\mathbf{A}_8 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_8 = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 6 & 0 \\ \hline 1 & 1 & 1 \\ \hline\end{array} \qquad z_8 = \boxed{-1}$$

The robustness of templates CENTER1 and CENTER2 are ρ(CENTER1) = 0.22 and ρ(CENTER2) = 0.15, respectively. Other templates are the rotated versions of CENTER1 and CENTER2, thus their robustness values are equal to the mentioned ones.

II. Example: image name: chineese.bmp, image size: 16x16; template name: center.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

CenterPoint_ACE4K: (Full-range model, ACE4K)

Discrete time CNN implementation:

$$\mathbf{A}_1 = \begin{array}{|c|c|c|}\hline 01 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_1 = \begin{array}{|c|c|c|}\hline 1 & 0 & 0 \\ \hline 1 & 3 & -1 \\ \hline 1 & 0 & 0 \\ \hline\end{array} \qquad z_1 = \boxed{-1.95}$$

$$\mathbf{A}_2 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_2 = \begin{array}{|c|c|c|}\hline .66 & .66 & .66 \\ \hline .66 & 2 & 0 \\ \hline .66 & 0 & -.66 \\ \hline\end{array} \qquad z_2 = \boxed{-1}$$

$$\mathbf{A}_3 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_3 = \begin{array}{|c|c|c|}\hline .75 & .75 & .75 \\ \hline 0 & 3 & 0 \\ \hline 0 & -.75 & 0 \\ \hline\end{array} \qquad z_3 = \boxed{-.75}$$

$$\mathbf{A}_4 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 3 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_4 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 3 & 1 \\ \hline -1 & 0 & 1 \\ \hline\end{array} \qquad z_4 = \boxed{-1.95}$$

. . .

$$\mathbf{A}_8 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 3 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_8 = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 3 & 0 \\ \hline 1 & 1 & 1 \\ \hline\end{array} \qquad z_8 = \boxed{-1.95}$$

Continuous time CNN implementation:

$$\mathbf{A}_1 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_1 = \begin{array}{|c|c|c|}\hline 1 & 0 & 0 \\ \hline 1 & 3 & -1 \\ \hline 1 & 0 & 0 \\ \hline\end{array} \qquad z_1 = \boxed{-2.5}$$

$$\mathbf{A}_2 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_2 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 3 & 0 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \qquad z_2 = \boxed{-6}$$

$$\mathbf{A}_3 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1.3 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_3 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 2 & 0 \\ \hline 0 & -1 & 0 \\ \hline\end{array} \qquad z_3 = \boxed{-6}$$

$$\mathbf{A}_4 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \qquad \mathbf{B}_4 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 3 & 1 \\ \hline -1 & 0 & 1 \\ \hline\end{array} \qquad z_4 = \boxed{-6}$$

. . .

$\mathbf{A}_8$ =

0	0	0
0	-1	0
0	0	0

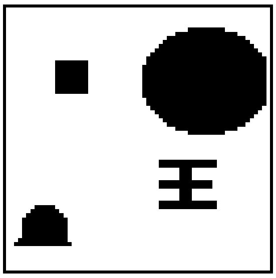
$\mathbf{B}_8$ =

1	0	-1
1	3	0
1	1	1

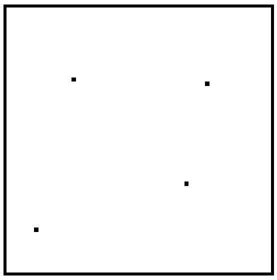
z_8 =

-6

Example 2 (resolution: 64x64): image name: conv2.bmp, AMC and ALPHA file names: Cpd_ace4k.amc & Centerpointch.alf.



input



output

Remarks:

- Before template running LLMs must be initialized with white;
- The logic values (zeros and ones) seems to be reversed in the chip…
- Both the discrete and continuous time CNN implementations are realizable on chip, but the later one is more sensitive for timing etc.
- The current values are extremely depends on the temperature of the chip and does not accord to the theoretical values…
- Continuous time version sometimes fail…

ContourExtraction: ***Grayscale contour detector*** **[8]**

Old names: ContourDetector, CONTOUR

A =

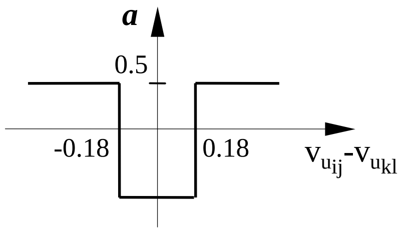
0	0	0
0	2	0
0	0	0

B =

a	***a***	***a***
a	0	***a***
a	***a***	***a***

z = 0.7

where ***a*** is defined by the following nonlinear function:



I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) = P**
Initial State:	**X(0) = P**
Boundary Conditions	Fixed type, u_{ij} = 0 for all virtual cells, denoted by [U]=0
Output:	**Y(t)⇒Y(∞)** = Binary image where black pixels represent the contours of the objects in **P**.

Remark:

The template extracts contours which resemble edges (resulting from big changes in gray level intensities) from grayscale images.

II. Example: image name: madonna.bmp, image size: 59x59; template name: contour.tem .

input

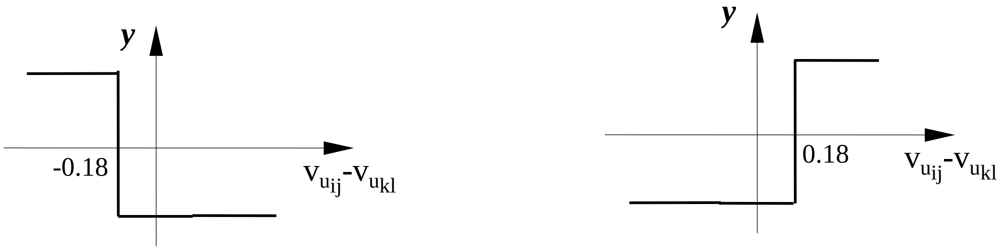
output

III. ACE4K implementation

Implementation method: decomposition and optimization.

Due to the hardware limitations the nonlinear template is replaced by eight pairs of linear B template. Templates check the difference between the central element and its nearest neighboring cells in eight directions. If the differences exceed a given threshold in certain number of directions, the pixel will be set to black, otherwise to white.

Only 1 of the 8 required template-pairs are shown, the others can easily be generated by rotating value +-3 of template B.



Horizontally, on the right:

Horizontally, on the right:

A =

0	0	0
0	-3	0
0	0	0

B =

0	0	0
0	3	-3
0	0	0

z_1 = -0.1 z_2 = 0

With opposite sign:

A =

0	0	0
0	-3	0
0	0	0

B =

0	0	0
0	-3	3
0	0	0

z_1 = -0.1 z_2 = 0

Results of the subsequent template-operations in each direction are summarized in one picture. Finally, a threshold operation should be applied:

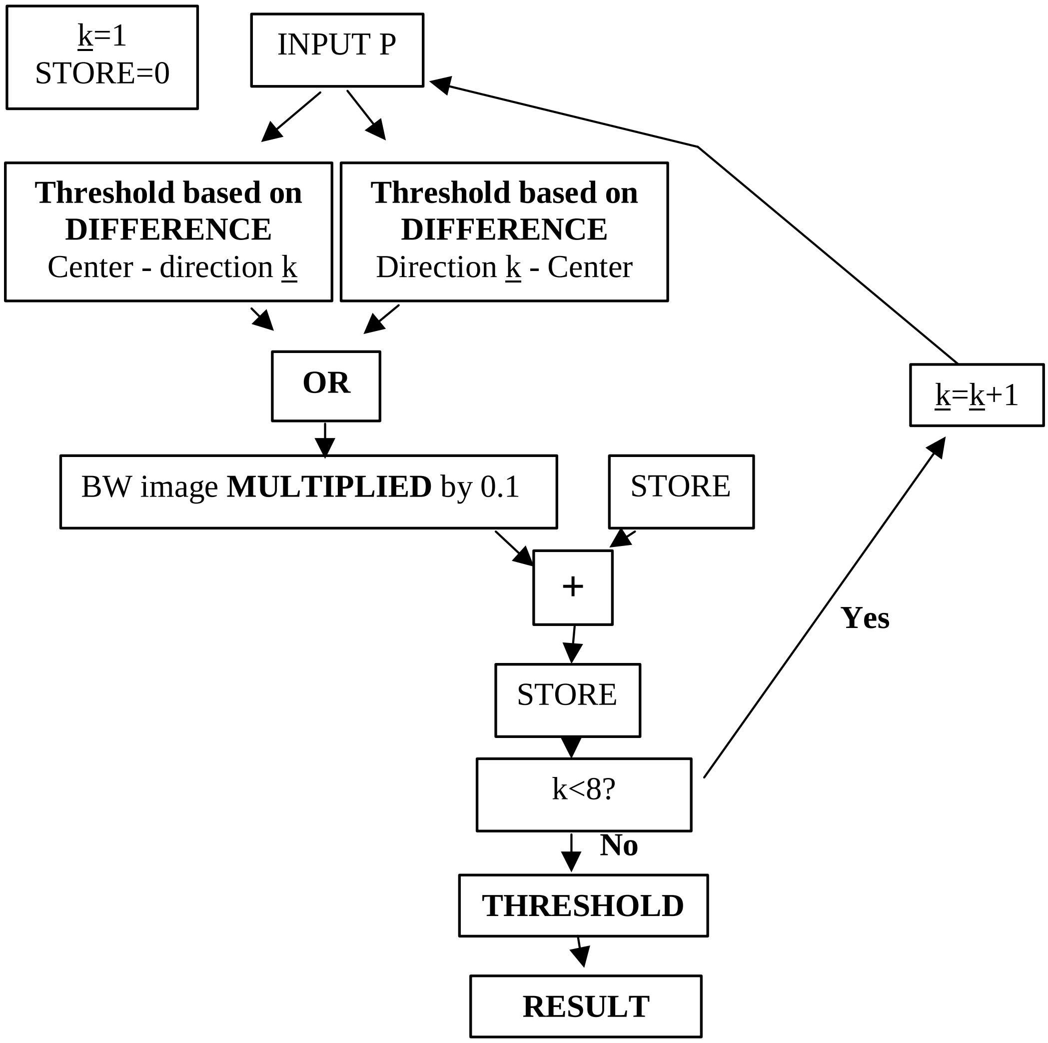
A =

0	0	0
0	-3	0
0	0	0

B =

0	0	0
0	3	0
0	0	0

z_1 = 1 z_2 = 0



Example (resolution: 64x64): image name: madonna.bmp.

input

output

- Local LLMs have to be initialized (filled up with zero).
- State capacitors of the cells are to be initialized as well: an appropriate template operation drives their values to +1 before each grayscale-to-binary operation.
- Each grayscale-to-binary operation is executed at least four times successively with the same input. Results are combined via AND logical operation. The net effect is a sort of noise-filtering: false positive (black) pixels are erased.

CornerDetection: ***Convex corner detection template*** **[1]**

<u>Old names</u>: CornerDetector, CORNER

CornerDetection (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

-1	-1	-1
-1	4	-1
-1	-1	-1

z = -5

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)

Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

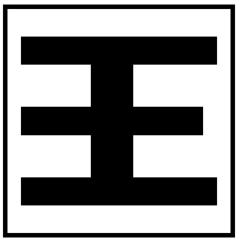
Output: **Y(t)⇒Y(∞)** = Binary image where black pixels represent the convex corners of objects in **P**.

Template robustness: ρ = 0.2 .

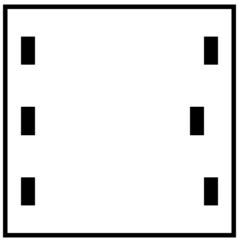
Remark:

Black pixels having at least 5 white neighbors are considered to be convex corners of the objects.

II. Example: image name: chineese.bmp, image size: 16x16; template name: corner.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

CornerDetection_ACE4K: (Full-range model, ACE4K)

A =

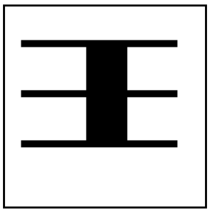
0	0	0
0	3	0
0	0	0

B =

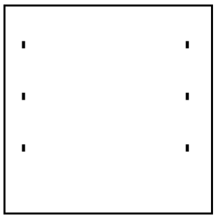
-1.3	-1.3	-1.3
-1.3	0	-1.3
-1.3	-1.3	-1.3

z_1 = -5 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: corner.bmp, template name: cornerdetection_ace4k.tem.



input



output

Remarks:

- We can use this template in the LAMs.

VerticalLineRemover: ***Deletes vertical lines*** **[8]**

Old names: DELVERT1

Delvert1 (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

0	-1	0
0	1	0
0	-1	0

z = -2

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)

Boundary Conditions: Fixed type, u_{ij} = -1, for all virtual cells, denoted by [U] = -1

Output: **Y(t)⇒Y(∞)** = Binary image representing **P** without vertical lines. Those parts of the objects that could be interpreted as vertical lines will also be deleted.

