CELLULAR SENSORY WAVE COMPUTERS LABORATORY
COMPUTER AND AUTOMATION RESEARCH INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES

CELLULAR WAVE COMPUTING LIBRARY

(TEMPLATES, ALGORITHMS, AND PROGRAMS)

VERSION 2.1

CSW-1-2007

BUDAPEST, 2007

EDITED BY L. KÉK, K. KARACS , AND T. ROSKA

BUDAPEST, HUNGARY

TABLE OF CONTENTS

INTRODUCTION

We are witnessing a proliferation of cellular topographic processor arrays: the Blue-Gene of IBM (the biggest supercomputer), the "heart" of the Playstation 3, the CELL Multiprocessor (Sony-IBM-Toshiba), as well as the FPGA chips and the new 80-tile chip of Intel and the 90 processor array of IBM. In addition, the Eye-RIS system of AnaFocus, and the Xenon chip of Eutecus (in collaboration with MTA SZTAKI and the Pazmany University).

Indeed, a new scenario is emerging in Cellular Wave Computing[*].

This new edition of the old CNN SOFTWARE LIBRARY contains some new elements and some editing. The style of the description follows the first textbook[+].

Algorithm or software development for the CNN Universal Machine needs similar tools as in the case of digital microprocessors. However, the software elements are completely different. Our software library contains instructions, subroutines, and modules/programs. The most important instruction is defined by a CNN template.

As for the templates/instructions, they are classified according to their functions: basic image processing, mathematical morphology, spatial logic, texture segmentation and detection, motion, color, depth, optimization, game of life and combinatorics. Some useful template sequences are defined as subroutines, and a few complete programs are shown as well. These software elements can be called in analogic programs described by the Alpha language. Some templates for neuromorphic modeling and solving partial differential equations (PDE) are also included.

Unless otherwise noted, the normalized first order CNN equation with linear delay-less templates is

$$\dot{x}_{ij} = -x_{ij} + z + \sum A_{ij;\,kl*}\, y_{kl} + \sum B_{ij;\,kl*}\, u_{kl} + \sum C_{ij;\,kl*}\, x_{kl} + \sum \hat{D}_{ij;\,kl}(u_{kl}, x_{kl}, y_{kl})$$

Time is scaled in $\tau = \tau_{CNN}$, the time constant of the first order CNN cell. As a default, $\tau_{CNN}=1$.

This library is an intermediate result of a continuous development. It contains results published by dozens of researchers all over the world.

The library is not complete[†]. New templates and subroutines can be added. We encourage CNN designers all over the world to send their templates or subroutines to be included in this library, with proper reference to the original publication. Initiating this action, you can e-mail to roska@sztaki.hu.

---

[*] T. Roska, Cellular wave computers for nano-tera-scale technology − beyond Boolean, spatial-temporal logic in million processor devices, *Electronics Letters,* Vol. 43., No.8., April 2007.

[+] L. O. Chua and T. Roska, Cellular Neural Networks: Foundations and applications, Cambridge University Press, 2002 (for the course EE129 at U.C. Berkeley, and a doctoral course at Pazmany University, Budapest).

[†] M. Itoh and L. O. Chua, Advanced Image Processing Cellular Neural Networks, *International Journal of Bifurcation and Chaos*, Vol. 17., No. 4, 2007.

UMF ALGORITHM DESCRIPTION
Version 1.3

1. Elementary instruction

An elementary CNN wave instruction is defined on a 2 dimensional grid (ij) by the constitutive standard CNN dynamics. The simplest PDDE (partial differential difference equation) defining it with a cloning template {A,B, z} is:

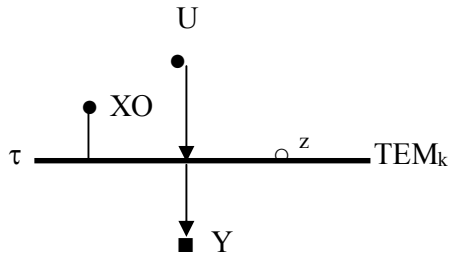
$$\tau\, dx_{ij}/dt = -x_{ij} + \sum A_{kl}\, y_{kl} + \sum B_{kl}\, u_{kl} + z_{ij}$$

(kl is in the r neighborhood of ij)

$$y_{ij} = f(x_{ij})$$

$\{x_{ij}(0)\}=X0$, $\{u_{ij}\}= U$,
$\{y_{ij}\}= Y$, $\{z_{ij}\}= Z$,
if z_{ij} is the same for all ij then $z_{ij} =z$, $t \in T$

The CNN Software Library contains many templates {A, B, z} implemen–ting simple and exotic waves, with standard and more complex templates and first order as well as higher order (complex) cells for various image processing tasks.

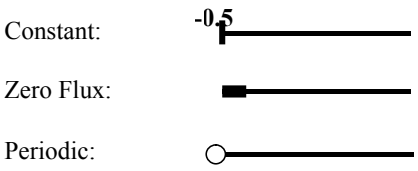


2. Signals, variables

■ logic array
■■ vector of logic arrays
□ logic value
● analog array
●● vector of analog arrays
○ analog value

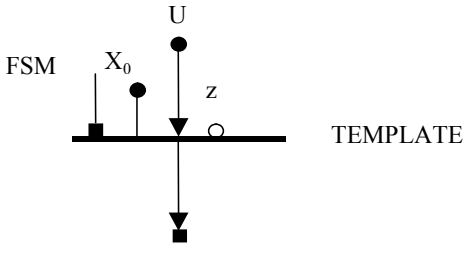
3. Boundary conditions

Left side: Input boundary condition
Right side: Output boundary condition

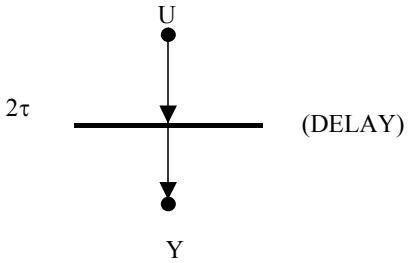


Boundary conditions are optional, if not given, it means "don't care"

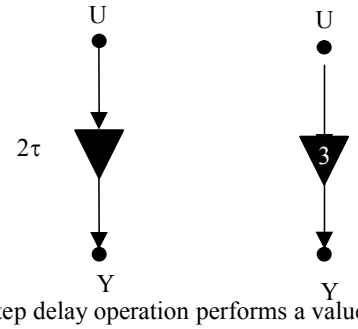
4. Fixed State Map



5. Continuous delay

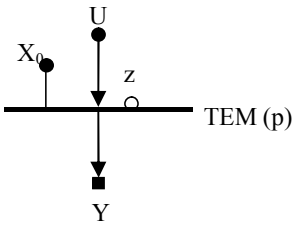


6. Step delay



A step delay operation performs a value delay. The time can be specified either in τ or in GAPU instruction steps. If neither is given, delay time defaults to a single GAPU instruction step.

7. Parametric templates



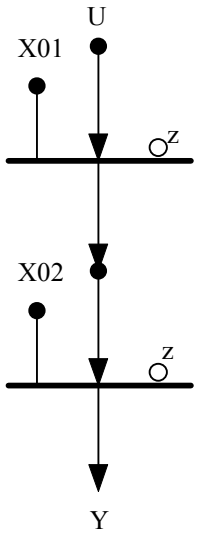
Example: Multiplication with constant

MULT(p)

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad , \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & p & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

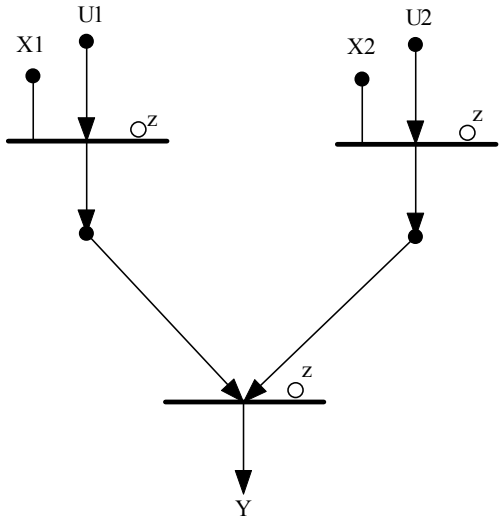
8. Algorithmic structures

Cascade

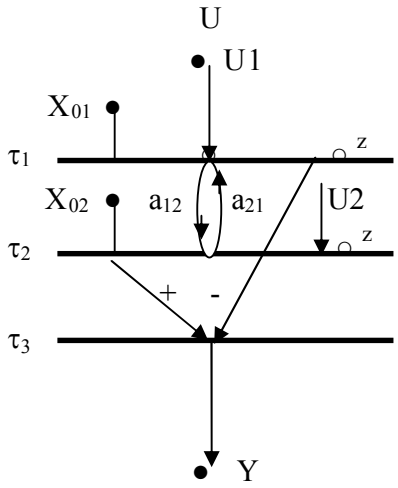


Parallel

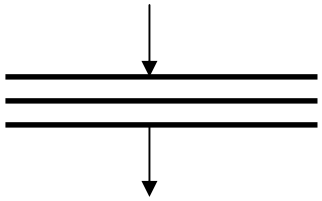
A typical parallel structure with two parallel flows is shown below, by combining them in the final



9. Three layer complex cell



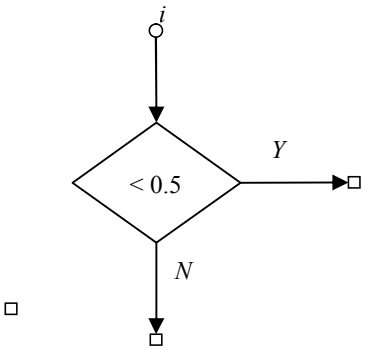
We can use a compact symbol below for the compact cell and use it in cascade, parallel and combined cascade-parallel structures.



10. Decisions

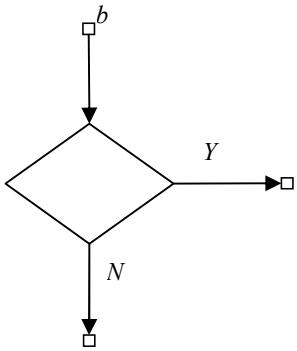
On global analog parameter

Is the value of variable *i* less then 0.5?



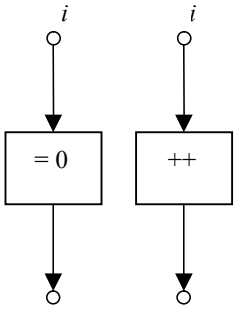
On global Logic parameter set, including global Fluctuation

Does the logic value of variable *b* refers to white?

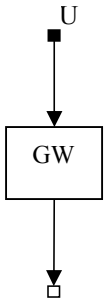


11. Operators

C-like imperative operators



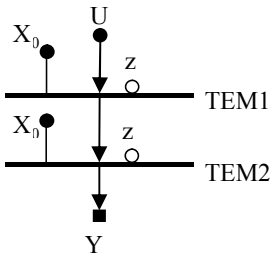
Global White operator



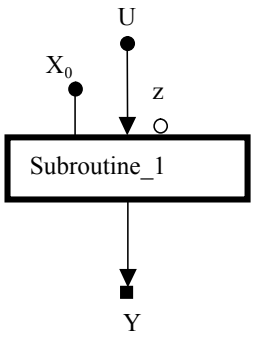
12. Subroutines

Definition

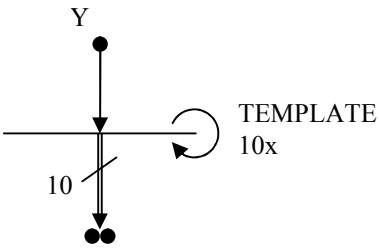
Subroutine_1



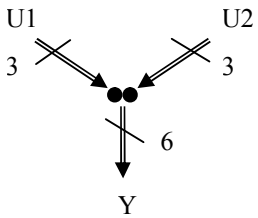
Usage



13. Iterations and vectors of arrays

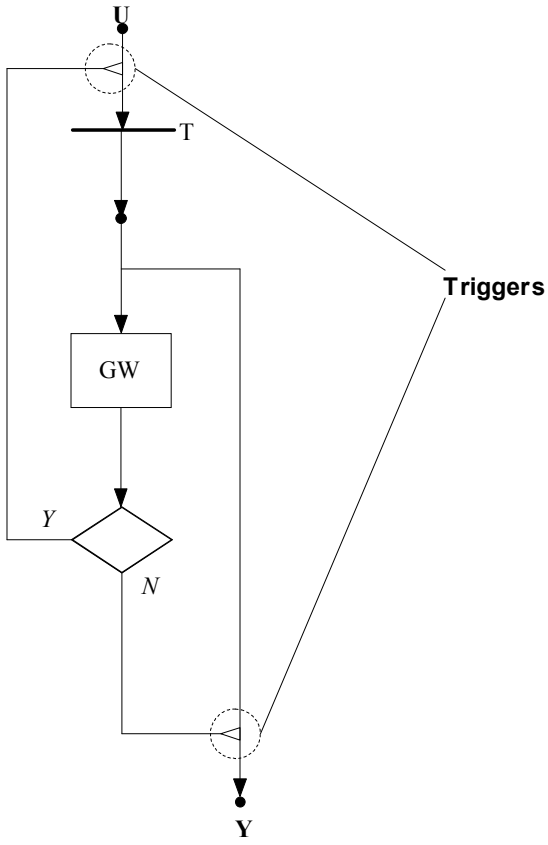


Merging arrays



14. Triggers, Cycles

A trigger makes a dataflow to continue



Depending on the output of the decision (that can be either yes or no) the proper dataflow continues.

```
flow=U;
repeat
       flow=T(flow);
until GW==1;
Y=flow;
```

TEMPLATE ROBUSTNESS

Here we will give the definition of template robustness for the case of uncoupled binary input/output templates.
It is known that the set of uncoupled binary input/output templates is isomorphic to the set of linearly separable Boolean functions of 9 variables [44]. Such functions can be described by 9-dimensional hyper-cubes [45]. If the function is linearly separable, a hyper-plane exists which separates the set of -1s from the set of 1s.
Definition: The robustness of the template T, denoted by ρ, is defined as the minimal distance of the hyper-plane, which separates the set of -1s from the set of 1s, and from the vertices of the hyper-cube (see Figure 1a for an illustration in 2 dimensions).

The robustness of T can be increased by choosing the optimal template T^{opt}, for which the minimal distance of the separating hyper-plane from the vertices is maximal [45] (see Figure 1b).