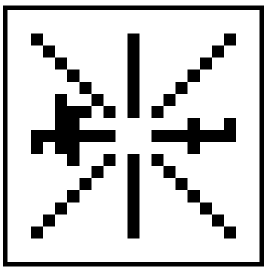
Template robustness: ρ = 0.58 .

Remark:

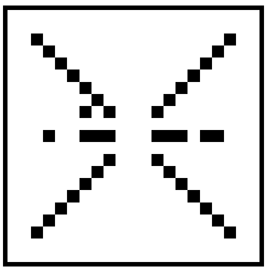
The template deletes every black pixel having either a northern or southern black neighbor.

The *HorizontalLineRemover* template, that deletes one pixel wide horizontal lines, can be obtained by rotating the *VerticalLineRemover* by 90°. The functionality of the WIREHOR and WIREVER templates that were published in earlier versions of this library, is identical to the functionality of the *HorizontalLineRemover* and *VerticalLineRemover* templates.

II. Example: image name: delvert1.bmp, image size: 21x21; template name: delvert1.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

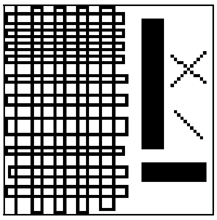
Delvert_ACE4K: (Full-range model, ACE4K)

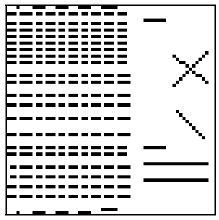
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1.2 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad z_1 = \boxed{-3} \quad z_2 = \boxed{0}$$

Delhor_ACE4K: (Full-range model, ACE4K)

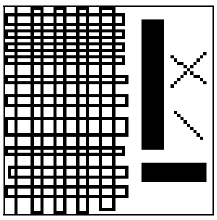
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1.2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = \boxed{-3} \quad z_2 = \boxed{0}$$

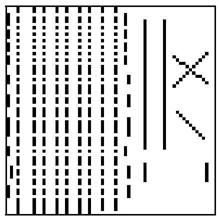
Example 1 (resolution: 64x64): image name: delverthor_i.bmp, template name: delvert_ace4k.tem.


input

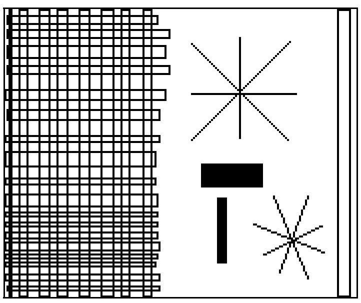

output

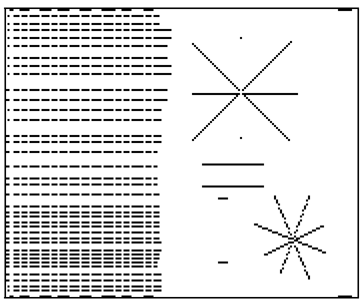
template name: delhor_ace4k.tem.


input

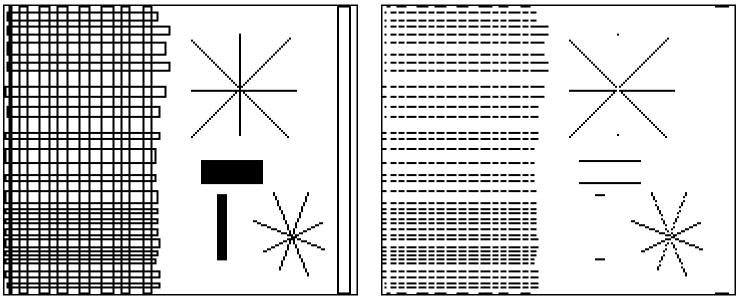

output

Example 2 (resolution: 176x144): image name: delverthor_qcif_i.bmp, template name: delvert_ace4k.tem.


input


output

template name: delhor_ace4k.tem.



input output

Remarks:

- Template operations should be executed twice successively.

DiagonalLineDetector: ***Diagonal-line-detector template***

Old names: DIAG1LIU, DetSWNE (Chua-Yang model)

A =

0	0	0
0	1	0
0	0	0

B =

-1	0	1
0	1	0
1	0	-1

z = -4

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) = P**

Boundary Conditions: Fixed type, u_{ij} = -1 for all virtual cells, denoted by [U]=-1

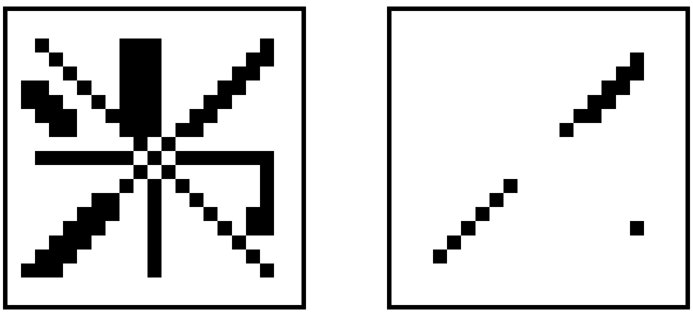
Output: **Y(t)⇒Y(∞)** = Binary image representing the locations of diagonal lines in **P**.

Template robustness: ρ = 0.45 .

Remark:

Detects every black pixel having black north-eastern, black south-western, white north-western, and white south-eastern neighbors. It may be used for detecting diagonal lines being in the SW-NE direction (like /). By modifying the position of the ±1 values of the **B** template, the template can be sensitized to other directions as well (vertical, horizontal or NW-SE diagonal).

II. Example: image name: diag1liu.bmp, image size: 21x21; template name: diag1liu.tem .



input output

III. ACE4K implementation

Implementation method: optimization

DIAG1LIU_ACE4K: (Full-range model, ACE4K)

A =

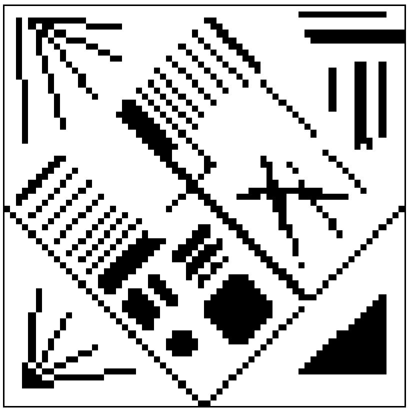
0	0	0
0	2	0
0	0	0

B =

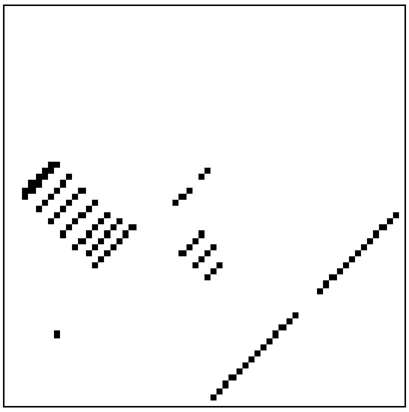
-1	0	1
0	1	0
1	0	-1

z_1 = -5 z_2 = -5

Example 1 (resolution: 64x64): image name: detdiag.bmp, template name: diag1liu_ace4k.tem.

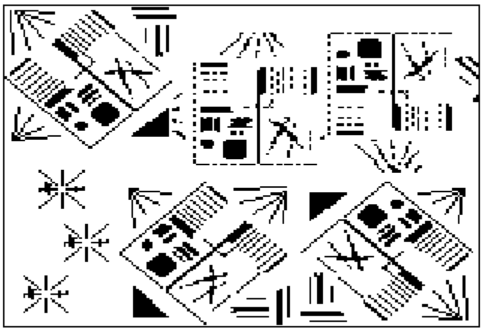


input

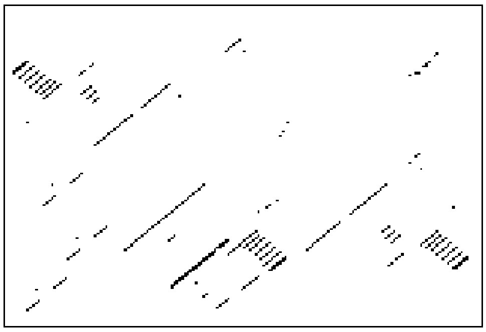


output

Example 2 (resolution: 176x144): image name: detdiag_tile.bmp, template name: diag1liu_ace4k.tem.



input



output

Remarks:

- By modifying the position of the ±1 values of the **B** template, the template can be sensitized to other directions as well (vertical, horizontal or NW-SE diagonal).
- hw.set.ref 0 60 -85 -110 -3 -55 51 113 84 ;nominal setting for template run

EdgeDetection: ***Binary edge detection template***

<u>Old names</u>: *EdgeDetector, EDGE*

EdgeDetection (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

-1	-1	-1
-1	8	-1
-1	-1	-1

z = -1

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)

Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

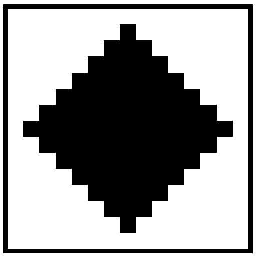
Output: **Y(t)⇒Y(∞)** = Binary image showing all edges of **P** in black

Template robustness: ρ = 0.12 .

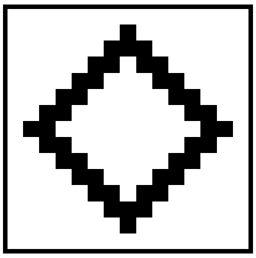
Remark:
Black pixels having at least one white neighbor compose the edge of the object.

II. Example

<u>Example:</u> image name: logic05.bmp, image size: 44x44; template name: edge.tem .



input



output

III. ACE4K implementation

Implementation method: template decomposition and optimization.

Result: *EdgeDetection ⇔ Edge1* ***AND*** *INPUT*

Edge1 (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

-1	-1	-1
-1	0	-1
-1	-1	-1

z = 7

Edge1_ACE4K: *(Full-range model, ACE4K)*

A =

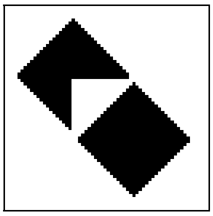
0	0	0
0	3	0
0	0	0

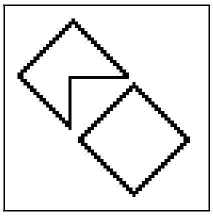
B =

-1	-1	-1
-1	0	-1
-1	-1	-1

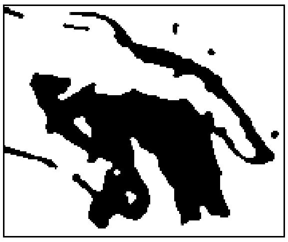
z_1 = 6 z_2 = 4.8

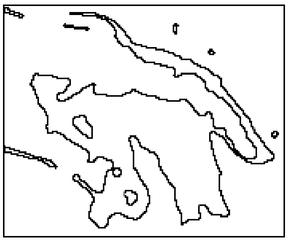
<u>Example 1</u> (resolution: 64x64): image name: edge64_i.bmp, template name: edge1_ace4k.tem.


input


output

<u>Example 2</u> (resolution: 176x144): image name: michel_qcif_i.bmp, template name: edge1_ace4k.tem.


input


output

Remarks:

- Template operations should be executed twice in a row.

OptimalEdgeDetector: ***Optimal edge detector [43]***

OptimalEdgeDetector (Chua-Yang model):

A =

0	0	0
0	0	0
0	0	0

B =

-0.11	0	0.11
-0.28	0	0.28
-0.11	0	0.11

z = 0

I. Global Task

Given: static grayscale image **P**

Input: **U(t) = P**

Initial State: **X**(0) **=** Arbitrary (in the examples we choose $x_{ij}(0)=0$)

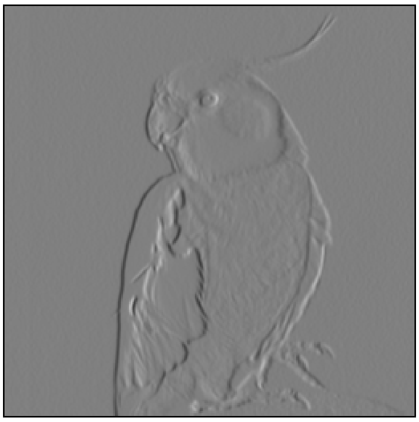
Boundary Conditions: Zero-flux boundary condition (duplicate)

Output: **Y(t)⇒Y(∞)** = Grayscale image representing edges calculated in horizontal direction.

Remark:
The **B** template represents the optimal edge detector operator.

II. Example: image name: bird.bmp, image size: 256x256; template name: optimedge.tem .

input



output

III. ACE4K implementation

Implementation method: optimization.