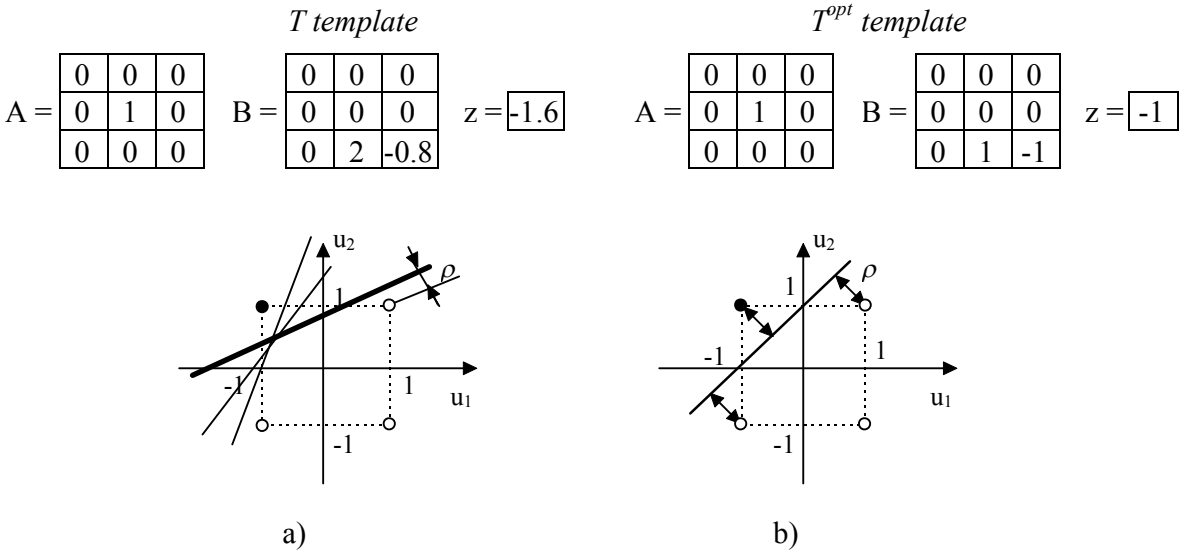


Figure 1. These diagrams illustrate the separation of vertices for the 2 dimensional logic function $F(u_1,u_2)=\overline{u}_1u_2$. Logic TRUE and FALSE are represented by filled and empty circles, respectively. The concept of robustness ρ is also illustrated. Figure a) shows a few possible separating lines. The thick line corresponds to the template T with robustness $\rho = 0.18$. Figure b) depicts the optimal separation line corresponding to the template T^{opt}; its robustness is $\rho = 0.71$.

Chapter 1. Templates/Instructions

1.1. BASIC IMAGE PROCESSING

GradientIntensityEstimation: Estimation of the gradient intensity in a local neighborhood

Old names: AVERGRAD

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \\ \boldsymbol{b} & 0 & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \qquad z = \boxed{0}$$

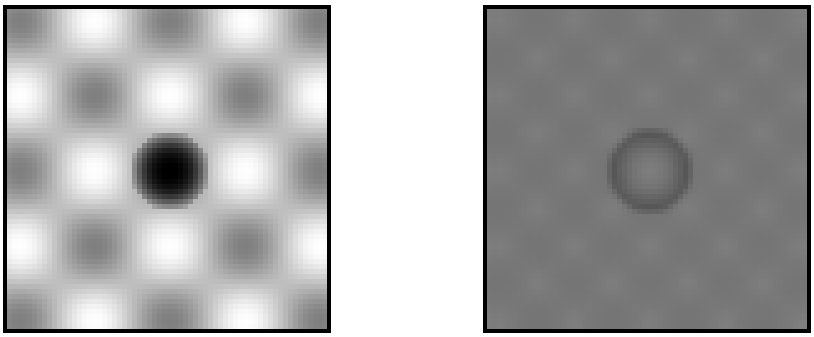
where $\boldsymbol{b} = |v_{u_{ij}} - v_{u_{kl}}| / 8$.

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**.

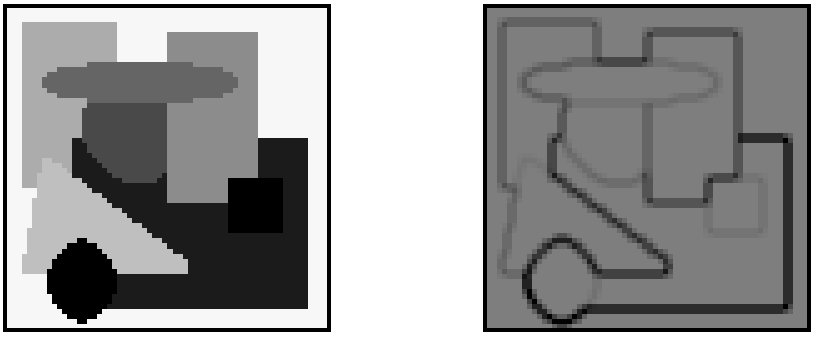
II. Examples

Example 1: image name: avergra1.bmp, image size: 64x64; template name: avergrad.tem .



input output

Example 2: image name: avergra2.bmp, image size: 64x64; template name: avergrad.tem.



input output

Smoothing: ***Smoothing with binary output*** **[1]**

Old names: AVERTRSH, AVERAGE

$\mathbf{A} =$ | 0 | 1 | 0 |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 1 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z =$ | 0 |
|---|

I. Global Task

Given:	static grayscale 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 black (white) pixels correspond to the locations in **P** where the average of pixel intensities over the r=1 feedback convolution window is positive (negative).

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

input

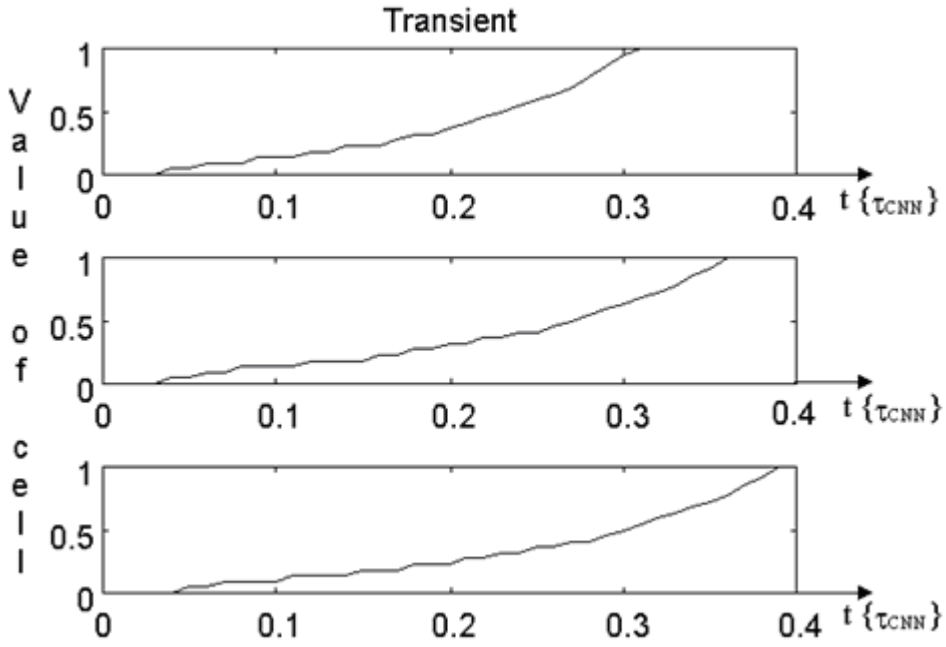
output

The transient of the cell C(38,10) has been examined in 3 cases:
1. applying the original **A** template shown before;
2. applying the following $\mathbf{A}_1$ template (template name: avertrs1.tem);
3. applying the following $\mathbf{A}_2$ template(template name: avertrs2.tem).

$\mathbf{A}_1 =$ | 0 | 1.2 | 0 |
|---|---|---|
| 1.2 | 1.8 | 1.2 |
| 0 | 1.2 | 0 |

$\mathbf{A}_2 =$ | 0 | 0.9 | 0 |
|---|---|---|
| 0.9 | 1.8 | 0.9 |
| 0 | 0.9 | 0 |

The transients of the examined cell are presented in the following figure corresponding to the templates **A**, $\mathbf{A}_1$ and $\mathbf{A}_2$.


Transient
Value of cell
1
0.5
0
0
0.1
0.2
0.3
0.4
t {τCNN}

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

Old names: CCD_DIAG

A = | 1 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | -1 |

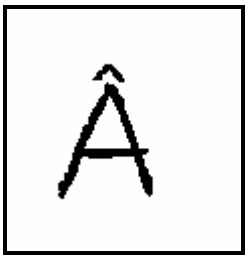
B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = | 0 |

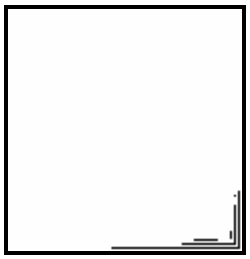
I. Global Task

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

HorizontalHoleDetection: ***Detects the number of horizontal holes from each horizontal row*** **[6]**

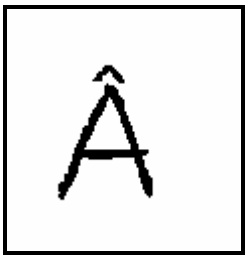
Old names: HorizontalCCD ,CCD_HOR

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

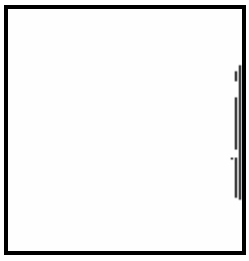
I. Global Task

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 horizontal holes in each horizontal row of image **P**.

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



input



output

VerticalHoleDetection: ***Detects the number of vertical holes from each vertical column*** **[6]**

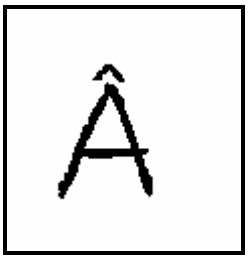
Old names: VerticalCCD, CCD_VERT

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

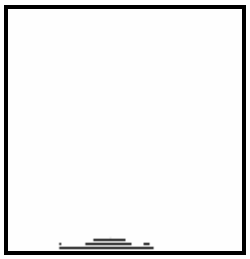
I. Global Task

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 vertical holes in each vertical column of image **P**.

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



input



output

MaskedCCD: ***Masked connected component detector*** **[24]**

Old names: CCDMASK

Left-to-right

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{-3}$$

I. Global Task

Given: static binary images $\mathbf{P_1}$ (mask) and $\mathbf{P_2}$

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

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

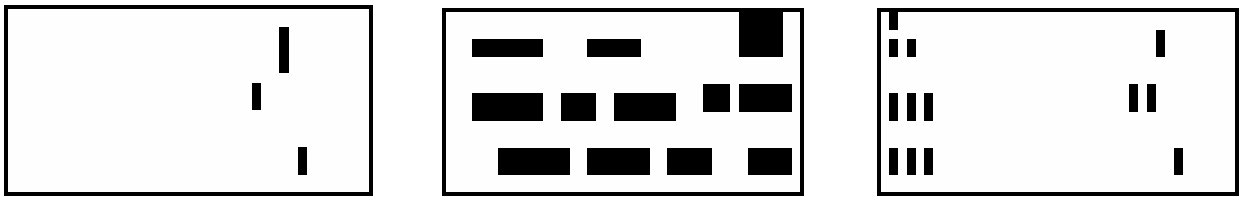
Boundary Conditions: Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image that is the result of CCD type shifting $\mathbf{P_2}$ from left to right. Shifting is controlled by the mask $\mathbf{P_1}$.

Remark:

This is a CCD operation, but the black trains stop in front of a black wall of the mask ($\mathbf{P_1}$) instead of the boundary of the image. The direction of shift is from the positive to the negative non-zero off-center feedback template entry. By rotating **A** the template can be sensitized to other directions as well.

II. Example: Right-to-left shifting. Image names: ccdmsk1.bmp, ccdmsk2.bmp; image size: 40x20; template name: ccdmaskr.tem .



input initial state output

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

Old names: CENTER

$\mathbf{A}_1 =$ | 1 | 0 | 0 |
|---|---|---|
| 1 | 4 | -1 |
| 1 | 0 | 0 |

$\mathbf{B}_1 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z_1 =$ | -1 |
|---|

$\mathbf{A}_2 =$ | 1 | 1 | 1 |
|---|---|---|
| 1 | 6 | 0 |
| 1 | 0 | -1 |

$\mathbf{B}_2 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z_2 =$ | -1 |
|---|

$\mathbf{A}_3 =$ | 1 | 1 | 1 |
|---|---|---|
| 0 | 4 | 0 |
| 0 | -1 | 0 |

$\mathbf{B}_3 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z_3 =$ | -1 |
|---|

$\mathbf{A}_4 =$ | 1 | 1 | 1 |
|---|---|---|
| 0 | 6 | 1 |
| -1 | 0 | 1 |

$\mathbf{B}_4 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z_4 =$ | -1 |
|---|

. . .

$\mathbf{A}_8 =$ | 1 | 0 | -1 |
|---|---|---|
| 1 | 6 | 0 |
| 1 | 1 | 1 |

$\mathbf{B}_8 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z_8 =$ | -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:

$\mathbf{A}_1 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_1 =$ | 1 | 0 | 0 |
|---|---|---|
| 1 | 4 | -1 |
| 1 | 0 | 0 |

$z_1 =$ -1

CENTER2:

$\mathbf{A}_2 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_2 =$ | 1 | 1 | 1 |
|---|---|---|
| 1 | 6 | 0 |
| 1 | 0 | -1 |

$z_2 =$ -1

CENTER3:

$\mathbf{A}_3 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_3 =$ | 1 | 1 | 1 |
|---|---|---|
| 0 | 4 | 0 |
| 0 | -1 | 0 |

$z_3 =$ -1

CENTER4:

$\mathbf{A}_4 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_4 =$ | 1 | 1 | 1 |
|---|---|---|
| 0 | 6 | 1 |
| -1 | 0 | 1 |

$z_4 =$ -1

. . .