OptimalEdgeDetector_ACE4K: *(Full-range model, ACE4K)*

A =

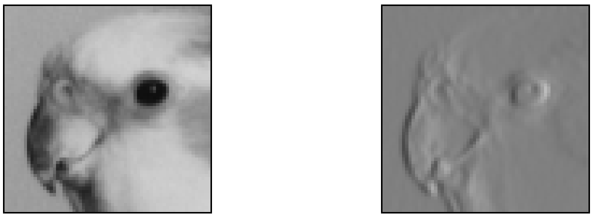
0	0	0
0	-2	0
0	0	0

B =

-0.33	0	0.33
-0.84	0	0.84
-0.33	0	0.33

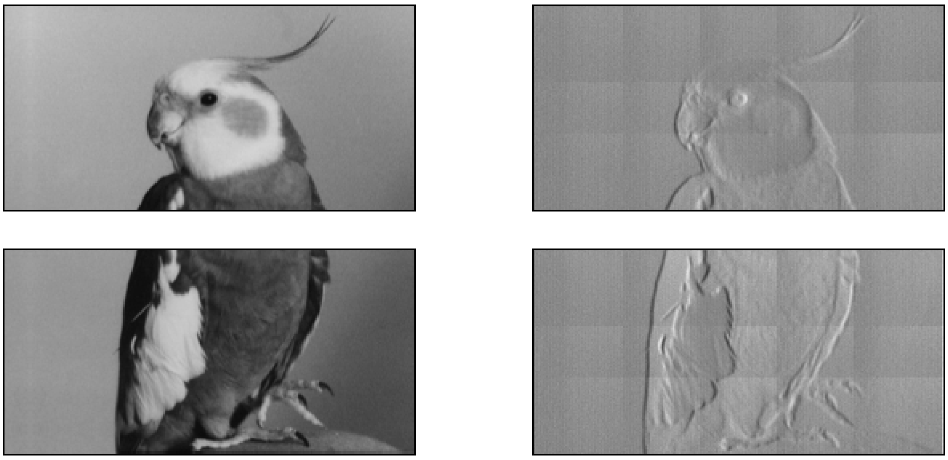
z_1 = 0.7 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: bird64_i.bmp, template name: optimedge_ace4k.tem.



input output

<u>Example 2</u> (resolution: 256x128): image names: bird01.bmp, bird02.bmp; template name: optimedge_ace4k.tem.



inputs outputs

Remarks:

- Due to memory problems the original 256x256 image was cut into 256x128 sub-images, and processed in two phases.

PointExtraction: ***Extracts isolated black pixels***

<u>Old names</u>: FigureRemover, FIGDEL

PointExtraction (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

-1	-1	-1
-1	1	-1
-1	-1	-1

z = -8

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

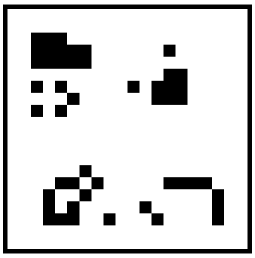
Initial State: **X(0) =** Arbitrary

Boundary Conditions: Fixed type, u_{ij} = 0 for all virtual cells, denoted by [U]=0

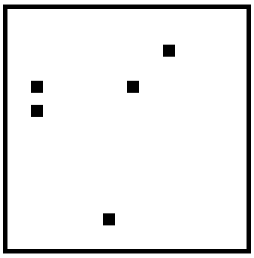
Output: **Y(t)⇒Y(∞)** = Binary image representing all isolated black pixels in **P**.

Template robustness: ρ = 0.33 .

II. Example: image name: figdel.bmp, image size: 20x20; template name: figdel.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

PointExtraction_ACE4K: (Full-range model, ACE4K)

A =

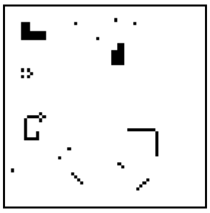
0	0	0
0	1.5	0
0	0	0

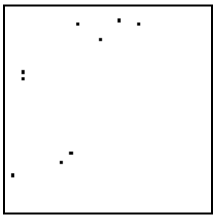
B =

-1	-1	-1
-1	0.5	-1
-1	-1	-1

z_1 = -6 z_2 = 0

Example 1 (resolution: 64x64): image name: points.bmp, template name: PointExtraction_ace4k.tem.


input


output

Remarks:

- This template can be used in LAMs.

PointRemoval: ***Deletes isolated black pixels***

Old names: FigureExtractor, FIGEXTR

PointRemoval (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

1	1	1
1	8	1
1	1	1

z = -1

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) =** Arbitrary

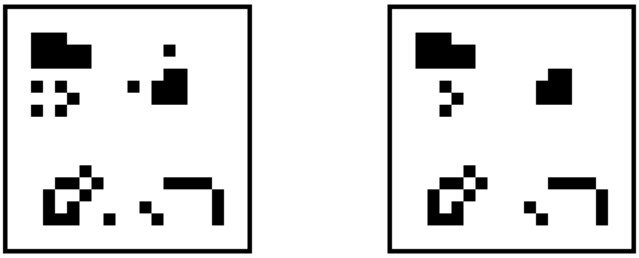
Boundary Conditions: Fixed type, u_{ij} = 0 for all virtual cells, denoted by [U]=0

Output: **Y(t)⇒Y(∞)** = Binary image showing all connected components in **P**.

Remark:

Black pixels having no black neighbors are deleted. This template is the opposite of *PointExtraction*.

II. Example: image name: figdel.bmp, image size: 20x20; template name: figextr.tem .



input output

III. ACE4K implementation

Implementation method: optimization.

PointRemoval_ACE4K: (Full-range model, ACE4K)

A =

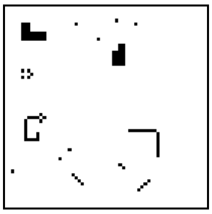
0	0	0
0	-2	0
0	0	0

B =

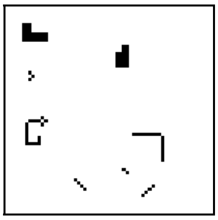
0.5	0.5	0.5
0.5	3	0.5
0.5	0.5	0.5

z_1 = 0.5 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: points.bmp, template name: PointRemoval_ace4k.tem.



input



output

Remarks:

1) We can use this template in the LLMs.

SelectedObjectsExtraction: Extracts marked objects

<u>Old names</u>: FigureReconstructor, FIGREC, RECALL (Chua-Yang model)

SelectedObjectsExtraction (Chua-Yang model):

A =

0.5	0.5	0.5
0.5	4	0.5
0.5	0.5	0.5

B =

0	0	0
0	4	0
0	0	0

z = 3

I. Global Task

Given: two static binary images $\mathbf{P_1}$ (mask) and $\mathbf{P_2}$ (marker). $\mathbf{P_2}$ contains just a part of $\mathbf{P_1}$ ($\mathbf{P_2} \subset \mathbf{P_1}$).

Input: $\mathbf{U(t) = P_1}$

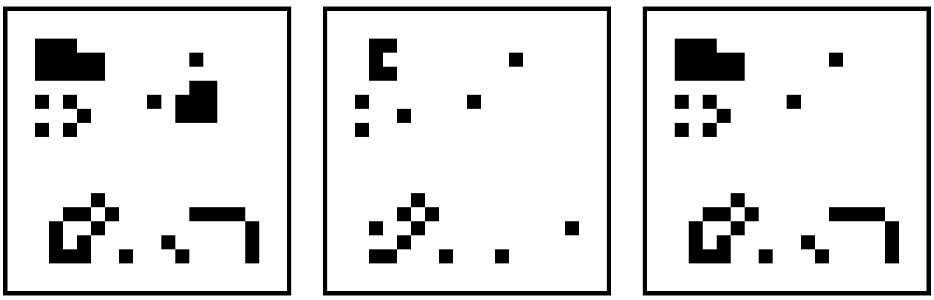
Initial State: $\mathbf{X(0) = P_2}$

Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image representing those objects of $\mathbf{P_1}$ which are marked by $\mathbf{P_2}$.

Template robustness: $\rho = 0.12$.

II. Example: image names: figdel.bmp, figrec.bmp; image size: 20x20; template name: figrec.tem



input initial state output

III. ACE4K implementation

Implementation method: *Recall operation is based on fixed state mask. Marker image (P_2) should be fed into the initial state (LLM1) while the mask image (P_1) should be placed into fixed state (LLM4).*

SelectedObjectsExtraction_ACE4K: (Full-range model, ACE4K)

A =

0.41	0.59	0.41
0.59	1.80	0.59
0.41	0.59	0.41

B =

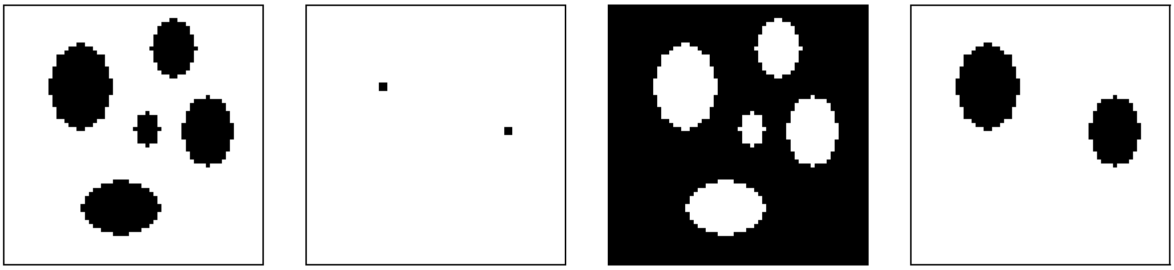
0	0	0
0	0	0
0	0	0

z_1 = 4 z_2 = 0

Remarks:

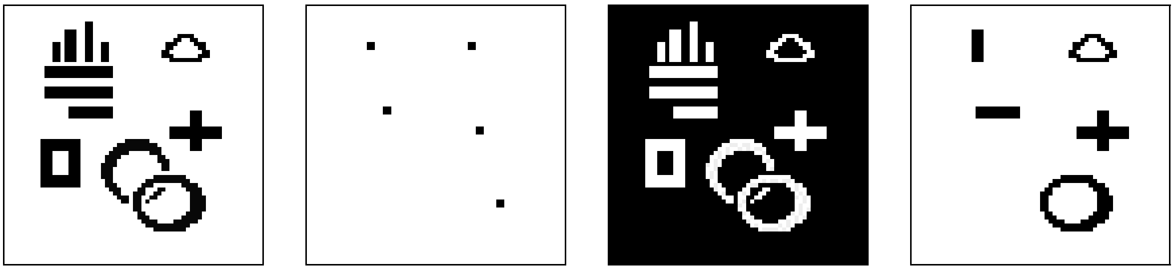
- Initial state should be a LLM;
- Fixed state should be the LLM4 and white pixels denotes for positions where cell transient can take place.
- Input image should be a LAM filled with zero.

<u>Example 1</u> (resolution: 64x64): image name: recall1.bmp, template name: *recall_ace4k.tem.*



initial state markers fixed state output

<u>Example 2</u> (resolution: 64x64): image name: recall2.bmp, template name: *recall_ace4k.tem.*



initial state markers fixed state output

3x3Halftoning: 3x3 image halftoning

<u>Old names</u>: HLF3, HLF33

3x3Halftoning (Chua-Yang model):

A =

-0.07	-0.1	-0.07
-0.1	1+ε	-0.1
-0.07	-0.1	-0.07

B =

0.07	0.1	0.07
0.1	0.32	0.1
0.07	0.1	0.07

z = 0

I. Global Task

Given: static grayscale image **P**

Input: **U(t) = P**

Initial State: **X(0) = P**

Boundary Conditions: Fixed type, $u_{ij} = 0$, $y_{ij} = 0$ for all virtual cells, denoted by [U]=[Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image preserving the main features of **P**.

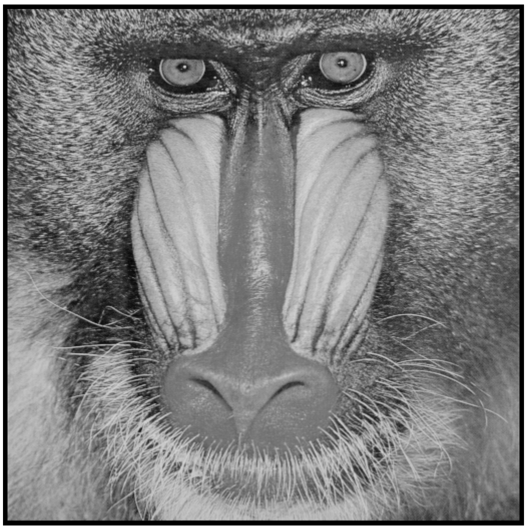
Remark:

The speed of convergence is controlled by ε≈[0.1...1]. The greater the ε is, the faster the process and the rougher the result will be. The inverse of the template is *3x3InverseHalftoning*. The result is acceptable in the Square Error measure [17,35].