CENTER8:

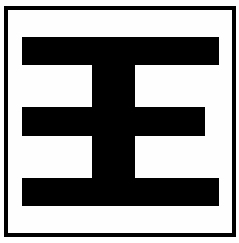
$\mathbf{A}_8 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_8 =$ | 1 | 0 | -1 |
|---|---|---|
| 1 | 6 | 0 |
| 1 | 1 | 1 |

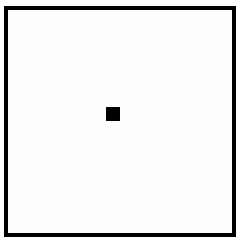
$z_8 =$ -1

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

ConcentricContourDetector: ***Concentric contour detection (DTCNN)*** **[16]**

Old names: CONCCONT

A = | 0 | -1 | 0 |
|---|---|---|
| -1 | 3.5 | -1 |
| 0 | -1 | 0 |

B = | 0 | 0 | 0 |
|---|---|---|
| 0 | 4 | 0 |
| 0 | 0 | 0 |

z = -4

I. Global Task

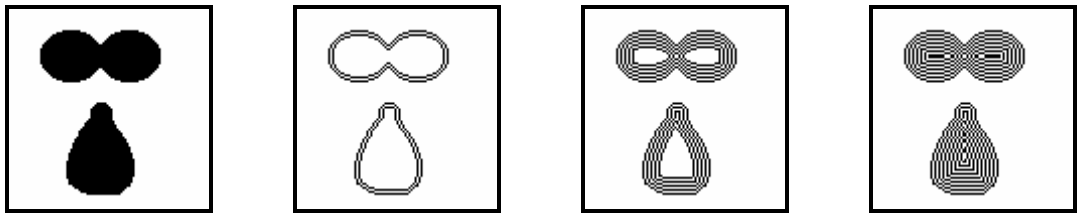
Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(0) = P**
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = Binary image representing the concentric black and white rings obtained from **P**.

Examples

Example 1: image name: conc1.bmp, image size: 16x16; template name: concont.tem .

input output

Example 2: image name: conc2.bmp, image size: 100x100; template name: concont.tem .



input output, 3. step output, 9. step output, t = ∞

GlobalConnectivityDetection: ***Deletes marked objects*** **[36]**

Old names: Connectivity

$\mathbf{A} =$ | 0 | 0.5 | 0 |
|---|---|---|
| 0.5 | 3 | 0.5 |
| 0 | 0.5 | 0 |

$\mathbf{B} =$ | 0 | -0.5 | 0 |
|---|---|---|
| -0.5 | 3 | -0.5 |
| 0 | -0.5 | 0 |

$z =$ | -4.5 |
|---|

I. Global Task

Given: two static binary images $\mathbf{P_1}$ (mask) and $\mathbf{P_2}$ (marker). The mask contains some black objects against the white background. The marker contains the same objects, except for some objects being marked. An object is considered to be marked, if some of its black pixels are changed into white.

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

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

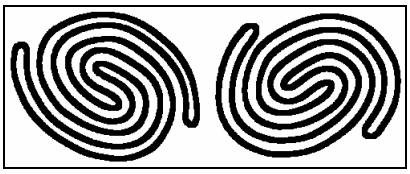
Boundary Conditions: Fixed type, $u_{ij} = -1$, $y_{ij} = -1$ for all virtual cells, denoted by [U]=[Y]=[-1]

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image containing the unmarked objects only.

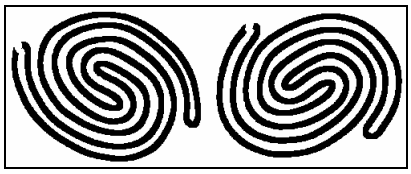
Remark:

The template determines whether a given geometric pattern is "globally" connected in one contiguous piece, or is it composed of two or more disconnected components.

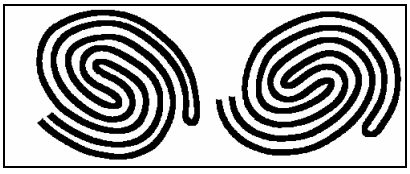
II. Example: image names: connect1.bmp, connect2.bmp; image size: 500x200; template name: connecti.tem .



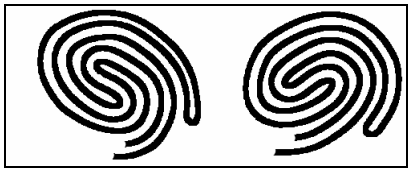
INPUT



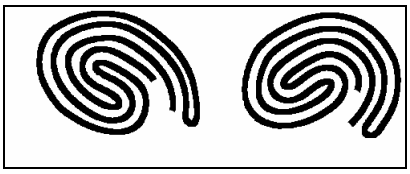
INITIAL STATE



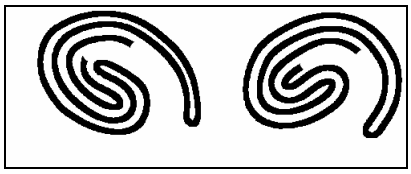
t=125τ



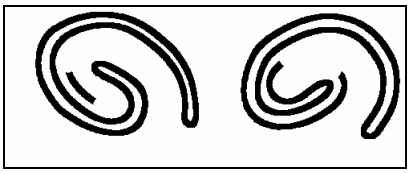
t=250τ



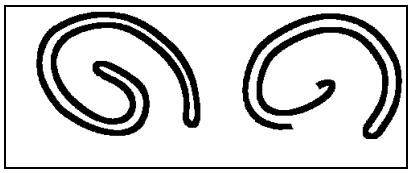
t=375τ



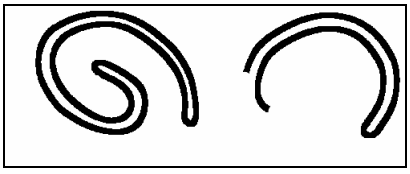
t=500τ



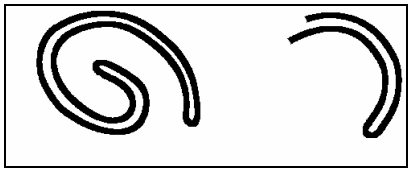
t=625τ



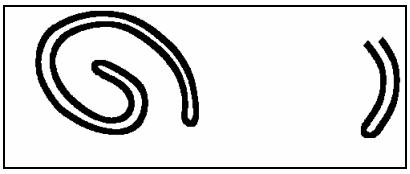
t=750τ



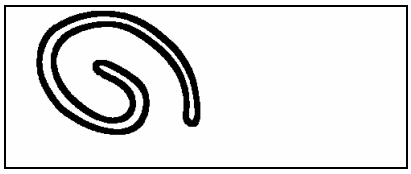
t=875τ



t=1000τ



t=1125τ



t=1250τ

***GlobalConncetivityDetection1: Detects the one-pixel thick closed curves and deletes the open curves from a binary image* [61]**

$\mathbf{A} =$ | 6 | 6 | 6 |
|---|---|---|
| 6 | 9 | 6 |
| 6 | 6 | 6 |

$\mathbf{B} =$ | -3 | -3 | -3 |
|---|---|---|
| -3 | 9 | -3 |
| -3 | -3 | -3 |

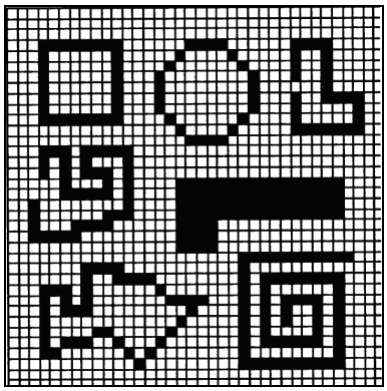
$z =$ -4.5

I. Global Task

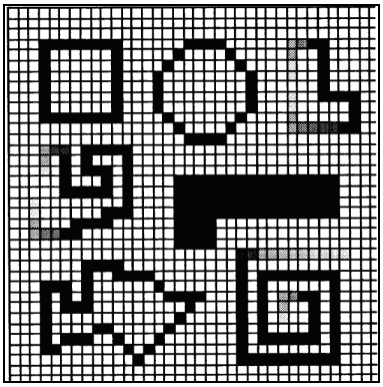
Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X**(0) **= P**
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = Binary image which contains all closed curves present in the initial image **P**

II. Example: image size: 36x36.

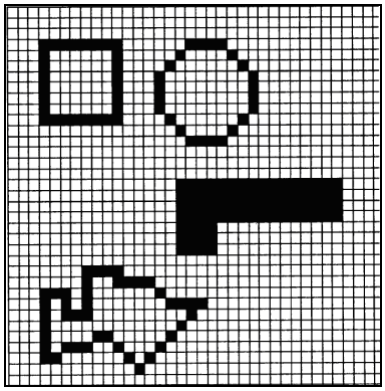
Remarks: *The binary image* **P** *containing closed and open curves (one-pixel thick) is applied both at the input and loaded as initial state. If one pixel is removed from a closed curve, it becomes an open curve and is deleted, as shown in the second image. The compact (solid) objects from the image are not modified.*



Input



Intermediate result



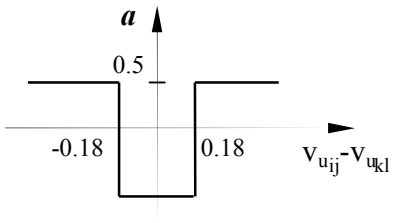
Output

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

Old names: ContourDetector, CONTOUR

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \boldsymbol{a} & \boldsymbol{a} & \boldsymbol{a} \\ \boldsymbol{a} & 0 & \boldsymbol{a} \\ \boldsymbol{a} & \boldsymbol{a} & \boldsymbol{a} \end{bmatrix} \qquad z = \boxed{0.7}$$

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

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

Old names: CornerDetector, CORNER

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

The transient of the cell C(6,3) has been examined in 3 cases:
1. applying the original *CornerDetector* template shown before;
2. applying the following CORNCH_1 template;
3. applying the following CORNCH_2 template.

CORNCH_1

A = | 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

B = | -1 | -1 | -1 |
| -1 | 3.9 | -1 |
| -1 | -1 | -1 |

z = -5

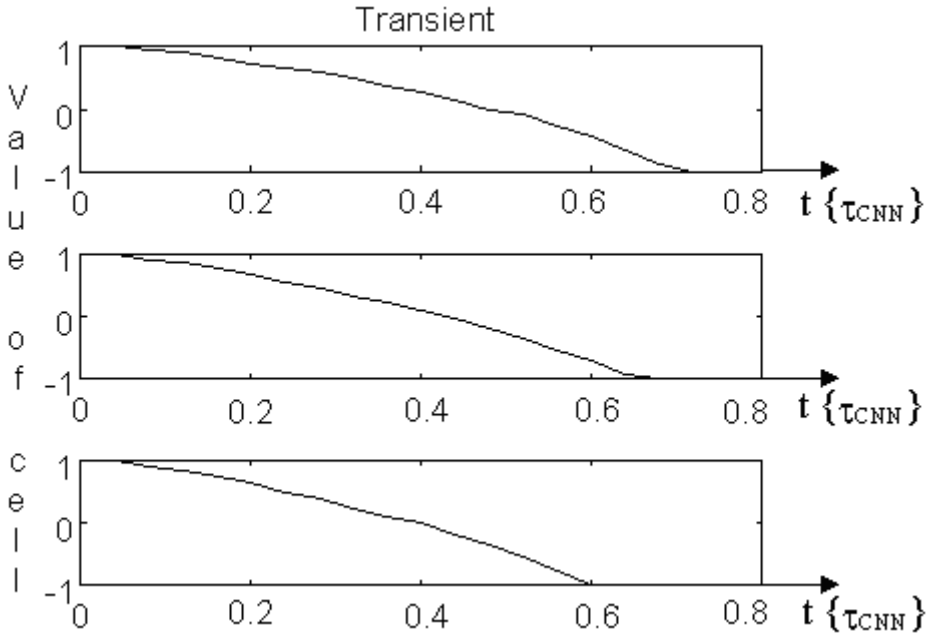
CORNCH_2

A = | 0 | 0 | 0 |
| 0 | 0.5 | 0 |
| 0 | 0 | 0 |

B = | -1 | -1 | -1 |
| -1 | 3.6 | -1 |
| -1 | -1 | -1 |

z = -5

The transients of the examined cell are presented in the following figure corresponding to the templates *CornerDetector*, CORNCH_1 and CORNCH_2.



DiagonalLineRemover: ***Deletes one pixel wide diagonal lines*** **[8]**

Old names: DELDIAG1

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | -1 | 0 | -1 |
|---|---|---|
| 0 | 1 | 0 |
| -1 | 0 | -1 |

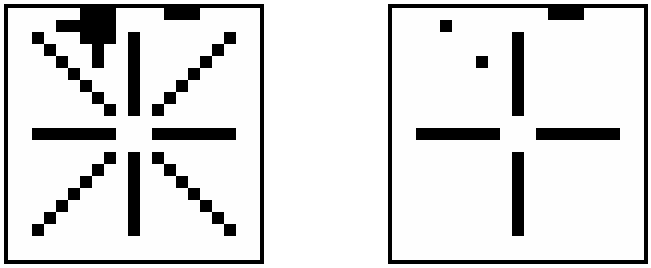
$z =$ | -4 |
|---|

I. Global Task

Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(0) =** 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 where black pixels have no black neighbors in diagonal directions in **P**.
Template robustness:	ρ = 0.45 .

Remark:
The template may be used for deleting one pixel wide diagonal lines.

II. Example: image name: deldiag1.bmp, image size: 21x21; template name: deldiag1.tem .



input output

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

Old names: DELVERT1

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \qquad z = \boxed{-2}$$

I. Global Task

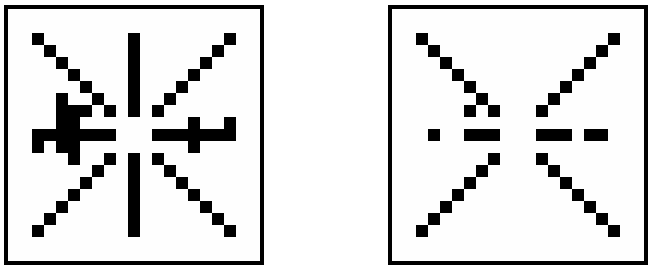
Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)
Boundary Conditions:	Fixed type, u_{ij} = -1, for all virtual cells, denoted by [U] = -1
Output:	**Y(t)⇒Y(∞)** = Binary image representing **P** without vertical lines. Those 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

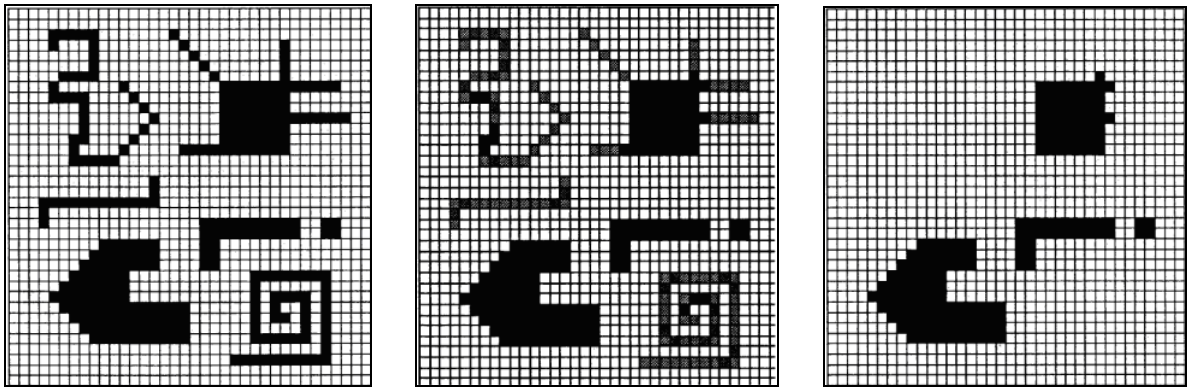
ThinLineRemover: ***Removes thin (one-pixel thick) lines from a binary image***

$$\mathbf{A} = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 8 & 2 \\ 2 & 2 & 2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{-2}$$

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 containing compact black objects (without any thin lines) against a white background

II. Example: image size: 36x36.



Initial state Intermediate state Output

ApproxDiagonalLineDetector: ***Detects approximately diagonal lines***

Old names: DIAG

A =

0	0	0	0	0
0	0	0	0	0
0	0	2	0	0
0	0	0	0	0
0	0	0	0	0

B =

-1	-1	-1	0.5	1
-1	-1	1	1	0.5
-1	1	5	1	-1
0.5	1	1	-1	-1
1	0.5	-1	-1	-1

z = -13

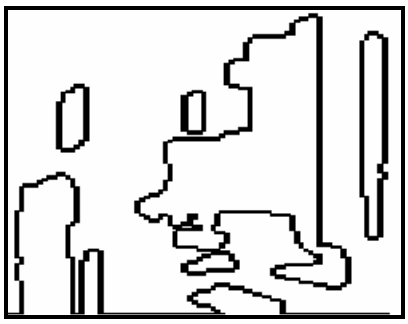
I. Global Task

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 locations of approximately diagonal lines in **P**.

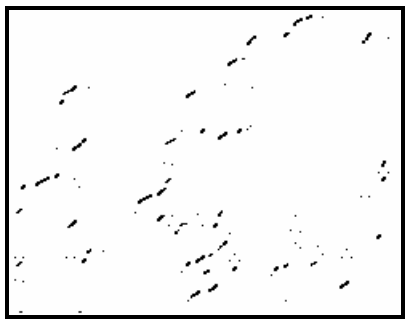
Remark:

Detects approximately diagonal lines being in the SW-NE direction. By modifying the positions of the elements of the **B** template, namely rotating **B**, the template can be sensitized to other directions as well (vertical, horizontal or NW-SE diagonal).

II. Example: image name: diag.bmp, image size: 246x191; template name: diag.tem .



input



output

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

Old names: DIAG1LIU

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | -1 | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | -1 |

$z =$ | -4 |
|---|

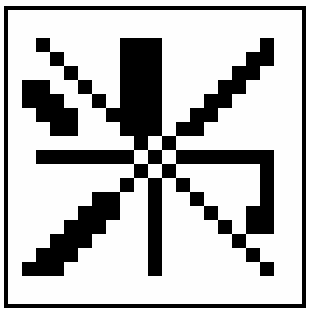
I. Global Task

Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X**(0) **=** 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 representing the locations of diagonal lines in **P**.
Template robustness:	$\rho = 0.45$.

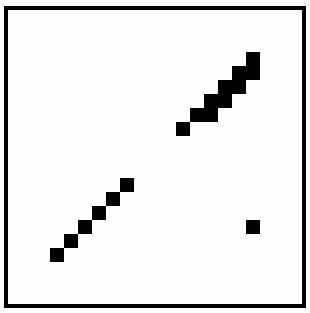
Remark:

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

GrayscaleDiagonalLineDetector: ***Grayscale diagonal line detector***

Old names: DIAGGRAY

A =

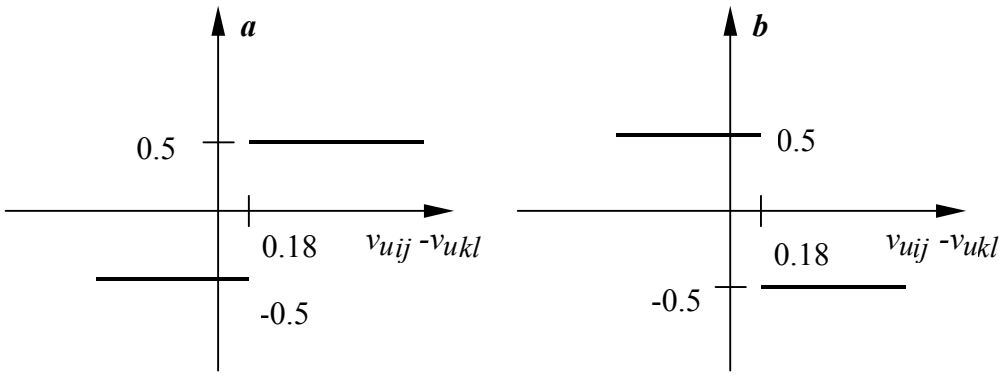
0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

B =

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

z = -1.8

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



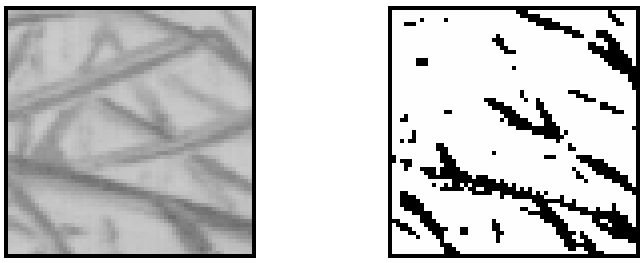
I. Global Task

Given:	static grayscale 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 where black pixels identify the diagonal lines of north-west, south-eastern direction in **P**.

Remark:

If the nonlinear **B** template is rotated by 90°, the grayscale line detector of the north-east, south-west direction will be obtained.

II. Example: image name: diaggray.bmp, image size: 61x61; template name: diaggray.tem .



input output

RotationDetector: ***Detects the rotation of compact objects in a binary image, having only horizontal and vertical edges; removes all inclined objects or objects having at least one inclined edge*** **[61]**

$$\mathbf{A} = \begin{bmatrix} -0.8 & 5 & -0.8 \\ 5 & 5 & 5 \\ -0.8 & 5 & -0.8 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -0.4 & -2.5 & -0.4 \\ -2.5 & 5 & -2.5 \\ -0.4 & -2.5 & -0.4 \end{bmatrix} \qquad z = \boxed{-11.2}$$

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 which retains from the initial state **P** only the compact objects with horizontal or vertical edges

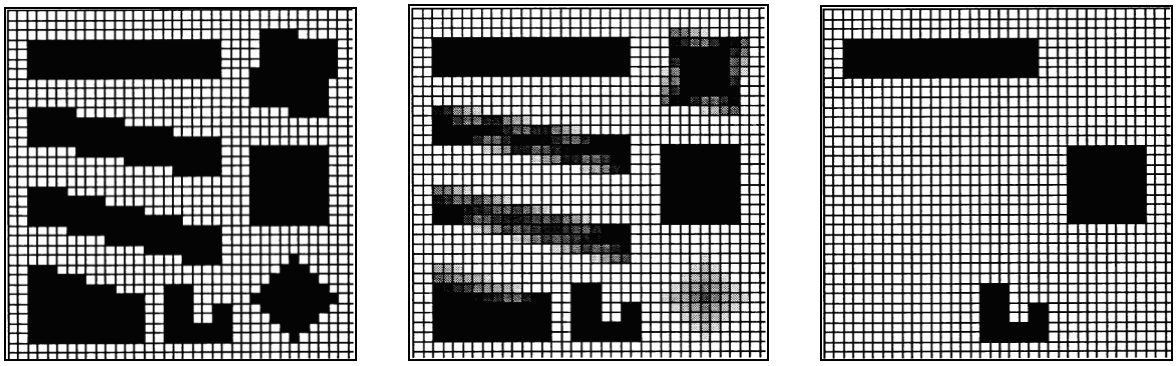
Remark:

The binary image is loaded as initial state and also applied at the input.

These templates can be used as a rotation detector, but only for a specific class of objects, namely with horizontal and vertical edges. Its robustness depends on the resolution of the CNN (number of cells) related to the object size. For larger objects, or a CNN with larger resolution, smaller rotation angles can be detected.

Every object having at least an inclined edge will be gradually deleted, as seen in the second image. Also the one pixel thick lines or curves will be removed. Binary noise can affect the operation. If a black or white parasitic pixel representing noise appears on the edge of an object, the object will be deleted, even if it has no inclined edge.

II. Example: image name: binary image; image size: 36x36



Initial state Intermediate state Output

HeatDiffusion: Heat-diffusion

Old names: DIFFUS

A = | 0.1 | 0.15 | 0.1 |
|---|---|---|
| 0.15 | 0 | 0.15 |
| 0.1 | 0.15 | 0.1 |

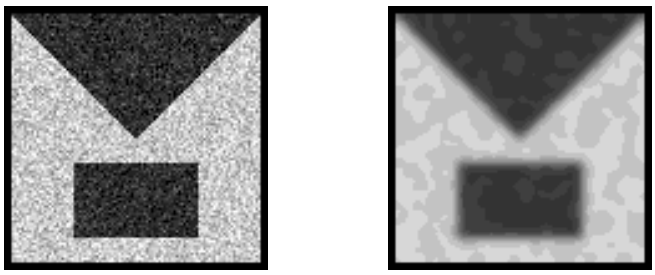
B = | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = | 0 |
|---|

I. Global Task

Given:	static noisy grayscale 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(T)** = Grayscale image representing the result of the heat diffusion operation.

II. Example: image name: diffus.bmp, image size: 106x106; template name: diffus.tem .



input output

EdgeDetection: Binary edge detection template

Old names: EdgeDetector, EDGE

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | -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

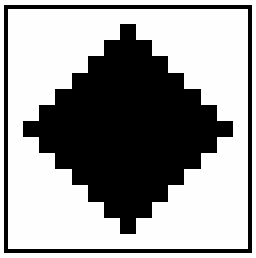
Output: **Y(t)⇒Y(∞)** = Binary image showing all edges of **P** in black

Template robustness: $\rho = 0.12$.

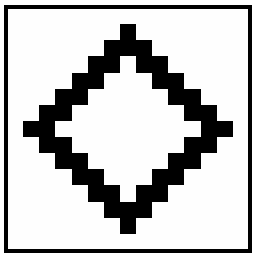
Remark:
Black pixels having at least one white neighbor compose the edge of the object.

II. Examples

Example 1: image name: logic05.bmp, image size: 44x44; template name: edge.tem .



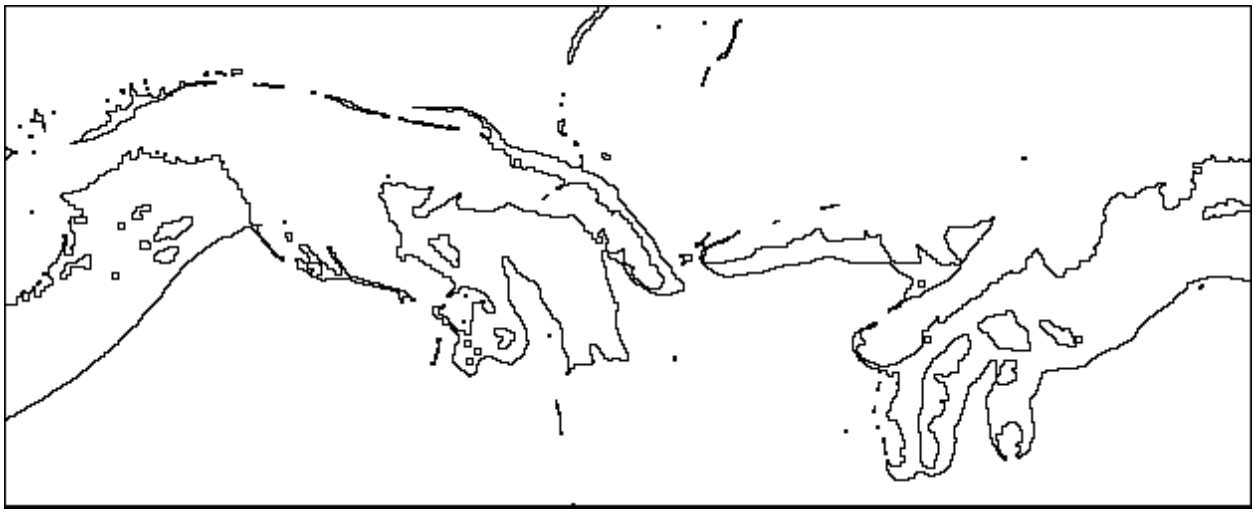
input



output

Example 2: image name: michelan.bmp, image size: 627x253; template name: edge.tem .

input



output

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

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.11 & 0 & 0.11 \\ -0.28 & 0 & 0.28 \\ -0.11 & 0 & 0.11 \end{bmatrix} \quad z = \boxed{0}$$

I. Global Task

Given: static grayscale image **P**

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

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

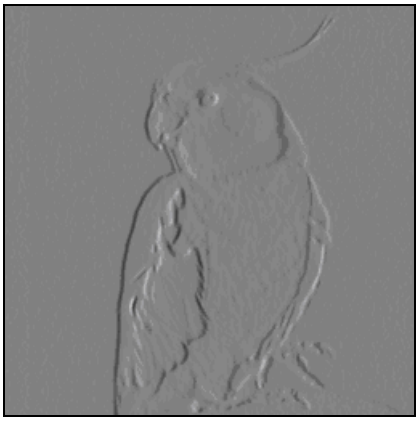
Boundary Conditions: Zero-flux boundary condition (duplicate)

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

MaskedObjectExtractor: ***Masked erase*** **[24]**

Old names: ERASMASK

Left-to-right

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 1.5 | 3 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1.5 | 0 |
| 0 | 0 | 0 |

$z =$ | -1.5 |
|---|

I. Global Task

Given: static binary images $\mathbf{P_1}$ (mask) and $\mathbf{P_2}$

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

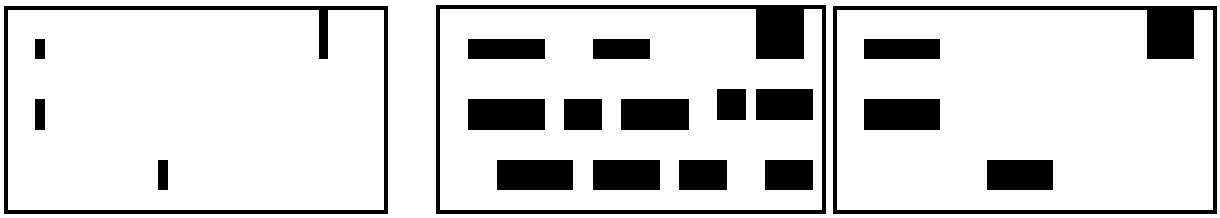
Initial State: $\mathbf{X(0) = P_2}$

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

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image that is the result of erasing $\mathbf{P_2}$ from left to right. Erasure is stopped by the black walls on the mask ($\mathbf{P_1}$) image.