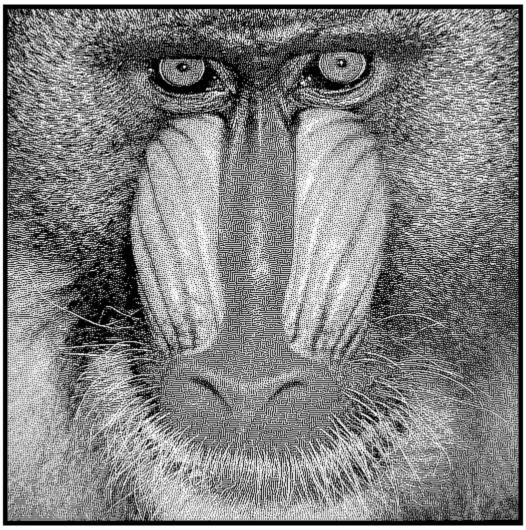
This template is called "Half-Toning" in [44].

II. Examples

<u>Example 1:</u> image name: baboon.bmp, image size: 512x512; template name: hlf3.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

3x3Halftoning_ACE4K: (Full-range model, ACE4K)

A =

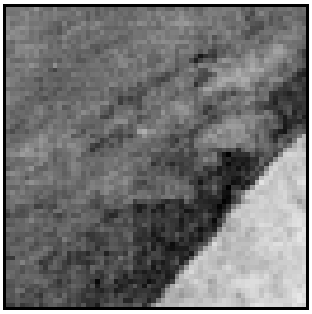
-0.07	-0.15	-0.07
-0.15	1.15	-0.15
-0.07	-0.15	-0.07

B =

0.07	0.15	0.07
0.15	0.15	0.15
0.07	0.15	0.07

z_1 = 1.25 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: michelan64.bmp, template name: halftc.tem



input

output

Remarks:

- There is some bias on the chip surface (top right corner is whiter, than it should be…).

Hole-Filling: ***Fills the interior of all closed contours*** **[6]**

<u>Old names</u>: HoleFiller, HOLE

Hole-Filling (Chua-Yang model):

A =

0	1	0
1	3	1
0	1	0

B =

0	0	0
0	4	0
0	0	0

z = -1

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X**(0) **= 1**

Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

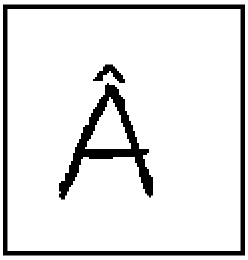
Output: **Y(t)⇒Y(∞)** = Binary image representing **P** with holes filled.

Remark:

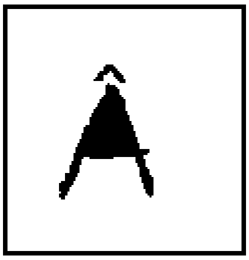
(i) this is a propagating template, the computing time is proportional to the length of the image

(ii) a more powerful template is the *ConcaveLocationFiller* template in this library.

II. Example: image name: a_letter.bmp, image size: 117x121; template name: hole.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

Hole-Filling_ACE4K: (Full-range model, ACE4K)

A =

0	0.68	0
0.68	1.71	0.68
0	0.68	0

B =

0	0	0
0	3	0
0	0	0

z_1 = -0.95 z_2 = -6

Hole-FillingDT_ACE4K: (DT-CNN mode, ACE4K)

A =

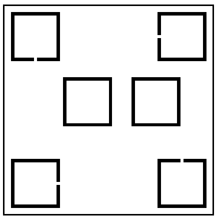
0	3	0
3	1	3
0	3	0

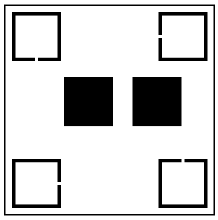
B =

0	0	0
0	3	0
0	0	0

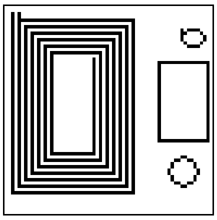
z_1 = 4 z_2 = 0

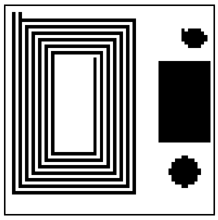
<u>Example 1</u> (resolution: 64x64): image name: hole_i.bmp, template name: hole_ace4k.tem, holeDT_ace4k.tem.


input


output

<u>Example 2</u> (resolution: 64x64): image name: labyrinth.bmp, template name: hole_ace4k.tem, holeDT_ace4k.tem.


input


output

Remarks:

- Special settings (continuos mode): hw.set.ref 0 60 -90 -81 16 -51 51 113 84
- Special settings (DTCNN mode): hw.set.mode 1 2

ObjectIncreasing: ***Increases the object by one pixel (DTCNN)*** **[16]**

<u>Old names</u>: *INCREASE*

A =

0.5	0.5	0.5
0.5	0.5	0.5
0.5	0.5	0.5

B =

0	0	0
0	0	0
0	0	0

z = 4

I. Global Task

Given: static binary image **P**

Input: **U(t) =** Arbitrary or as a default U(t)=0

Initial State: **X(**0**) = P**

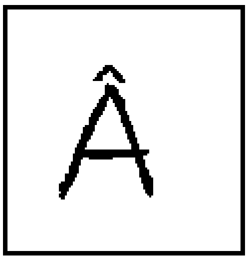
Boundary Conditions: Zero-flux boundary condition (duplicate)

Output: **Y(1)** = Binary image representing the objects of **P** increased by 1 pixel in all direction.

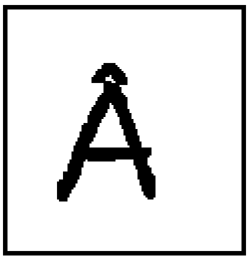
Remark:

Increasing the size of an object by *N* pixels in all directions can be achieved by *N* iteration steps of a DTCNN.

II. Example: image name: a_letter.bmp, image size: 117x121; template name: increase.tem . One iteration step of a DTCNN is performed.



input



output

III. ACE4K implementation

Implementation method: optimization.

ObjectIncreasing_ACE4K: (Full-range model, ACE4K) 1.

A =

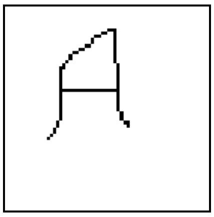
0	1.6	0
1.6	-3	1.6
0	1.6	0

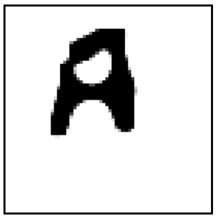
B =

0	0	0
0	3	0
0	0	0

z_1 = 0 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: aletter.bmp, template name: cornerdetection_ace4k.tem.


input


output

ObjectIncreasing_ACE4K: (Full-range model, ACE4K) 2.

A =

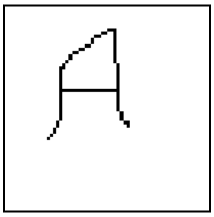
0	0	0
0	-2	0
0	0	0

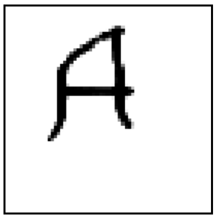
B =

0.5	0.5	0.5
0.5	2	0.5
0.5	0.5	0.5

z_1 = 0 z_2 = 0

<u>Example 2</u> (resolution: 64x64): image name: aletter.bmp, template name: cornerdetection_ace4k.tem.


input


output

ObjectIncreasing_ACE4K: (Full-range model, ACE4K) 3.

A =

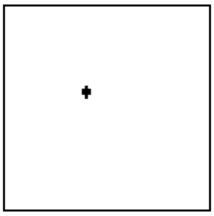
0	0	0
0	-3	0
0	0	0

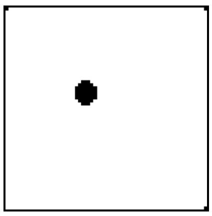
B =

0.5	0.5	0.5
0.5	1.2	0.5
0.5	0.5	0.5

z_1 = 5 z_2 = 0

<u>Example 3</u> (resolution: 64x64): image name: circle.bmp, template name: cornerdetection_ace4k.tem.


input


output

Remarks:

- The Examples 1 and 2 were run in the LAMs. The Example 3 was run in the LLMs.

LocalSouthernElementDetector: ***Local southern element detector*** **[11]**

<u>Old names</u>: LSE

LocalSouthernElementDetector (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

0	0	0
0	1	0
-1	-1	-1

z = -3

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) =** Arbitrary

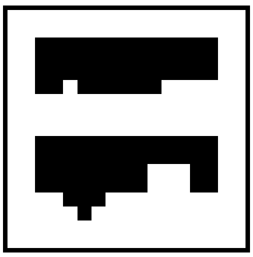
Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

Output: **Y(t)⇒Y(∞)** = Binary image representing local southern elements of objects in **P**.

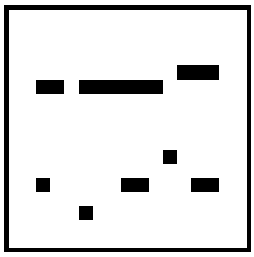
Remark:

Local southern elements are pixels having neither south-western, nor southern or south-eastern neighbors.

II. Example: image name: lcp_lse.bmp, image size: 17x17; template name: lse.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

LocalSouthernElemenDetector_ACE4K: (Full-range model, ACE4K)

A =

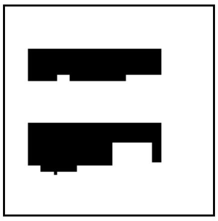
0	0	0
0	1	0
0	0	0

B =

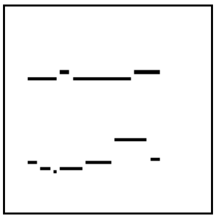
0	0	0
0	1	0
0	0	0

z_1 = -3.5 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: local.bmp, template name: localsouthernelementdetector_ace4k.tem.



input



output

Remarks:

- This template can be used in the LAMs.

RightEdgeDetection: ***Extracts right edges of objects***

Old names: RightContourDetector, RIGHTCON

RightEdgeDetection (Chua-Yang model):

A =

0	0	0
0	1	0
0	0	0

B =

0	0	0
1	1	-1
0	0	0

z = -2

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) = P**

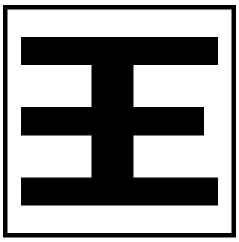
Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

Output: **Y(t)⇒Y(∞)** = Binary image representing the right edges of objects in **P**.

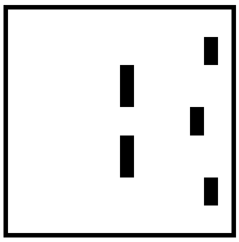
Template robustness: $\rho = 0.58$.

Remark:
By rotating **B** the template can be sensitized to other directions as well.

II. Example: image name: chineese.bmp, image size: 16x16; template name: rightcon.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

RightEdgeDetection_ACE4K: (Full-range model, ACE4K)

A =

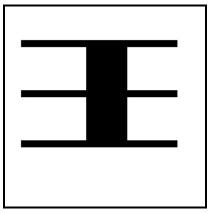
0	0	0
0	2	0
0	0	0

B =

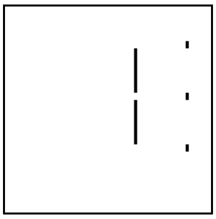
0	0	0
2	3	-2
0	0	0

z_1 = -6 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: corner.bmp, template name: rightedgedetection_ace4k.tem.



input



output

Remarks:

- This template can be used in the LAMs.

ShadowProjection: ***Projects onto the left the shadow of all objects illuminated from the right [6]***

Old names: LeftShadow, SHADOW

ShadowProjection (Chua-Yang model):

A =

0	0	0
0	2	2
0	0	0

B =

0	0	0
0	2	0
0	0	0

z = 0

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) = 1**

Boundary Conditions: Fixed type, y_{ij} = 0 for all virtual cells, denoted by [Y]=0

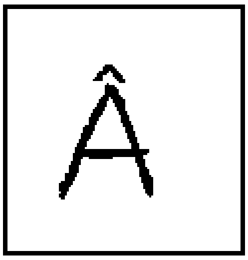
Output: **Y(t) Y( )** = Binary image representing the left shadow of the objects in **P**.

Template robustness: = 0.12 .

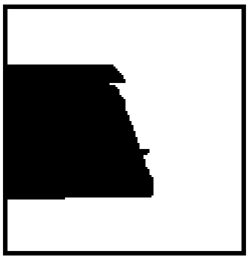
Remark:
The shadow is the projection in direction left of the black pixels.