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

II. Example: Left-to-right erase. Image names: ccdmsk3.bmp, ccdmsk2.bmp; image size: 40x20; template name: erasmask.tem .



GradientDetection: ***Finds the locations where the gradient of the field is smaller than a given threshold value*** **[9]**

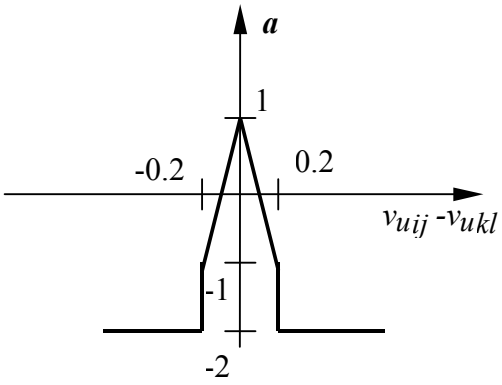
Old names: EXTREME

A = | 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

B = | ***a*** | ***a*** | ***a*** |
| ***a*** | 0 | ***a*** |
| ***a*** | ***a*** | ***a*** |

z = z^*

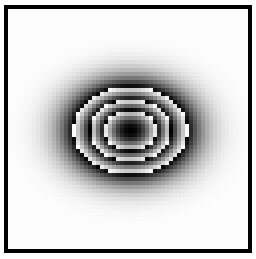
where z^* is a given threshold value, and ***a*** is defined by the following nonlinear function:



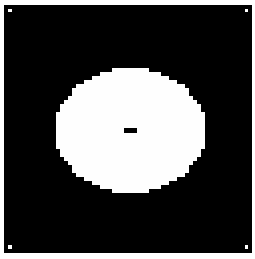
I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) = P**
Initial State:	**X(**0**) =** Arbitrary (in the example we choose **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 locations in **P** where the gradient of the field is smaller than a given threshold value.

II. Example: image name: circles.bmp, image size: 60x60; template name: extreme.tem . Threshold value z^* = 3.9 .



input



output

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

Old names: FigureRemover, FIGDEL

A = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

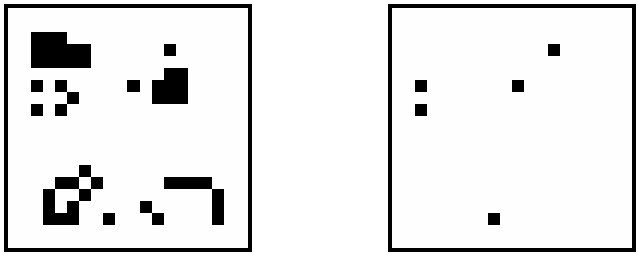
B = | -1 | -1 | -1 |
| --- | --- | --- |
| -1 | 1 | -1 |
| -1 | -1 | -1 |

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

PointRemoval: Deletes isolated black pixels

Old names: FigureExtractor, FIGEXTR

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad z = \boxed{-1}$$

I. Global Task

Given: static binary image **P**

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

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

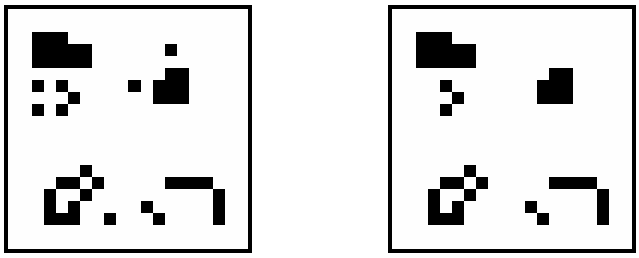
Boundary Conditions: Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0

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 *FigureRemover*.

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



input output

SelectedObjectsExtraction: ***Extracts marked objects***

Old names: FigureReconstructor, FIGREC, RECALL

$\mathbf{A} =$ | 0.5 | 0.5 | 0.5 |
|---|---|---|
| 0.5 | 4 | 0.5 |
| 0.5 | 0.5 | 0.5 |

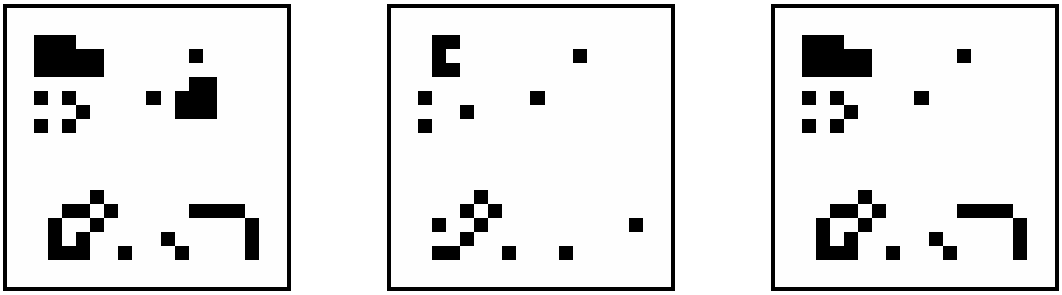
$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 4 | 0 |
| 0 | 0 | 0 |

$z =$ | 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

FilledContourExtraction: ***Finds solid black framed areas***

Old names: FramedAreasFinder, FINDAREA

$\mathbf{A}$ =

0	1	0
1	5	1
0	1	0

$\mathbf{B}$ =

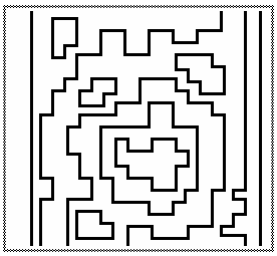
0	0	0
0	2	0
0	0	0

z = -5.25

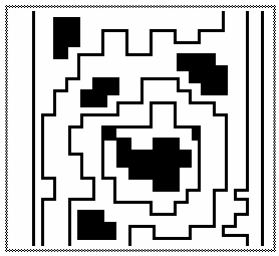
I. Global Task

Given:	two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$
Input:	$\mathbf{U(t) = P_1}$
Initial State:	$\mathbf{X(0) = P_2}$
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 objects of $\mathbf{P_2}$ which totally fit/fill in closed curves of $\mathbf{P_1}$.

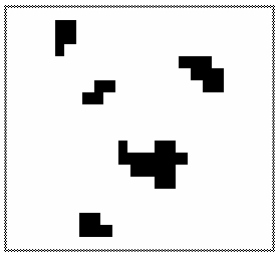
II. Example: image names: findare1.bmp, findare2.bmp; image size: 270x246; template name: findarea.tem .



input



initial state



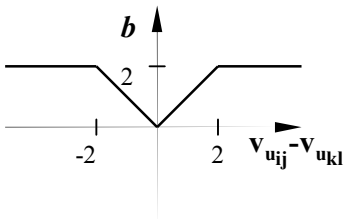
output

ThresholdedGradient: ***Finds the locations where the gradient of the field is higher than a given threshold value*** **[9]**

Old names: GRADIENT

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \\ \boldsymbol{b} & 0 & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \qquad z = \boxed{z^*}$$

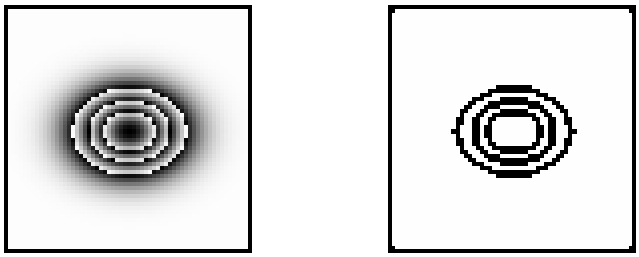
where z^* is a given threshold value, and $\boldsymbol{b}$ is defined by the following nonlinear function:



I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) = P**
Initial State:	**X(0) =** Arbitrary (in the example we choose **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 locations in **P** where the gradient of the field is higher than a given threshold value.

II. Example: image name: circles.bmp, image size: 60x60; template name: gradient.tem . Threshold value $z^* = -4.8$.



input output

3x3Halftoning: 3x3 image halftoning

Old names: HLF3, HLF33

$$\mathbf{A} = \begin{bmatrix} -0.07 & -0.1 & -0.07 \\ -0.1 & 1+\varepsilon & -0.1 \\ -0.07 & -0.1 & -0.07 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.1 & 0.07 \\ 0.1 & 0.32 & 0.1 \\ 0.07 & 0.1 & 0.07 \end{bmatrix} \quad z = \boxed{0}$$

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**.

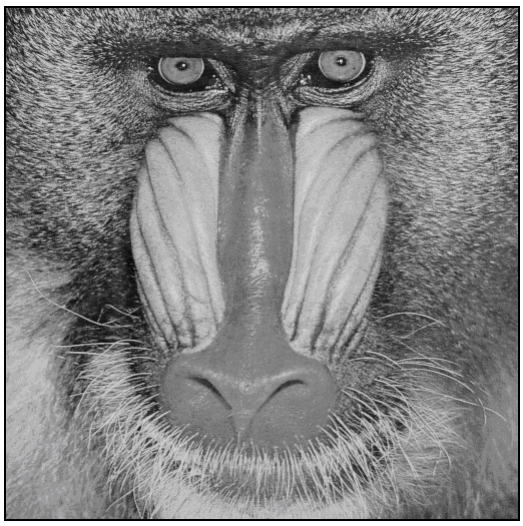
Remark:

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].

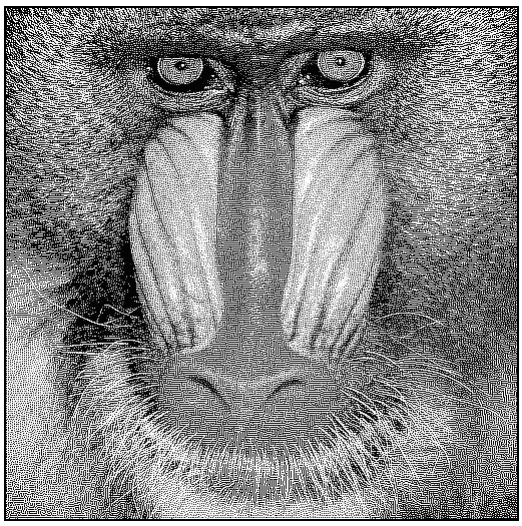
This template is called "Half-Toning" in [44].

II. Examples

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



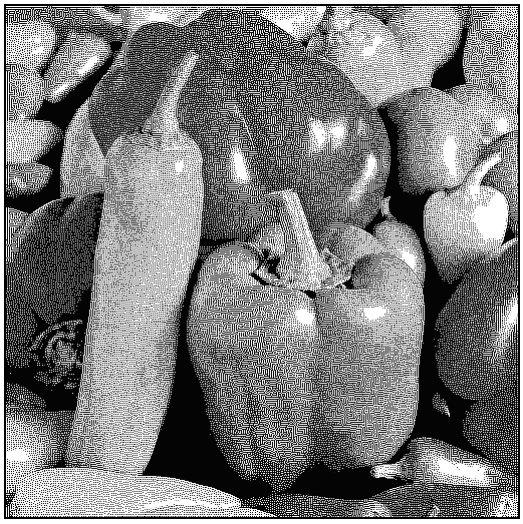
input



output

Example 2: image name: peppers.bmp, image size: 512x512; template name: hlf3.tem .

input



output

5x5Halftoning1: ***5x5 image halftoning*** **[15]**

Old names: HLF5KC, HLF55_KC

A =

-0.03	-0.09	-0.13	-0.09	-0.03
-0.09	-0.36	-0.60	-0.36	-0.09
-0.13	-0.60	1.05	-0.60	-0.13
-0.09	-0.36	-0.60	-0.36	-0.09
-0.03	-0.09	-0.13	-0.09	-0.03

B =

0	0	0.07	0	0
0	0.36	0.76	0.36	0
0.07	0.76	2.12	0.76	0.07
0	0.36	0.76	0.36	0
0	0	0.07	0	0

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

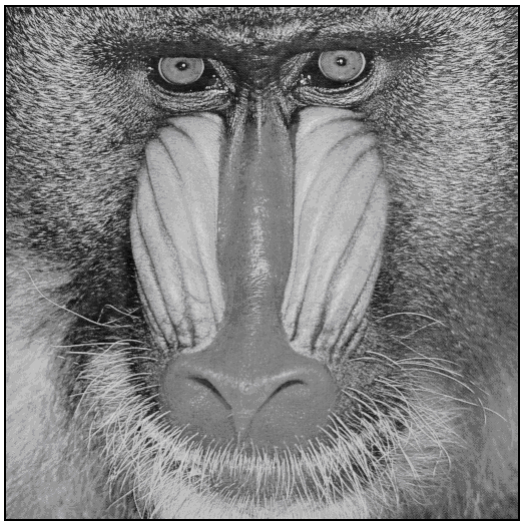
Output: **Y(t)⇒Y(∞)** = Binary image preserving the main features of **P**.

Remark:

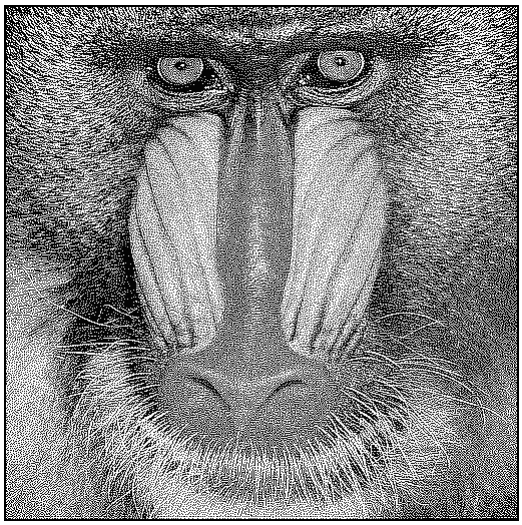
The output image quality is optimized by considering human visualisation. When simulating the behavior of the CNN by using the forward Euler integration form, the time step should be less than 0.4.

II. Examples

Example 1: image name: baboon.bmp, image size: 512x512; template name: hlf5kc.tem .



input



output

Example 2: image name: peppers.bmp, image size: 512x512; template name: hlf5kc.tem .

input

output

5x5Halftoning2: 5x5 image halftoning

Old names: HLF5, HLF55

A =

-0.02	-0.07	-0.10	-0.07	-0.02
-0.07	-0.32	-0.46	-0.32	-0.07
-0.10	-0.46	1.05	-0.46	-0.10
-0.07	-0.32	-0.46	-0.32	-0.07
-0.02	-0.07	-0.10	-0.07	-0.02

B =

0.02	0.07	0.10	0.07	0.02
0.07	0.32	0.46	0.32	0.07
0.10	0.46	0.81	0.46	0.10
0.07	0.32	0.46	0.32	0.07
0.02	0.07	0.10	0.07	0.02

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

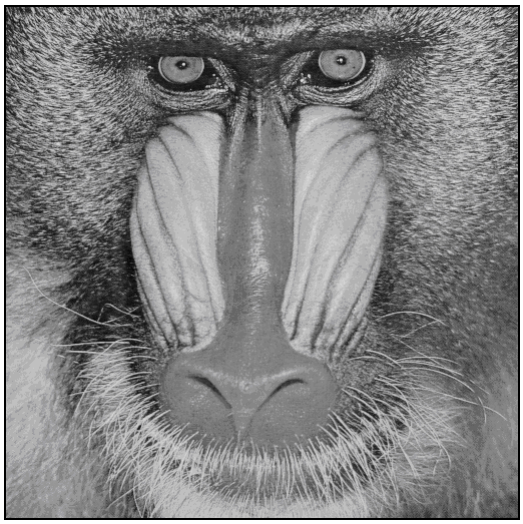
Output: **Y(t)⇒Y(∞)** = Binary image preserving the main features of **P**.

Remark:

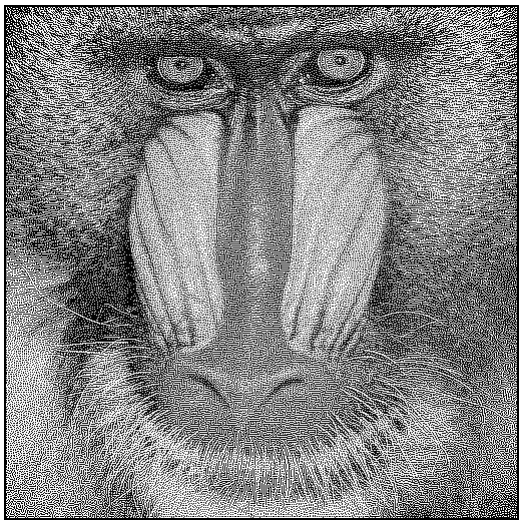
The inverse of the template is *5x5InverseHalftoning*. The result is optimal in the Square Error measure [17,35].

II. Examples

Example 1: image name: baboon.bmp, image size: 512x512; template name: hlf5.tem .



input



output

Example 2: image name: peppers.bmp, image size: 512x512; template name: hlf5.tem .

input

output

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

Old names: HoleFiller, HOLE

$\mathbf{A} =$ | 0 | 1 | 0 |
|---|---|---|
| 1 | 3 | 1 |
| 0 | 1 | 0 |

$\mathbf{B} =$ | 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

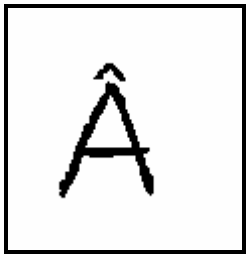
Output: **Y(t)⇒Y(∞)** = Binary image representing **P** with holes filled.

Remark:

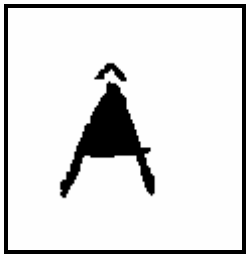
(i) this is a propagating template, the computing time is proportional to the length of the image

(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

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

Old names: 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**

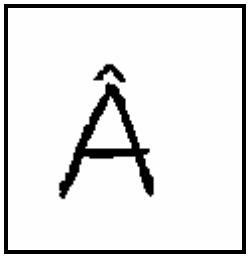
Boundary Conditions: Zero-flux boundary condition (duplicate)

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

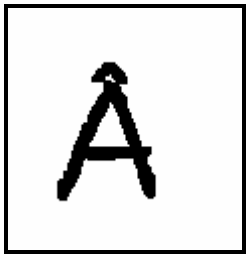
Remark:

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

3x3InverseHalftoning: ***Inverts the halftoned image by a 3x3 template***

Old names: INVHLF3, INVHLF33

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.1 & 0.07 \\ 0.1 & 0.32 & 0.1 \\ 0.07 & 0.1 & 0.07 \end{bmatrix} \quad z = \boxed{0}$$

I. Global Task

Given: static binary image **P** obtained by using the 3x3Halftoning template

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(∞)** = Grayscale image representing **P**.

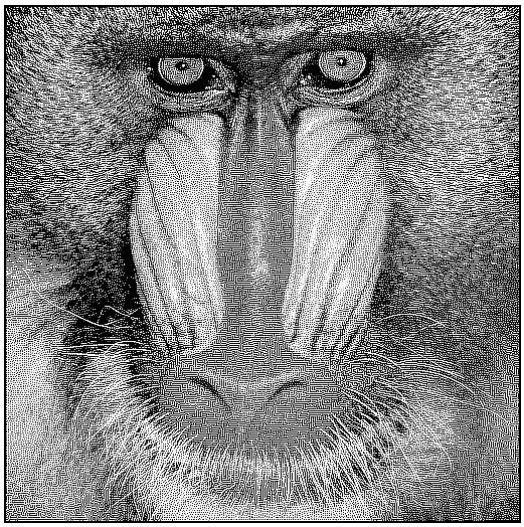
Remark:

Inverts the 3*3 halftoned image created by the *3x3Halftoning* template. The result lacks fine edges as the control of *3x3Halftoning* smoothes the input [17,35].

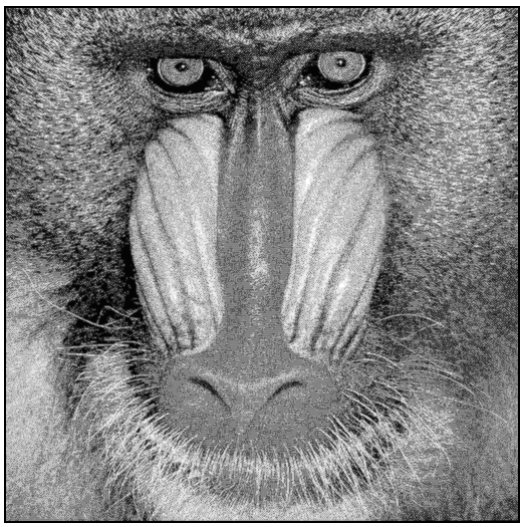
This template is called "Inverse Half-Toning" in [44].

II. Examples

Example 1: image name: invhlf3_1.bmp, image size: 512x512; template name: invhlf3.tem .

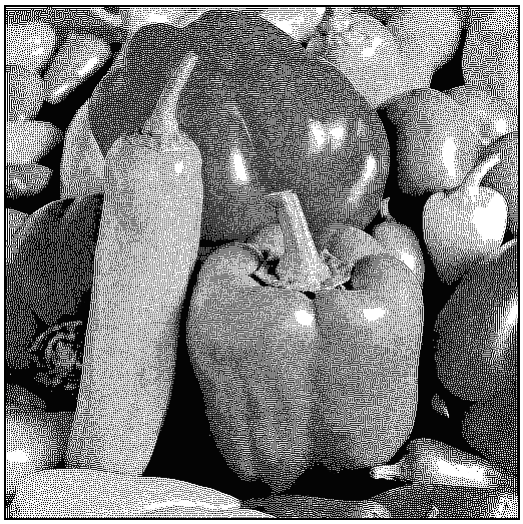


input

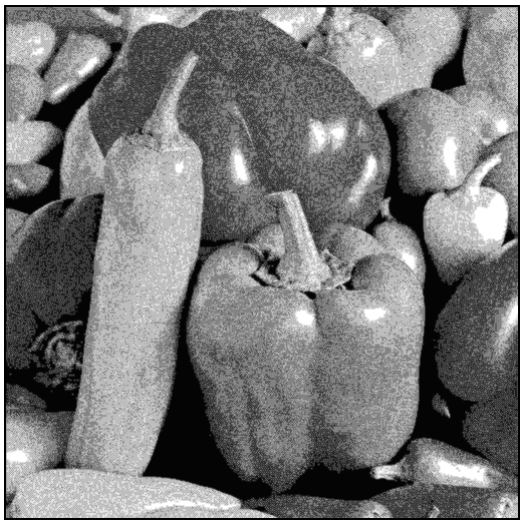


output

Example 2: image name: invhlf3_2.bmp, image size: 512x512; template name: invhlf3.tem .



input



output

5x5InverseHalftoning: ***Inverts the halftoned image by a 5x5 template***

Old names: INVHLF5, INVHLF55

A =

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

B =

0	0.01	0.02	0.01	0
0.01	0.06	0.09	0.06	0.01
0.02	0.09	0.16	0.09	0.02
0.01	0.06	0.09	0.06	0.01
0	0.01	0.02	0.01	0

z = 0

I. Global Task

Given: static binary image **P** obtained by using the 5x5Halftoning2 template

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

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

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

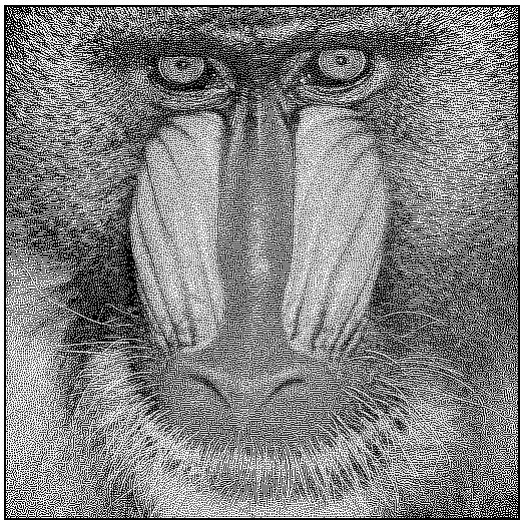
Output: **Y(t)⇒Y(∞)** = Grayscale image representing **P**.

Remark:

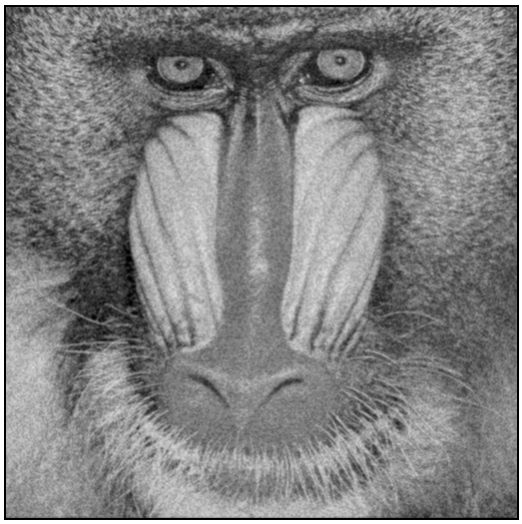
Inverts the 5*5 halftoned image created by the *5x5Halftoning2* template. The result lacks fine edges as the control of *5x5Halftoning2* smoothes the input.

II. Examples

Example 1: image name: invhlf5_1.bmp, image size: 512x512; template name: invhlf5.tem .



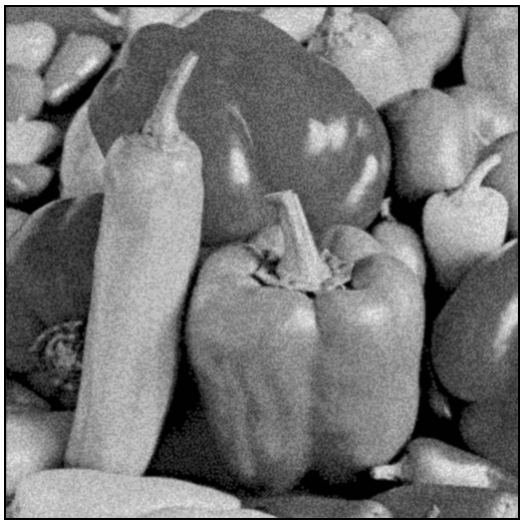
input



output

Example 2: image name: invhlf5_2.bmp, image size: 512x512; template name: invhlf5.tem .

input



output

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

Old names: LSE

$\mathbf{A} =$

0	0	0
0	1	0
0	0	0

$\mathbf{B} =$

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

$z =$ -3

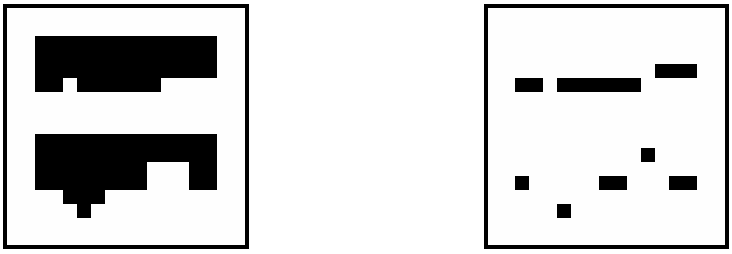
I. Global Task

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

PatternMatchingFinder: ***Finds matching patterns***

Old names: MATCH

$$\mathbf{A}=\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}\qquad \mathbf{B}=\begin{bmatrix}\boldsymbol{b}&\boldsymbol{b}&\boldsymbol{b}\\\boldsymbol{b}&\boldsymbol{b}&\boldsymbol{b}\\\boldsymbol{b}&\boldsymbol{b}&\boldsymbol{b}\end{bmatrix}\qquad z=\boxed{-N+0.5}$$

where

$$\boldsymbol{b}=\begin{cases}1, & \text{if corresponding pixel is required to be black}\\ 0, & \text{if corresponding pixel is do not care}\\ -1, & \text{if corresponding pixel is required to be white}\end{cases}$$

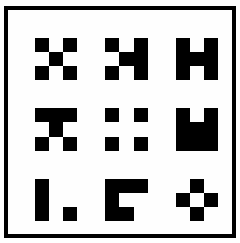
N = number of pixels required to be either black or white, i.e. the number of non-zero values in the **B** template

I. Global Task

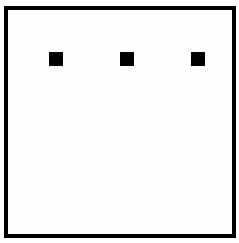
Given:	static binary image **P** possessing the 3x3 pattern prescribed by the template
Input:	**U(t) = P**
Initial State:	**X(0) =** Arbitrary (in the examples we choose $x_{ij}(0)=0$)
Boundary Conditions:	Fixed type, $u_{ij} = 0$ for all virtual cells, denoted by [U]=0
Output:	**Y(t)⇒Y(∞)** = Binary image representing the locations of the 3x3 pattern prescribed by the template. The pattern having a black/white pixel where the template value is +1/-1, respectively, is detected.