II. Example

Example: Left shadow. Image name: a_letter.bmp, image size: 117x121; template name: shadow.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

ShadowProjection _ACE4K: (Full-range model, ACE4K)

A =

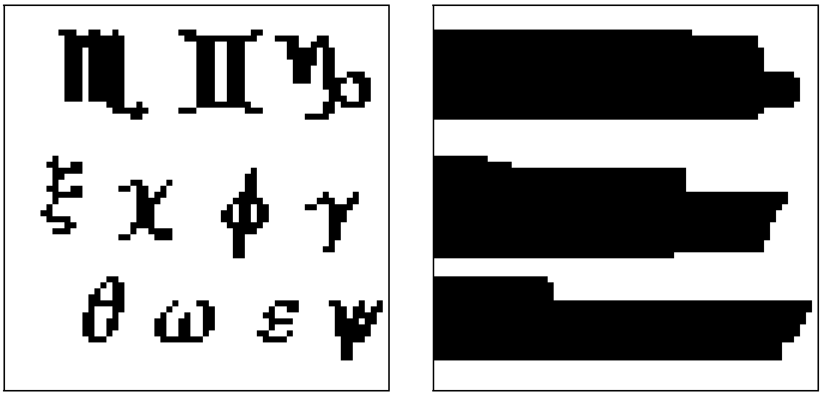
0	0	0
0	2	2
0	0	0

B =

0	0	0
0	2	0
0	0	0

z_1 = 1 z_2 = 0

Example 1 (resolution: 64x64): image name: shadow.bmp, template name: shadow_ace4k.tem.



input output

Remarks:

- The execution of the template could not be solved using LLM-s. The loading of logical TRUE in the initial state was also faulty. There was no problem by using LAM-s. LAM with value 1 was used for the initial state.
- The template worked only in a loop, after many executions. The range of the shadow effect increased continuously in the repetitions.

VerticalShadow: ***Vertical shadow template***

Old names: SHADSIM, SUPSHAD

VerticalShadow (Chua-Yang model):

A =

0	1	0
0	2	0
0	1	0

B =

0	0	0
0	0	0
0	0	0

z = 2

I. Global Task

Given: static binary image **P**

Input: **U(t) =** Arbitrary

Initial State: **X(**0**) = P**

Boundary Conditions: Zero-flux boundary condition (duplicate)

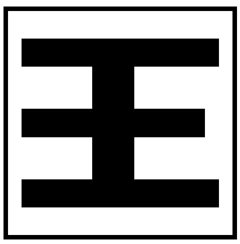
Output: **Y(t) Y( )** = Binary image representing the vertical shadow of the objects in **P** taken upward and downward simultaneously.

Template robustness: = 0.12 .

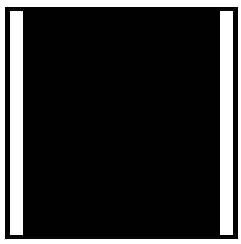
Remark: The vertical shadow is the union of those columns, which contain at least one black pixel.

II. Example

Example: image name: chineese.bmp, image size: 16x16; template name: shadsim.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

VerticalShadow _ACE4K: (Full-range model, ACE4K)

A =

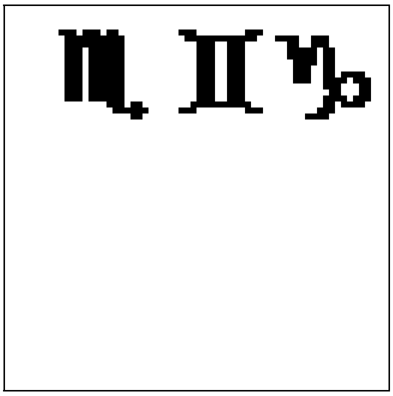
0	2	0
0	0	0
0	2	0

B =

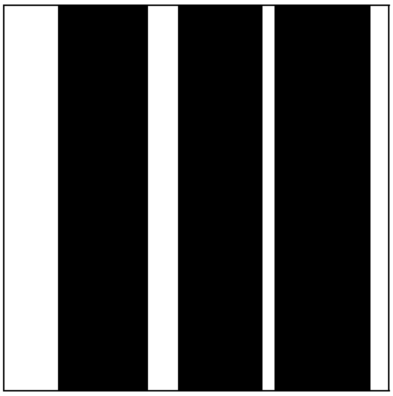
0	0	0
0	0	0
0	0	0

z_1 = 3.5 z_2 = 0

Example 1 (resolution: 64x64): image name: skelbwi64.bmp, template name: shadsim_ace4k.tem.



input



output

Remarks:

- The execution of the template could not be solved using LLM-s.
- The template worked only in a loop, after many executions. The range of the shadow effect increased continuously in the repetitions.

1.3. SPATIAL LOGIC

ConcaveLocationFiller: ***Fills the concave locations of objects*** **[22]**

<u>Old names</u>: HOLLOW

ConcaveLocationFiller (Chua-Yang model):

A =

0.5	0.5	0.5
0.5	2	0.5
0.5	0.5	0.5

B =

0	0	0
0	2	0
0	0	0

z = 3.25

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) = P**

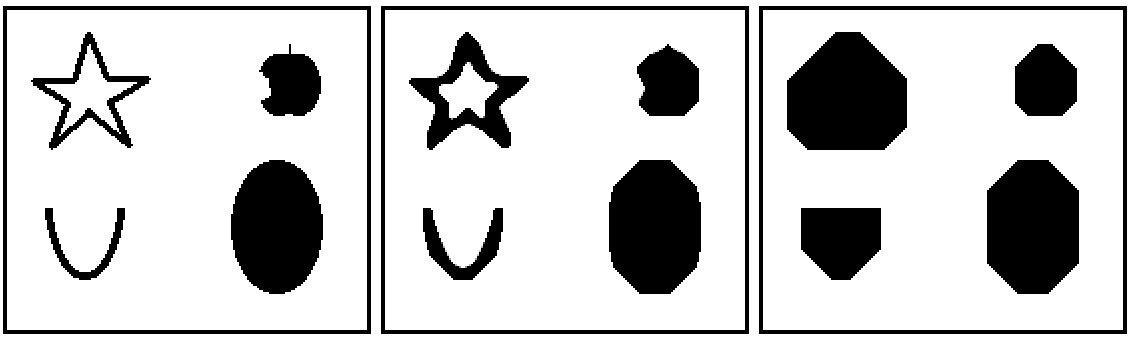
Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image in which the concave locations of objects are black.

Remark:

In general, the objects of **P** that are not filled should have at least a 2-pixel-wide contour. Otherwise the template may not work properly.
The template transforms all the objects to solid black concave polygons with vertical, horizontal and diagonal edges only.

II. Example: image name: hollow.bmp, image size: 180x160; template name: hollow.tem .



input output ($t=20\tau_{CNN}$) output ($t=\infty$)

III. ACE4K implementation

Implementation method:

ConcaveLocationFiller_*ACE4K: (Full-range model, ACE4K)*

A =

0.5	0.5	0.5
0.5	-3	0.5
0.5	0.5	0.5

B =

0	0	0
0	2.5	0
0	0	0

z_1 = - z_2 = 1.0

Example (resolution: 64x64): image names: inputCLF.bmp, white.bmp; template name: hollow_ace4k.tem.

input output after 100 it. output after 200 it.

Remarks:

- Initial State: white.bmp

GrayscaleLineDetector: *Grayscale line detector template*

Old names: LINE3060

A =

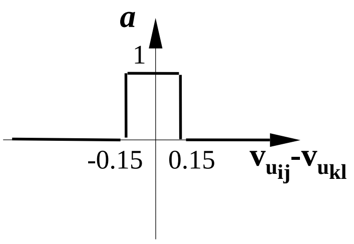
0	0	0
0	1.5	0
0	0	0

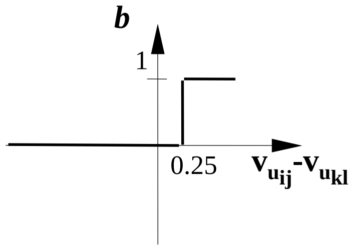
B =

b	***a***	***a***
b	***0***	***a***
a	***b***	***b***

z = -4.5

where ***a*** and ***b*** are defined by the following nonlinear functions:





I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X**(0) = **0**

Boundary Conditions: Zero-flux boundary condition (duplicate)

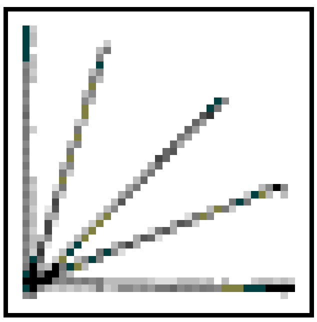
Output: **Y(t)⇒Y(∞)** = Binary image where black pixels correspond to the grayscale lines within a slope range of approximately 30° (30°-60°) in **P**.

Remark:

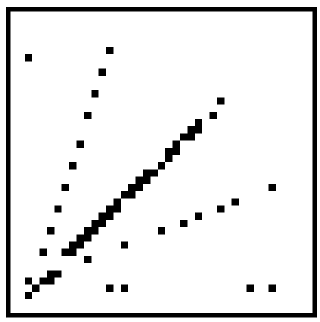
It is supposed that the difference between values of a grayscale line and those of the background is not less than 0.25 (see function ***b***). Analogously, the difference between values representing a grayscale line is supposed to be in the interval [-0.15, 0.15] (see function ***a***). The template can easily be tuned for other input assumptions by changing functions ***a*** and ***b***.
The functionality of this template is similar to that of the rotated version of the *GrayscaleDiagonalLineDetector* template.

II. Examples

Example 1 (simple): image name: line3060.bmp, image size: 41x42; template name: line3060.tem .



input



output

III. ACE4K implementation

Implementation method: optimization.

GrayscaleLineDetector _ACE4K: (Full-range model, ACE4K)

'Center –surround' template by continuous time cnn.

A =

-0.2	-0.20	0
-0.20	1.2	-0.2
0	-0.2	-0.2

B =

-0.2	-0.2	0
-0.2	1.2	-0.2
0	-0.2	-0.2

z_1 = -0.5 z_2 = 0

Line detection template

A =

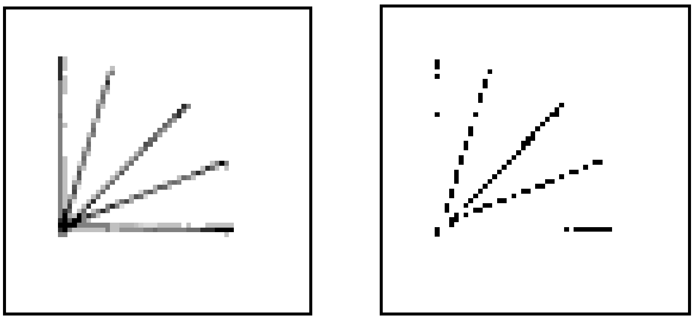
0	0	0
0	-1	0
0	0	0

B =

-0.75	0.25	0.25
-0.75	1	-0.25
0.25	-0.75	-0.75

z_1 = 0 z_2 = 0.2

Example (resolution: 64x64): image name: lin3060.bmp, amc name: gsline_ace4k.amc.



input output

Remarks:

- Before template running LLMs must be initialized with white.

LogicANDOperation: ***Logic AND and Set Intersection ∩ (Conjunction ∧) template***

<u>Old names</u>: LogicAND, LOGAND, AND

A =

0	0	0
0	2	0
0	0	0

B =

0	0	0
0	1	0
0	0	0

z = -1

I. Global Task

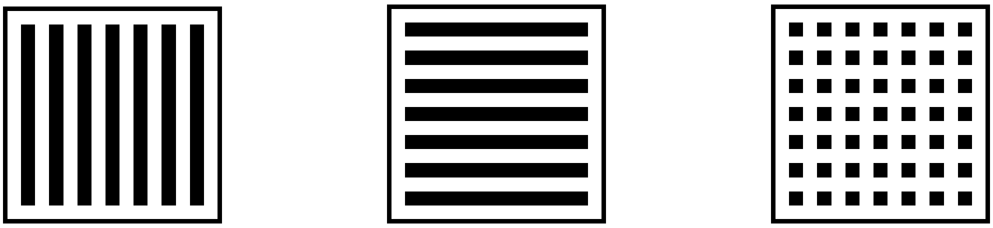
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

Input: $\mathbf{U(t) = P_1}$

Initial State: $\mathbf{X(0) = P_2}$

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = binary output of the logic operation "**AND**" between $\mathbf{P_1}$ and $\mathbf{P_2}$. In logic notation, $\mathbf{Y(\infty)=P_1 \wedge P_2}$, where $\wedge$ denotes the "conjunction" operator. In set-theoretic notation, $\mathbf{Y(\infty)=P_1 \cap P_2}$, where $\cap$ denotes the "intersection" operator.

II. Example: image names: logic01.bmp, logic02.bmp; image size: 44x44; template name: logand.tem .



input initial state output

III. ACE4K implementation

Implementation method: optimization.