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

$$\mathbf{A}=\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}\qquad \mathbf{B}=\begin{bmatrix}1&-1&1\\0&1&0\\1&-1&1\end{bmatrix}\qquad z=\boxed{-6.5}$$



input



output

LocalMaximaDetector: ***Local maxima detector template*** **[33]**

Old names: MAXLOC

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \\ \boldsymbol{b} & 0 & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{b} & \boldsymbol{b} \end{bmatrix} \qquad z = \boxed{-3.5}$$

where

$$\boldsymbol{b} = \begin{cases} 0.5 & \text{if } v_{u_{ij}} - v_{u_{kl}} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

I. Global Task

Given: static grayscale image **P**

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

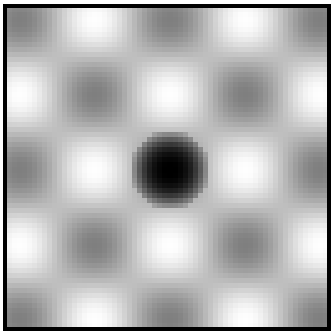
Initial State: **X(0) = 0**

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

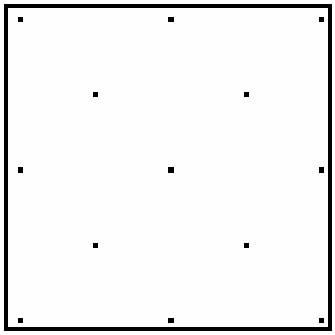
Output: **Y(t)⇒Y(∞)** = Binary image representing the locations of all local maxima in the specified local neighborhood.

Remark:
All local minima can also be detected if the input image is inverted.

II. Example: image name: avergra1.bmp, image size: 64x64; template name: maxloc.tem .



input



output

MedianFilter: ***Removes impulse noise from a grayscale image*** **[34]**

Old names: MEDIAN

CNN base model of the median filter (linear region of f):

$$C\frac{d}{dt}v_{x_{ij}}(t) = -R^{-1}v_{x_{ij}}(t) + Af(v_{x_{ij}}) + \sum_{kl\in N_r}\hat{D}^{M}{}_{ij,kl}(v_{x_{ij}}(t) - v_{u_{kl}}(t))$$

The corresponding CNN template (T_M) ($R = 1, C = 1, Q = 1$):

$$\mathbf{A} = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \qquad \hat{D}^{M} = \begin{array}{|c|c|c|}\hline \boldsymbol{d} & \boldsymbol{d} & \boldsymbol{d} \\ \hline \boldsymbol{d} & 0 & \boldsymbol{d} \\ \hline \boldsymbol{d} & \boldsymbol{d} & \boldsymbol{d} \\ \hline \end{array} \qquad z = \begin{array}{|c|}\hline 0 \\ \hline \end{array}$$

where $\boldsymbol{d}$ = -Q sign($v_{x_{ij}}$ - $v_{u_{kl}}$).

Special types derived from the base model:

- **(i)** **Rank Order Class** **-** simply varying the bias value
  / e.g. **Min filter** : I=+8Q, **Max filter** :I=-8Q /
- **(ii)** **Weighted Median** **-** locally space variant weighting
  / e.g. Plus-shape Median Filter (Q=0 in the corners) /
- **(iii)** **Cascade Median** **-** consecutive steps with T_M,
  intermediate result is stored in LAMs of the CNNUM
- **(iv)** **Selective Median** **-** filtering is made only at the locations
  of the local extremities with T_M - fixed-state CNN model
- **(v)** **MinMax, MaxMin, Pseudo Median -** combining (i)-(iv)

I. Global Task

Given:	static grayscale image **P** with impulse noise
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(∞)** = Grayscale image representing **P** filtered from impulse noise.

Remark:

Impulse noise is removed only if the impulses are placed further from each other than half of the window width (r+1).

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

input

output

LeftPeeler: ***Peels one pixel from the left*** **[14]**

Old names: PEELHOR

A = | 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

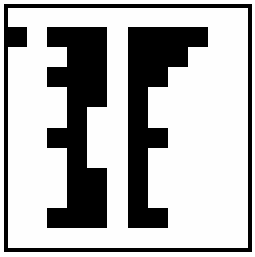
B = | 0 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |

z = | -1 |

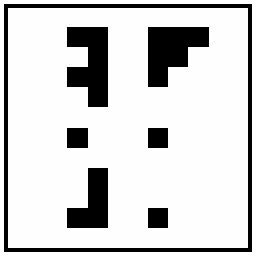
I. Global Task

Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(**0**) =** Arbitrary
Boundary Conditions:	Fixed type, u_{ij} = -1 for all virtual cells, denoted by [U]= -1
Output:	**Y(t)⇒Y(∞)** = Binary image representing the objects of **P** peeled with one pixel from the left.
Template robustness:	ρ = 0.71 .

II. Example: image name: peelhor.bmp, image size: 12x12; template name: peelhor.tem .



input



output

RightEdgeDetection: Extracts right edges of objects

Old names: RightContourDetector, RIGHTCON

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 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:	ρ = 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

***MaskedShadow: Masked shadow* [24]**

Old names: SHADMASK, MASKSHAD

Left-to-right

A =

0	0	0
1.5	1.8	0
0	0	0

B =

0	0	0
0	-1.2	0
0	0	0

z = 0

I. Global Task

Given: static binary images $\mathbf{P_1}$ (mask) and $\mathbf{P_2}$

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

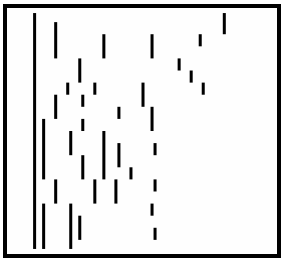
Initial State: $\mathbf{X(0) = P_2}$

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

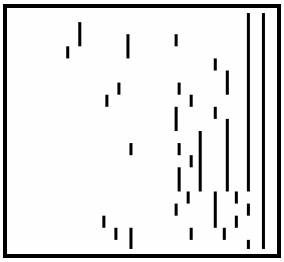
Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image representing the result of pattern propagation of $\mathbf{P_2}$ in a particular direction. The propagation goes from the direction of the non-zero off-center feedback template entry $\mathbf{A}_{ij}$ and is halted by the mask $\mathbf{P_1}$.

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

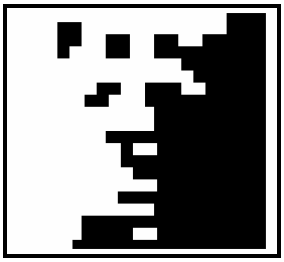
II. Example: Right-to-left horizontal shadow. Image names: shdmsk1.bmp, shdmsk2.bmp; image size: 270x246; template name: shadmask.tem .



input



initial state



output

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

Old names: LeftShadow, SHADOW

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

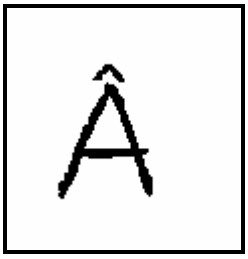
Output: **Y(t)⇒Y(∞)** = Binary image representing the left shadow of the objects in **P**.

Remark:

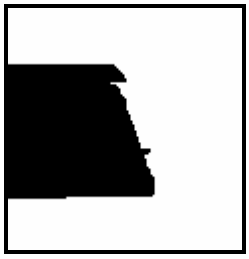
By modifying the position of the off-center **A** template element the template can be sensitized to other directions as well.

II. Examples

Example 1: Left shadow. Image name: a_letter.bmp, image size: 117x121; template name: shadow.tem .



input



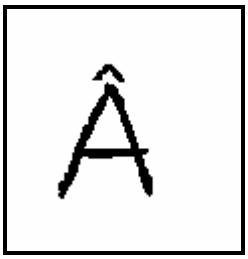
output

Example 2: Shadow in the east-western direction. Image name: a_letter.bmp, image size: 117x121.

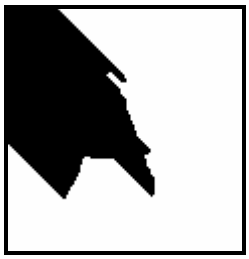
A = | 0 | 0 | 0 |
|---|---|---|
| 0 | 2 | 0 |
| 0 | 0 | 2 |

B = | 0 | 0 | 0 |
|---|---|---|
| 0 | 2 | 0 |
| 0 | 0 | 0 |

z = 0



input



output

VerticalShadow: ***Vertical shadow template***

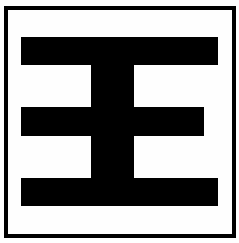
Old names: SHADSIM, SUPSHAD

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{2}$$

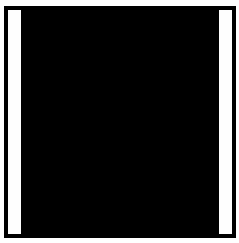
I. Global Task

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.

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



input



output

DirectedGrowingShadow: ***Generate growing shadows starting from black points***

SHADOW0:

$$\mathbf{A} = \begin{bmatrix} 0.4 & 0.3 & 0 \\ 1 & 2 & -1 \\ 0.4 & 0.3 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{2.5}$$

SHADOW45:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{2.5}$$

I. Global Task

Given: static binary image **P**

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

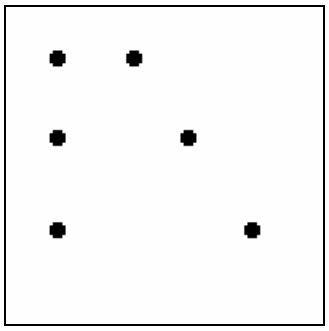
Initial State: **X(0) = P**

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

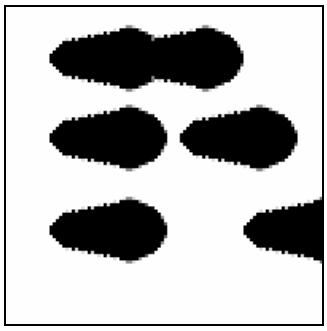
Output: **Y(t)⇒Y(∞)** = Binary image in which shadows are generated starting from black pixels. During the transient shadows become wider and wider.

Remark:
Other directions can be gained by appropriate rotation and flipping of the templates.

II. Example: image name: points.bmp; image size: 100x100; template names: shadow0.tem, shadow45.tem.



input



output of shadow0 template ($t=35\tau_{CNN}$)

output of shadow45 template ($t=45\tau_{CNN}$)

Threshold: ***Grayscale to binary threshold template***

Old names: TRESHOLD

A = | 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z = \boxed{-z^*}$, $-1 < z^* < 1$

I. Global Task

Given:	static grayscale image **P** and threshold z^*
Input:	**U(t) =** Arbitrary or as a default U(t)=0
Initial State:	**X(0) = P**
Output:	**Y(t)⇒Y(∞)** = Binary image where black pixels correspond to pixels in **P** with grayscale intensity $p_{ij} > z^*$.

II. Example: Threshold value $z^* = 0.4$. Image name: madonna.bmp, image size: 59x59; template name: treshold.tem .

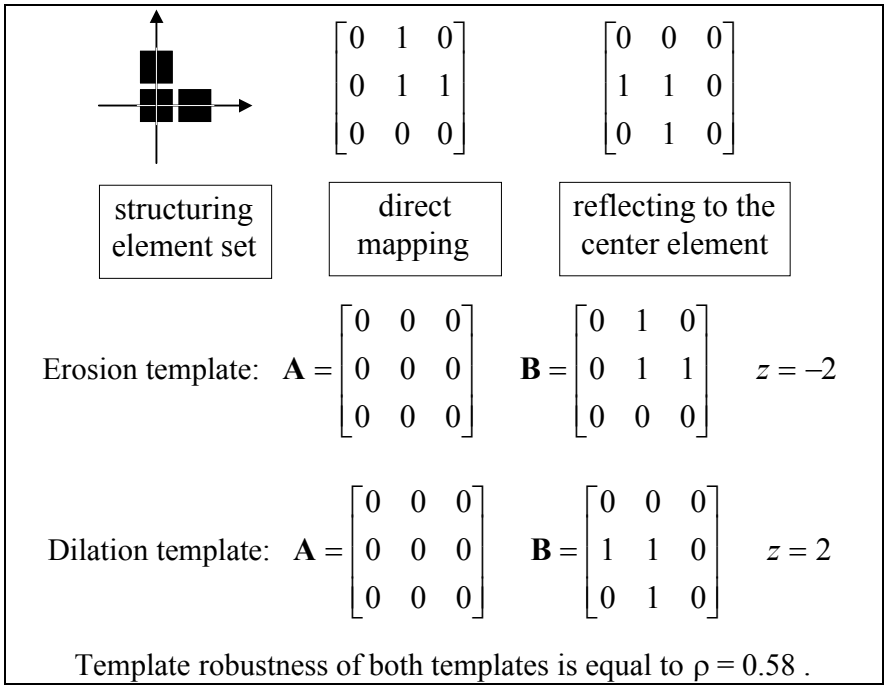
input

output

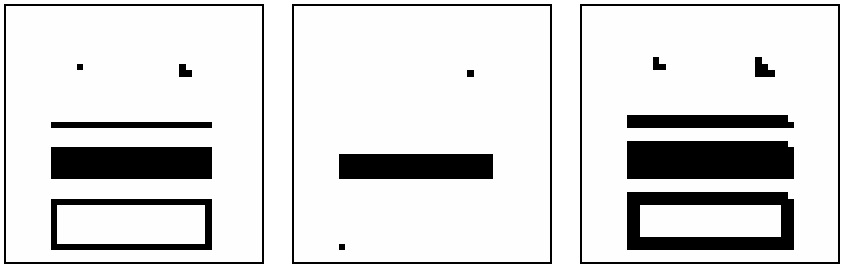
1.2. MATHEMATICAL MORPHOLOGY

BINARY MATHEMATICAL MORPHOLOGY

The basic operations of binary mathematical morphology [32] are erosion and dilation. These operations are defined by two binary images, one being the operand, the other the structuring element. In the CNN implementation, the former image is the input, while the function (the templates) itself depends on the latter image. If the structuring element set does not exceed the size of the CNN template the dilation and erosion operators can be implemented with a single CNN template. The implementation method is the following: The **A** template matrix is zero in every position. The structuring element set should be directly mapped to the **B** template (See Figure). If it is an erosion operator, the z value is equal to (1-n), where n is the number of 1s in the **B** template matrix. If it is a dilation operator, the **B** template must be reflected to the center element, and the z value is equal to (n-1), where n is the number of 1s in the **B** template matrix. When calculating the operator, the image should be put to the input of the CNN, and the initial condition is zero everywhere. The next Figure shows an example for template syntetization.