LogAND_ACE4K: (Full-range model, ACE4K)

A =

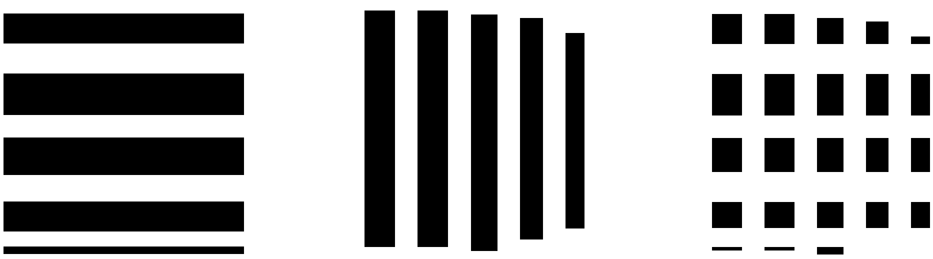
0	0	0
0	2	0
0	0	0

B =

0	0	0
0	1	0
0	0	0

z_1 = 0 z_2 = 0

<u>Example1</u> (resolution: 64x64): image name: striphor.bmp and stripver.bmp , template name: logand.tem.



input 1 input 2 output

Remarks:

- Because of the experienced interference between binary pictures put in a common template operation robust operation could achieved only by the use of fixed map which works very reliably.
- The test AMC code can be found here:

LogicOROperation: Logic OR and Set Union ∪ (Disjunction ∨) template

<u>Old names</u>: LogicOR, LOGOR, OR

A =

0	0	0
0	2	0
0	0	0

B =

0	0	0
0	1	0
0	0	0

z = 1

I. Global Task

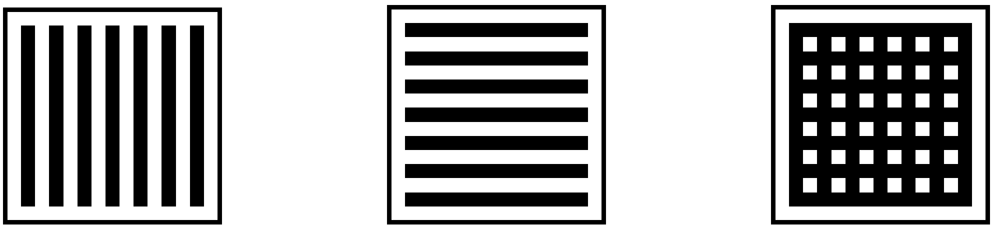
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

Input: $\mathbf{U(t) = P_1}$

Initial State: $\mathbf{X}(0) = \mathbf{P_2}$

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = binary output of the logic operation **OR** between $\mathbf{P_1}$ and $\mathbf{P_2}$. In logic notation, $\mathbf{Y(\infty)=P_1 \vee P_2}$, where $\vee$ denotes the "disjunction" operator. In set-theoretic notation, $\mathbf{Y(\infty)=P_1 \cup P_2}$ where $\cup$ denotes the "set union" operator.

II. Example: image names: logic01.bmp, logic02.bmp; image size: 44x44; template name: logor.tem .



input initial state output

III. ACE4K implementation

Implementation method: optimization.

LogOR_ACE4K: (Full-range model, ACE4K)

A =

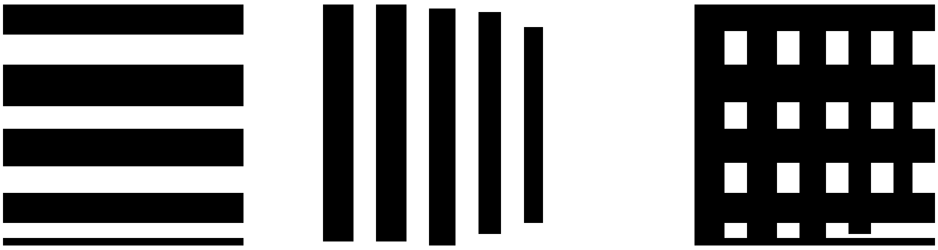
0	0	0
0	1	0
0	0	0

B =

0	0	0
0	2	0
0	0	0

z_1 = 0 z_2 = 0

Example 1(resolution: 64x64): image name: striphor.bmp and stripver.bmp , template name: logor.tem.



input 1 input 2 output

Remarks:

- Because of the experienced interference between binary pictures put in a common template operation robust operation could achieved only by the use of fixed map which works very reliably.

PatchMaker: ***Patch maker template* [22]**

<u>Old names</u>: *PATCHMAK (Chua-Yang model)*

A =

0	1	0
1	2	1
0	1	0

B =

0	0	0
0	1	0
0	0	0

z = 4.5

I. Global Task

Given: static binary image **P**

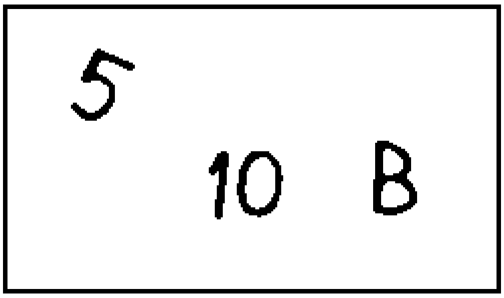
Input: **U(t) = P**

Initial State: **X**(0) **= P**

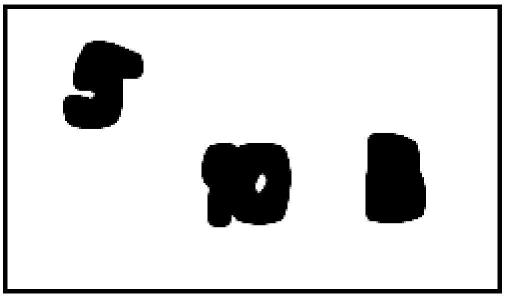
Boundary Conditions: Zero-flux boundary condition (duplicate)

Output: **Y(t)⇒Y(T)** = Binary image with enlarged objects of the input obtained after a certain time t = **T**. The size of the objects depends on time **T**. When **T** → ∞ all pixels will be driven to black.

II. Example: image name: patchmak.bmp; image size: 245x140; template name: patchmak.tem .



input



output

III. ACE4K implementation

Implementation method:

patchmaker_ace4k: (Full-range model, ACE4K)

A =

0	1	0
1	1	1
0	1	0

B =

0	0	0
0	1	0
0	0	0

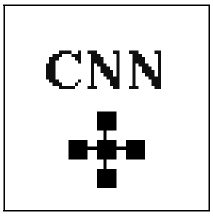
z = 4.5

Remarks:

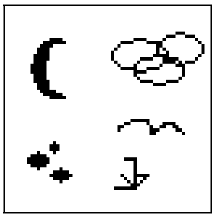
- Images should be fed into LLMs;

<u>Example 1</u> (resolution: 64x64); template: *patchmaker_ace4k.tem.*

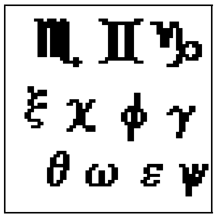
inputs



a,

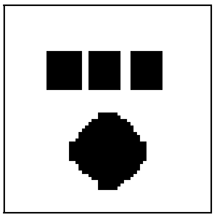


b,

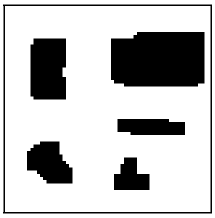


c,

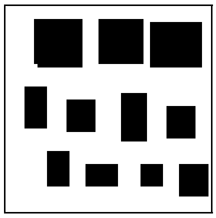
outputs



a,



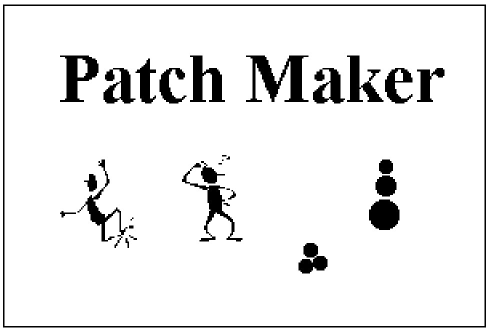
b,



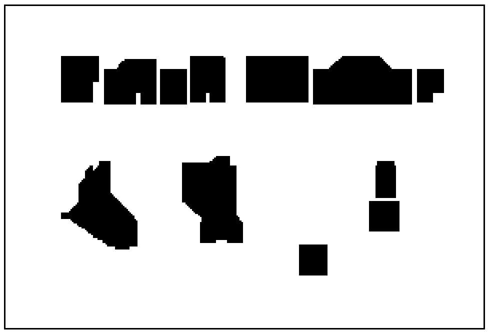
c,

Running time: 350 μs.

<u>Example 2</u> (resolution: 200x300); template: *patchmaker_ace4k.tem.*



input



output

Running time: 630 μs.

SmallObjectRemover: ***Deletes small objects*** **[22]**

<u>*Old names*</u>*: SMKILLER*

SmallObjectRemover (Chua-Yang model):

A =

1	1	1
1	2	1
1	1	1

B =

0	0	0
0	0	0
0	0	0

z = 0

I. Global Task

Given: static binary image **P**

Input: **U(t) = P**

Initial State: **X(0) = P**

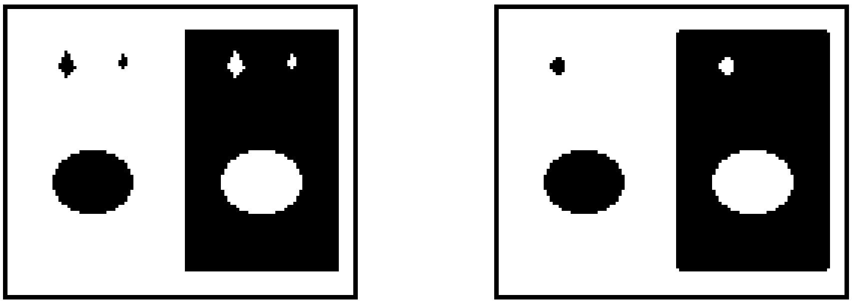
Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image representing **P** without small objects.

Remark:

This template drives dynamically white all those black pixels that have more than four white neighbors, and drives black all white pixels having more than four black neighbors.

II. Example: image name: smkiller.bmp; image size: 115x95; template name: smkiller.tem .



input output

III. ACE4K implementation

Implementation method: optimization.

SmallObjectRemover_ACE4K: *(Full-range model, ACE4K)*

A =

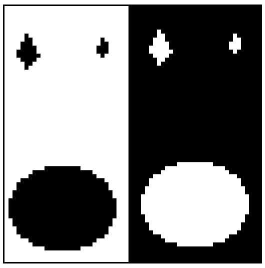
0.8	0.9	0.9
0.8	1	0.9
0.8	0.9	0.9

B =

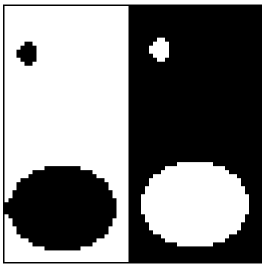
0	0	0
0	0	0
0	0	0

z_1 = -1.4 z_2 = 0

<u>Example 1</u> (resolution: 64x64): image name: smkiller64_i.bmp, template name: smkiller_ace4k.tem.



input



output

Remarks:

- LAM must be used for the template execution.

1.4. TEXTURE SEGMENTATION AND DETECTION

***3x3TextureSegmentation: Segmentation of four textures by a 3*3 template* [17]**

<u>Old names</u>: TX_RACC3

A =

0.86	0.94	3.75
2.11	-2.81	3.75
-1.33	-2.58	-1.02

B =

0.16	-1.56	1.25
-2.89	1.09	-3.2
4.06	4.69	3.75

z = 1.8

I. Global Task

Given: static grayscale image **P** representing four textures (raffia, aluminum mesh, 2 clothes) having the same flat grayscale histograms

Input: **U(t) = P**

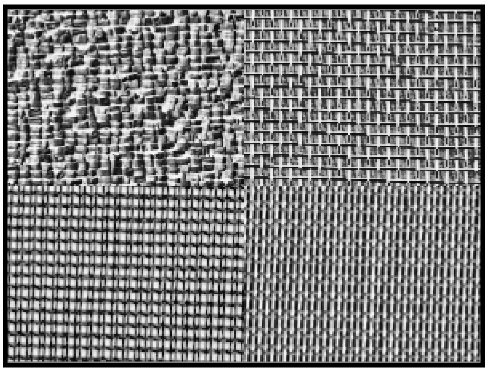
Initial State: **X(**0**) = P**

Boundary Conditions: Fixed type, $u_{ij} = 0$, $y_{ij} = 0$ for all virtual cells, denoted by [U]=[Y]=0

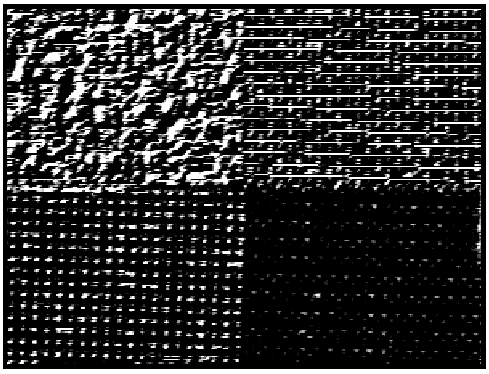
Output: **Y(t)⇒Y(T)** = Nearly binary image representing four patterns that differ in average gray-levels.

Remark:
This template is called "Texture Discrimination" in [44].

II. Example: image name: tx_racc.bmp, image size: 296x222; template name: tx_racc3.tem .



input



output

III. ACE4K implementation

Implementation method: recalculation of the template elements

Texture_ACE4K: (Full-range model, ACE4K)

<u>Example:</u> (resolution: 64x64): image name: text3x3.bmp, code name: text3x3.amc.

text3x3.tem: segmentation template

A =

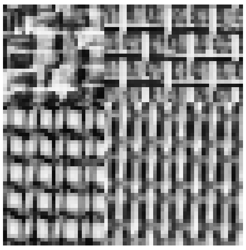
0.55	0.6	2.39
-1.65	-2.43	-1.65
2.39	2.39	-0.65

B =

0.1	-0.99	0.79
-1.84	0.69	-2.04
2.59	3	2.39

z_1 = 0 z_2 = 1.51

Using LAMs for the operation

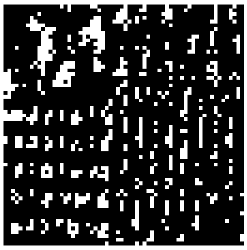


input

output

Using LLMs for the operation.

input



output

clean.tem: Used for initializing LAM before the usage.

A =

0	0	0
0	-1	0
0	0	0

B =

0	0	0
0	0	0
0	0	0

z_1 = 0 z_2 = 0

Remarks:

- The original picture read back after the operation seems quite agreement to the original picture so no significant distortion of the gray scale image were assumed.
- The original template's values were down-sized to the range of the chip (-3,3).
- The classification work reliably on binary images only. The gray scale variant turns to black very soon in case of loop running

GameofLife1Step: ***Simulates one step of the game of life*** **[11]**

Old names: LIFE_1

$\mathbf{A}_{11}$ =

0	0	0
0	1	0
0	0	0

$\mathbf{B}_{11}$ =

-1	-1	-1
-1	0	-1
-1	-1	-1

z = -1

$\mathbf{A}_{22}$ =

0	0	0
0	1	0
0	0	0

$\mathbf{B}_{21}$ =

-1	-1	-1
-1	-1	-1
-1	-1	-1

z = -4

I. Global Task

Given: static binary image **P**

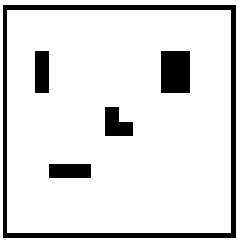
Inputs: $\mathbf{U_1(t) = P, U_2(t) = P}$

Initial States: $\mathbf{X_1}(0) = \mathbf{X_2}(0) = -1$

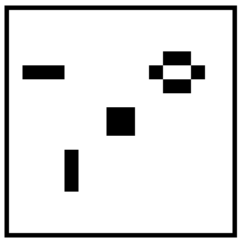
Boundary Conditions: Fixed type, u_{ij} = -1 for all virtual cells, denoted by [U]= -1

Outputs: $\mathbf{Y_1(t), Y_2(t) \Rightarrow Y_1(\infty), Y_2(\infty)}$ = Binary images representing partial results. The desired output is $\mathbf{Y_1(\infty)}$ **XOR** $\mathbf{Y_2(\infty)}$. For the simulation of the following steps of game of life this image should be fed to the input of the first layer.

II. Example: image name: life_1.bmp, image size: 16x16; template name: life_1.tem .



input



output

III. ACE4K implementation

Implementation method: optimization

LIFE_1_ACE4K: (Full-range model, ACE4K)

$\mathbf{A}_{11}$ =

0	0	0
0	2.5	0
0	0	0

$\mathbf{B}_{11}$ =

-2	-2	-2
-2	0	-2
-2	-2	-2

z = 1.4

$\mathbf{A}_{22}$ =

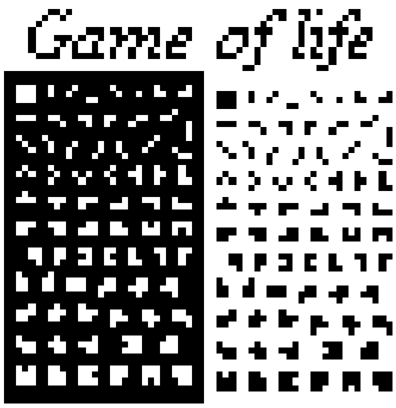
0	0	0
0	1	0
0	0	0

$\mathbf{B}_{21}$ =

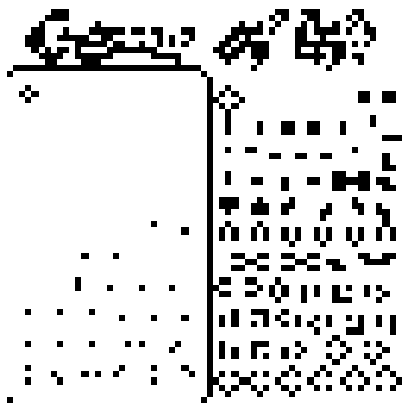
-1	-1	-1
-1	-1	-1
-1	-1	-1

z = -3

<u>Example 1</u> (resolution: 64x64): image name: life_i.bmp, template name: life_1_ace4k.tem.

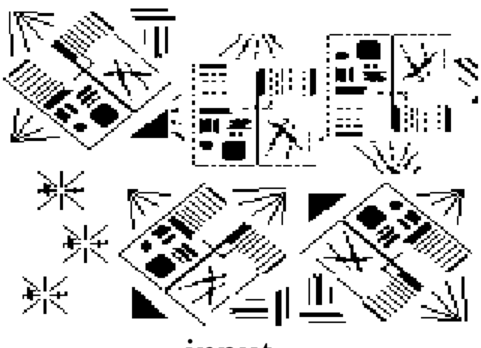


input

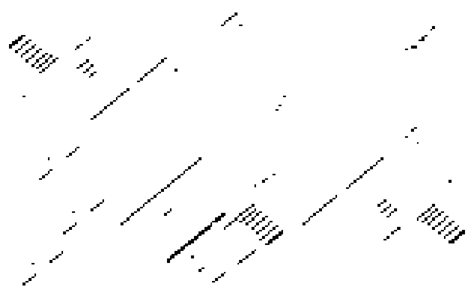


output

<u>Example 2</u> (resolution: 176x144): image name: life_tile.bmp, template name: life_1_ace4k.tem.



input



output

Remarks:

1. Boundary condition could be periodic, approximate running time is 100 tau.
2. hw.set.ref 0 60 -85 -110 -3 -55 51 113 84 ;nominal setting for template run

2. Subroutines

EDGE CONTROLLED DIFFUSION

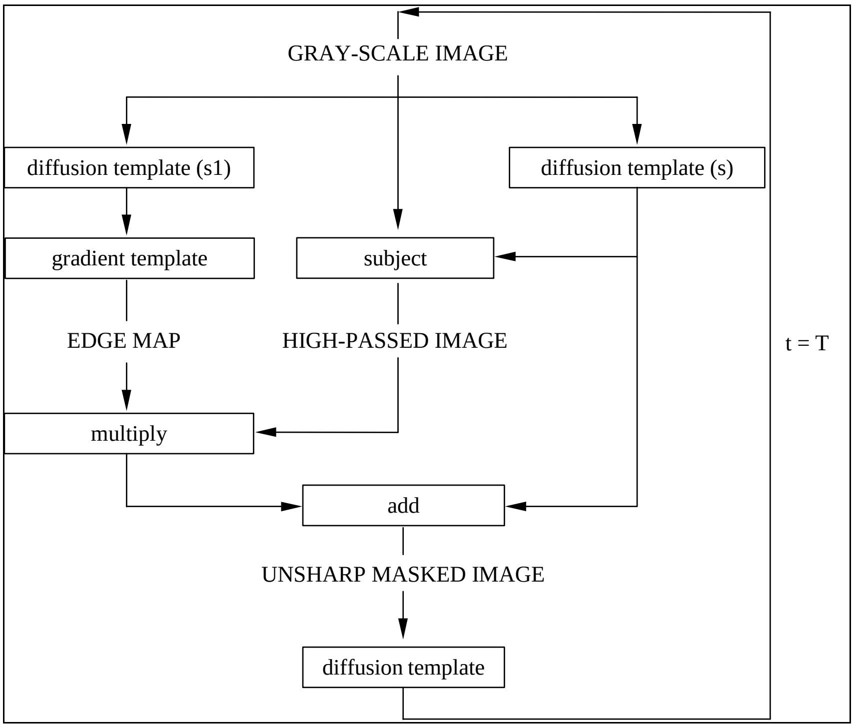
I. Description of the (original) gradient controlled diffusion algorithm

The edge controlled diffusion algorithm is a modification of the gradient controlled diffusion algorithm, which was included in the CNN Software algorithm.
The gradient controlled diffusion performs edge-enhancement during noise-elimination [17,25,30]. The equation used for filtering is as follows:

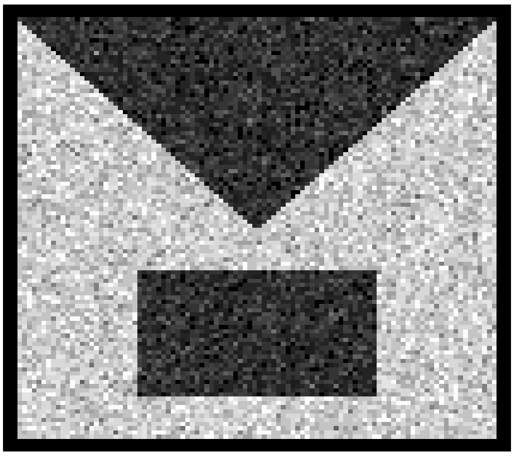
$$\frac{\partial I}{\partial t} = \Delta\left[I(x,y,t)\cdot\left(1-k\cdot\left|grad\left(G(s)*I(x,y,t)\right)\right|\right)\right]$$

Here $I(x,y,t)$ is the image changing in time, $G(s)$ is the Gaussian filter with aperture s, k is a constant value between 1 and 3. Both the Gaussian filtering and the Laplace operator (Δ) is done by the *HeatDiffusion* (diffusion) template with different diffusion coefficients. The *ThresholdedGradient* (gradient) template can also be found in this library. This equation can be used for noise filtering without decreasing the sharpness of edges.

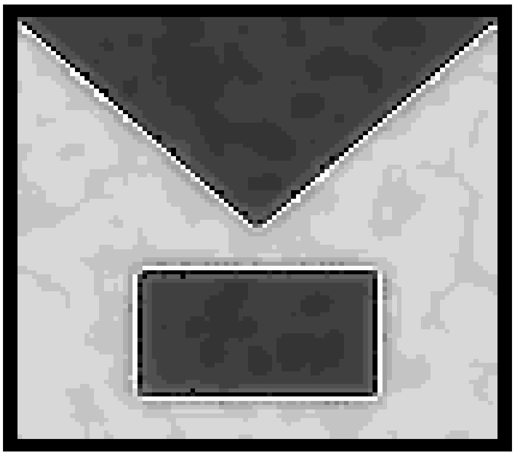
The flow-chart of the algorithm:



Example of the original algorithm: image name: laplace.bmp; image size: 100x100.



input



output

II. Description of the edge controlled diffusion (on-chip) algorithm

The algorithm contains a non-linear template for computing the gradient. This was not directly realizable on-chip. Therefore a collection of linear templates was chosen: orientation selective edge detection templates. Not all orientation were included in this test, thus the difference between the edge controlled and simple diffusion can be seen in the output picture. The remaining part of the algorithm was basically not modified. The diffusion was realized by a template, which was previously developed for diffusion.

Diffusion:

A =

0.35	0.35	0.35
0.35	-2.8	0.35
0.35	0.35	0.35

B =

0.2	0.2	0.2
0.2	0.1	0.2
0.2	0.2	0.2

z = 1.2

Edge1:

A =

0	0	0
0	1	0
0	0	0

B =

0.7	0.7	0
0.7	0	-0.7
0	-0.7	-0.7

z = -1.5

Edge2:

A =

0	0	0
0	1	0
0	0	0

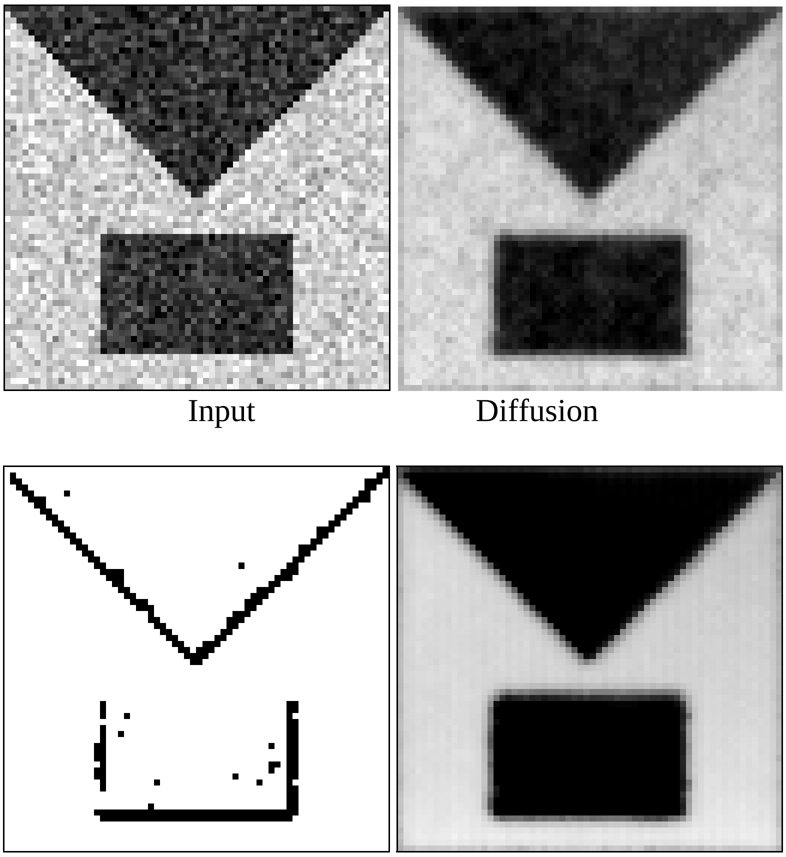
B =

-0.7	-0.7	0
-0.7	0	0.7
0	0.7	0.7

z = -1.5

Execution time:
Diffusion: 1
Edge: 50

Example on chip: ecd.bmp image size: 64x64



Input Diffusion

Selected edges Output

REFERENCES

[1] L. O. Chua and L. Yang, “Cellular neural networks: Theory and Applications”, *IEEE Transactions on Circuits and Systems*, Vol. 35, pp. 1257-1290, October 1988.

[2] L. O. Chua and L. Yang, “The CNN Paradigm”, *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, Vol. 40, pp. 147-156, March 1993.

[3] T. Roska and L. O. Chua, “The CNN Universal Machine: An Analogic Array Computer”, *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, Vol. 40, pp. 163-173, March 1993.

[4] The CNN Workstation Toolkit, Version 6.0, MTA SzTAKI, Budapest, 1994.

[5] P. L. Venetianer, A. Radványi, and T. Roska, "ACL (an Analogical CNN Language), Version 2.0, *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-3-1994, Budapest, 1994.

[6] T. Matsumoto, T. Yokohama, H. Suzuki, R. Furukawa, A. Oshimoto, T. Shimmi, Y. Matsushita, T. Seo and L. O. Chua, "Several Image Processing Examples by CNN", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-90)*, pp. 100-112, Budapest, 1990.

[7] T. Roska, T. Boros, A. Radványi, P. Thiran, L. O. Chua, "Detecting Moving and Standing Objects Using Cellular Neural Networks", *International Journal of Circuit Theory and Applications*, October 1992, and *Cellular Neural Networks*, edited by T. Roska and J. Vandewalle, 1993.

[8] T. Boros, K. Lotz, A. Radványi, and T. Roska, "Some Useful New Nonlinear and Delay-type Templates", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-1-1991, Budapest, 1991.

[9] S. Fukuda, T. Boros, and T. Roska, "A New Efficient Analysis of Thermographic Images by using Cellular Neural Networks", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-11-1991, Budapest, 1991.

[10] L. O. Chua, T. Roska, P. L. Venetianer, and Á. Zarándy, "Some Novel Capabilities of CNN: Game of Life and Examples of Multipath Algorithms", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-3-1992, Budapest, 1992.

[11] L. O. Chua, T. Roska, P. L. Venetianer, and Á. Zarándy, "Some Novel Capabilities of CNN: Game of Life and Examples of Multipath Algorithms", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-92)*, pp. 276-281, Munich, 1992.

[12] T. Roska, K. Lotz, J. Hámori, E. Lábos, and J. Takács, "The CNN Model in the Visual Pathway - Part I: The CNN-Retina and some Direction- and Length-selective Mechanisms", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-8-1991, Budapest, 1991.

[13] T. Roska, J. Hámori, E. Lábos, K. Lotz, L. Orzó, J. Takács, P. L. Venetianer, Z. Vidnyánszky, and Á. Zarándy, "The Use of CNN Models in the Subcortical Visual Pathway", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-16-1992, Budapest, 1992.

[14] P. Szolgay, I. Kispál, and T. Kozek, "An Experimental System for Optical Detection of Layout Errors of Printed Circuit Boards Using Learned CNN Templates", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-92)*, pp. 203-209, Munich, 1992.

[15] K. R. Crounse, T. Roska, and L. O. Chua, "Image halftoning with Cellular Neural Networks", *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, Vol. 40, No. 4, pp. 267-283, 1993.

[16] H. Harrer and J. A. Nossek, "Discrete-Time Cellular Neural Networks", TUM-LNS-TR-91-7, Technical University of Munich, Institute for Network Theory and Circuit Design, March 1991.

[17] T.Sziranyi and M.Csapodi, "Texture classification and Segmentation by Cellular Neural Network using Genetic Learning", *Computer Vision and Image Understanding*, Vol. 71, No. 3, pp. 255-270, September 1998.

[18] A. Schultz, I. Szatmári, Cs. Rekeczky, T. Roska, and L. O. Chua, “Bubble-debris classification via binary morphology and autowave metric on CNN”, *International Symposium on Nonlinear Theory and its Applications*, Hawaii, 1997

[19] P. L. Venetianer, F. Werblin, T. Roska, and L. O. Chua, "Analogic CNN Algorithms for some Image Compression and Restoration Tasks", *IEEE Transactions on Circuits and Systems*, Vol. 42, No.5, 1995.

[20] P. L. Venetianer, K. R. Crounse, P. Szolgay, T. Roska, and L. O. Chua, "Analog Combinatorics and Cellular Automata - Key Algorithms and Layout Design using CNN", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-94)*, pp. 249-256, Rome, 1994.

[21] H. Harrer, P. L. Venetianer, J. A. Nossek, T. Roska, and L. O. Chua, "Some Examples of Preprocessing Analog Images with Discrete-Time Cellular Neural Networks", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-94)*, pp. 201-206, Rome, 1994.

[22] Á. Zarándy, F. Werblin, T. Roska, and L. O. Chua, "Novel Types of Analogic CNN Algorithms for Recognizing Bank-notes", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-94)*, pp. 273-278, Rome, 1994.

[23] E. R. Kandel and J. H. Schwartz, "Principles of Neural Science", Elsevier, New York, 1985.

[24] A. Radványi, "Using Cellular Neural Network to 'See' Random-Dot Stereograms" in *Computer Analysis of Images and Patterns*, Lecture Notes in Computer Science 719, Springer Verlag, 1993.

[25] M. Csapodi, Diploma Thesis, Technical University of Budapest, 1994.

[26] K. Lotz, Z. Vidnyánszky, T. Roska, and J. Hámori, "The receptive field ATLAS for the visual pathway", Report NIT-4-1994, Neuromorphic Information Technology, Graduate Center, Budapest, 1994.

[27] G. Tóth, Diploma Thesis, Technical University of Budapest, 1994.

[28] T. Boros, K. Lotz, A. Radványi, and T.Roska, "Some useful, new, nonlinear and delay-type templates", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-1-1991, Budapest, 1991.

[29] G. Tóth, "Analogic CNN Algorithm for 3D Interpolation-Approximation", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-2-1995, Budapest, 1995.

[30] P. Perona and J. Malik, “Scale space and edge detection using anisotropic diffusion”, *Proceedings of the IEEE Computer Society Workshop on Computer Vision*, 1987.

[31] F. Werblin, T. Roska, and L. O. Chua, “The Analogic Cellular Neural Network as a Bionic Eye”, *International Journal of Circuit Theory and Applications*, Vol. 23, No. 6, pp. 541-569, 1995.

[32] R. M. Haralick, S. R. Sternberg, and X. Zhuang, “Image Analysis Using Mathematical Morphology”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 532-550, Vol. PAMI-9, No. 4, July 1987.

[33] L. O. Chua, T. Roska, T. Kozek, and Á. Zarándy, “The CNN Paradigm – A Short Tutorial”, *Cellular Neural Networks*, T. Roska and J. Vandewalle, editors, John Wiley & Sons, New York, 1993, pp. 1-14.

[34] Cs. Rekeczky, Y. Nishio, A. Ushida, and T. Roska, “CNN Based Adaptive Smoothing and Some Novel Types of Nonlinear Operators for Grey-Scale Image Processing”, in *proceedings of NOLTA’95*, Las Vegas, December 1995.

[35] T. Szirányi, “Robustness of Cellular Neural Networks in image deblurring and texture segmentation”, *International Journal of Circuit Theory and Applications*, Vol. 24, pp. 381-396, May 1996.

[36] Á. Zarándy, “The Art of CNN Template Design”, *International Journal of Circuit Theory and Applications*, Vol. 27, No. 1, pp. 5-23, 1999.

[37] M. Csapodi, J. Vandewalle, and T. Roska, "Applications of CNN-UM chips in multimedia authentication", ESAT-COSIC Report / TR 97-1, Department of Electrical Engineering, Katholieke Universiteit Leuven, 1997.

[38] L. Nemes, L. O. Chua, "**TemMaster** Template Design and Optimization Tool for Binary Input-Output CNNs, User's Guide", *Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA-SzTAKI)*, Budapest, 1997.

[39] P. Szolgay, K. Tömördi**,** "Optical detection of breaks and short circuits on the layouts of printed circuit boards using CNN", *Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-96)*, pp. 87-91, Seville, 1996.

[40] Hvilsted, S.; Ramanujam, P.S., "Side-chain liquid crystalline azobenzene polyesters with unique reversible optical storage properties". *Curr. Trends Pol. Sci.* (1996) v.1, pp. 53-63.

[41] S. Espejo, A. Rodriguez-Vázquez, R. A. Carmona, P. Földesy, Á. Zarándy, P. Szolgay, T. Szirányi, and T. Roska, "0.8µm CMOS Two Dimensional Programmable Mixed-Signal Focal-Plane Array Processor with On-Chip Binary Imaging and Instruction Storage", *IEEE Journal on Solid State Circuits*, Vol. 32., No. 7., pp. 1013-1026,. July 1997.

[42] G. Liñán, S. Espejo, R. Domínguez-Castro, E. Roca, and A. Rodriguez-Vázquez, "CNNUC3: A Mixed-Signal 64x64 CNN Universal Chip", *Proceedings of the International Conference on Microelectronics for Neural, Fuzzy and Bio-inspired Systems (MicroNeuro'99)*, pp. 61-68, Granada, Spain, 1999.

[43] S. Ando, "Consistent Gradient Operations", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22., No. 3., pp. 252-265,. March 2000.

[44] L. O. Chua, "CNN: a paradigm for complexity", *World Scientific Series On Nonlinear Science*, Series A, Vol. 31, 1998.

[45] L. Nemes, L.O. Chua, and T. Roska, "Implementation of Arbitrary Boolean Functions on the CNN Universal Machine", *International Journal of Circuit Theory and Applications - Special Issue: Theory, Design and Applications of Cellular Neural Networks: Part I: Theory*, (CTA Special Issue - I), Vol. 26. No. 6, pp. 593-610, 1998.

[46] I. Szatmári, Cs. Rekeczky, and T. Roska, "A Nonlinear Wave Metric and its CNN Implementation for Object Classification", *Journal of VLSI Signal Processing, Special Issue: Spatiotemporal Signal Signal Processing with Analogic CNN Visual Microprocessors*, Vol.23, No.2/3, pp. 437-448, Kluwer, 1999.

[47] I. Szatmári, "The implementation of a Nonlinear Wave Metric for Image Analysis and Classification on the 64x64 I/O CNN-UM Chip", CNNA 2000, 6th *IEEE International Workshop on Cellular Neural Networks and their Applications*, May 23-25, 2000, University of Catania, Italy.

[48] I. Szatmári, A. Schultz, Cs. Rekeczky, T. Roska, and L. O. Chua, "Bubble-Debris Classification via Binary Morphology and Autowave Metric on CNN", *IEEE Trans. on Neural Networks*, in print.

[49] P. Földesy, L. Kék, T. Roska, Á. Zarándy, and G. Bártfai, "Fault Tolerant CNN Template Design and Optimization Based on Chip Measurements", *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'98)*, pp. 404-409, London, 1998.

[50] P. Földesy, L. Kék, Á. Zarándy, T. Roska, and G. Bártfai, "Fault Tolerant Design of Analogic CNN Templates and Algorithms – Part I: The Binary Output Case", *IEEE Transactions on Circuits and Systems* special issue on *Bio-Inspired Processors and Cellular Neural Networks for Vision,* Vol. 46, No. 2, pp. 312-322, February 1999.

[51] Á. Zarándy, T. Roska, P. Szolgay, S. Zöld, P. Földesy and I. Petrás, "CNN Chip Prototyping and Development Systems", *European Conference on Circuit Theory and Design - ECCTD'99,* Design Automation Day proceedings, (ECCTD'99-DAD), Stresa, Italy, 1999.

[52] I. Petrás, T. Roska, "Application of Direction Constrained and Bipolar Waves for Pattern Recognition", *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA2000),* in print

INDEX