Example: Erosion and dilation with the given structuring element set. Image name: binmorph.bmp; image size: 40x40; template names: eros_bin.tem, dilat_bin.tem .



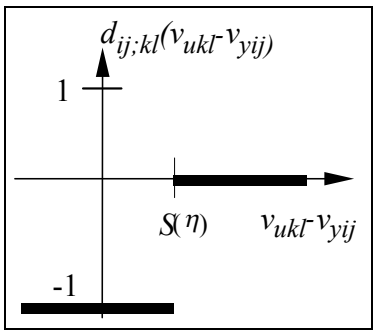
GRAYSCALE MATHEMATICAL MORPHOLOGY

The basic operations of grayscale mathematical morphology [32] are erosion and dilation. These operations are defined by two grayscale images, one being the operand, the other the structuring element set (S). In the CNN implementation, the former image is the input, while the function (the templates) itself depends on the latter image. If the structuring element set does not exceed the size of the CNN template the dilation and erosion operators can be implemented with a single CNN template. The implementation method is the following: single template grayscale mathematical morphology is implemented on a slightly modified CNN. The state equation of the modified CNN is the following:

$$\dot{v}_{xij}(t) = -v_{xij}(t) + v_{yij}(t) + \sum_{kl \in N_r(ij)} \hat{D}_{ij;kl}(v_{ukl}(t) - v_{yij}(t)) + z$$

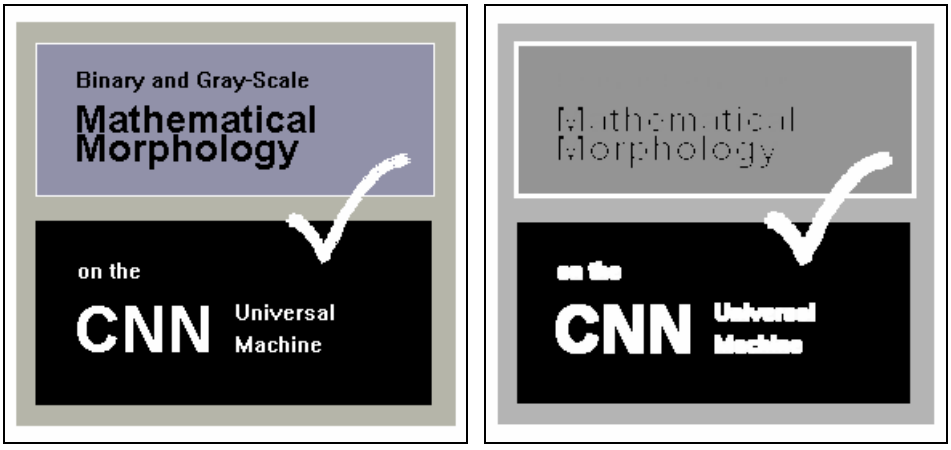
It means that **A**=[1], and the inputs of the nonlinear functions of the **D** template are the difference between the input values and the appropriate neighborhood positions, and the center output value. The morphological operation can be implemented with a single template on this CNN structure.
The erosion template is the following:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} d_{-1,1} & d_{0,1} & d_{1,1} \\ d_{-1,0} & d_{0,0} & d_{1,0} \\ d_{-1,-1} & d_{0,-1} & d_{1,-1} \end{bmatrix}, \quad z = 1$$



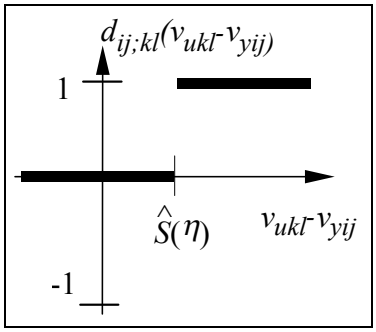
where: $S(\eta)$ is the real value of the structuring element set (S) at point $\eta=(k,l)$. If it is not defined $d_{ij;kl} \equiv 0$.

Example for grayscale erosion with a 3x3 square shaped zero value structuring element set. The black areas have shrunk. Image name: grmorph.bmp; image size: 288x272.



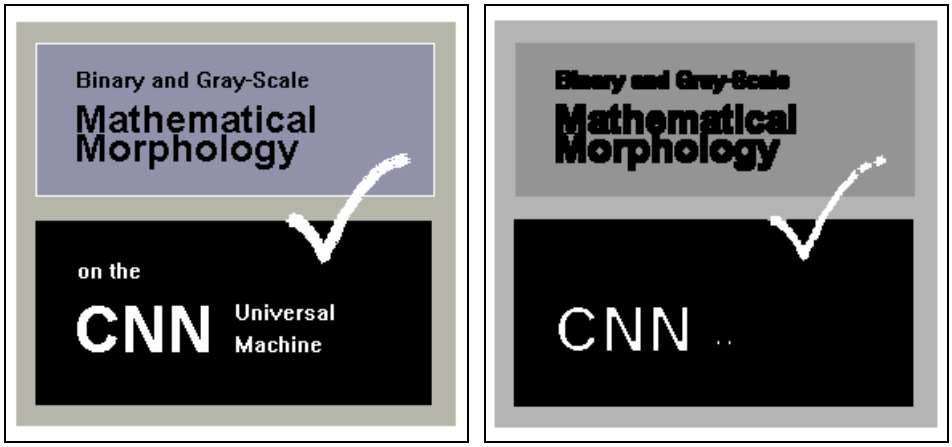
The dilation template is the following:

$$\mathbf{A}=\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix},\quad \mathbf{D}=\begin{bmatrix}d_{-1,1} & d_{0,1} & d_{1,1}\\ d_{-1,0} & d_{0,0} & d_{1,0}\\ d_{-1,-1} & d_{0,-1} & d_{1,-1}\end{bmatrix},\quad z=-1$$



where: $\hat{S}(\eta)$ is the real value of the inverted and reflected structuring element set ($\hat{S}=-S(-x)$) at point $\eta=(k,l)$. If it is not defined $d_{ij;kl}\equiv 0$.

Example for grayscale dilation with a 3x3 square shaped zero value structuring element set. The black areas have dilated. Image name: grmorph.bmp; image size: 288x272.



1.3. SPATIAL LOGIC

BlackFiller: ***Drives the whole network into black***

Old names: FILBLACK, BLACK

A = | 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = 4

I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) =** Arbitrary or as a default U(t)=0
Initial State:	**X(0) = P**
Output:	**Y(t)⇒Y(∞)** = black image (all pixels are black)

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

initial state

output

WhiteFiller: ***Drives the whole network into white***

Old names: FILWHITE, WHITE

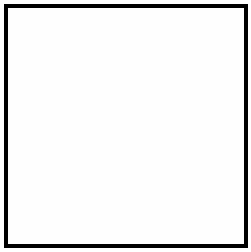
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{-4}$$

I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) =** Arbitrary or as a default U(t)=0
Initial State:	**X**(0) **= P**
Output:	**Y(t)⇒Y(∞)** = white image (all pixels are white)

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

initial state



output

BlackPropagation: ***Starts omni-directional black propagation from black pixels [54]***

A = | 0.25 | 0.25 | 0.25 |
| 0.25 | 3 | 0.25 |
| 0.25 | 0.25 | 0.25 |

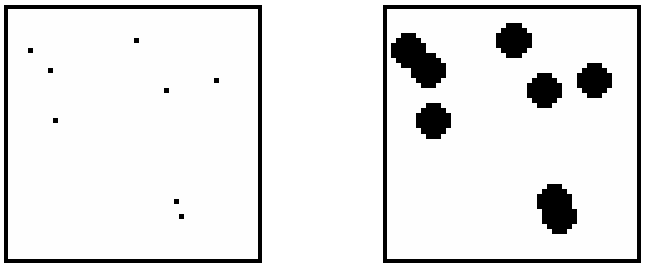
B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = 3.75

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)** = Binary image showing black objects in **P** with increasing black neighborhood (white objects decreasing in size).

II. Example: image name: points.bmp, image size: 50x50; template name: bprop.tem .



input output

WhitePropagation: ***Starts omni-directional white propagation from white pixels [54]***

A = | 0.25 | 0.25 | 0.25 |
| 0.25 | 3 | 0.25 |
| 0.25 | 0.25 | 0.25 |

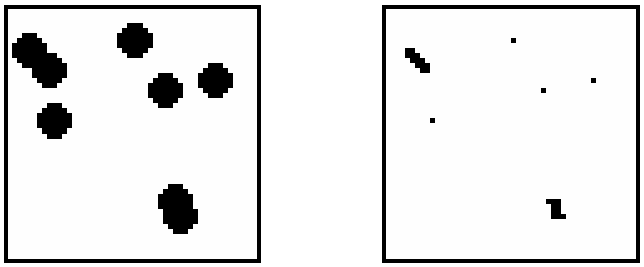
B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = -3.75

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)** = Binary image showing white objects in **P** with increasing white neighborhood (black objects decreasing in size).

II. Example: image name: patches.bmp, image size: 50x50; template name: wprop.tem .



input output

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

Old names: HOLLOW

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 2 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{3.25}$$

I. Global Task

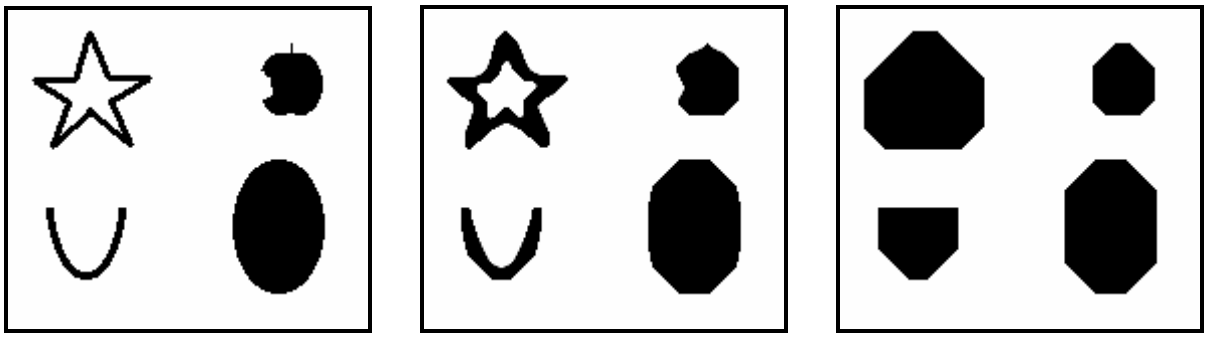
Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(0) = P**
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = Binary image in which the concave locations of objects are black.

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 ($t=\infty$)

ConcaveArcFiller: ***Fills the concave arcs of objects to prescribed direction***

FILL35:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{2}$$

FILL65:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{3}$$

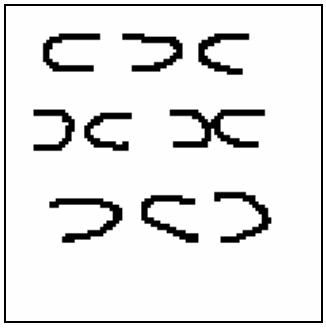
I. Global Task

Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X**(0) **= P**
Boundary Conditions:	Fixed type, y_{ij} = -1 for all virtual cells, denoted by [Y]=-1
Output:	**Y(t)⇒Y(∞)** = Binary image in which those arcs of objects are filled which have a prescribed orientation.

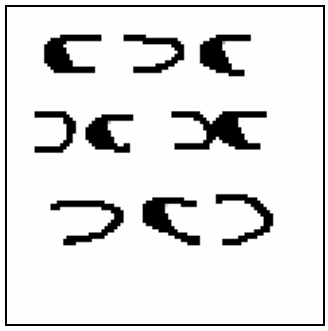
Remark:

In general, the objects of **P** that are not filled should have at least 2 pixel wide contour. Otherwise the template may not work correctly.

II. Example: image name: arcs.bmp, image size: 100x100; template name: arc_fill.tem .



input



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

SurfaceInterpolation: Interpolates a smooth surface through given points

Old names: INTERP, INTERPOL

A =

0	0	-2	0	0
0	-4	16	-4	0
-2	16	-39	16	-2
0	-4	16	-4	0
0	0	-2	0	0

B =

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

z = 0

I. Global Task

Given: a static grayscale image $\mathbf{P_1}$ and a static binary image $\mathbf{P_2}$

Input: $\mathbf{U(t)}$ = Arbitrary or as a default U(t)=0

Initial State: $\mathbf{X(0) = P_1}$

Fixed State Mask: $\mathbf{X_{fix} = P_2}$

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

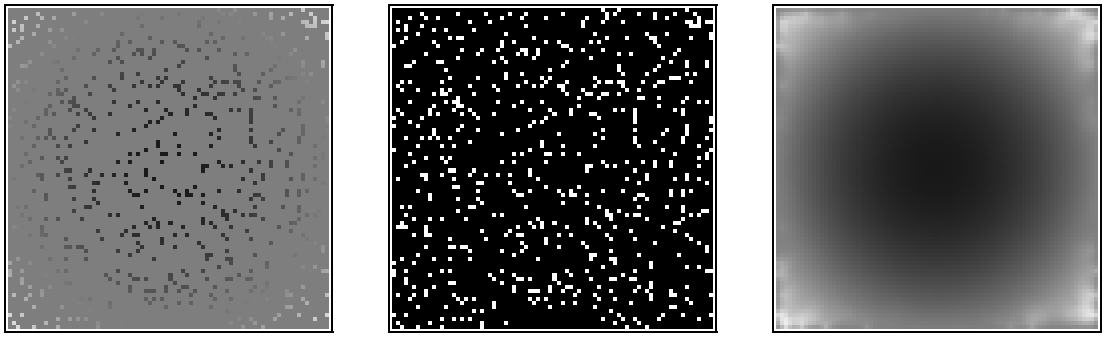
Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Grayscale image representing an interpolated surface that fits the given points and is as smooth as possible.

Remark:

Images $\mathbf{P_1}$ and $\mathbf{P_2}$ are to be constructed as follows: if the altitude of the surface to be interpolated is known at point (i,j), it is preset in $\mathbf{P_1}$ (state) at the position (i,j), and the state is kept fixed ($\mathbf{P_2}$(i,j) = -1) during the transient. If the altitude is not known, then zero is filled into the state ($\mathbf{P_1}$(i,j) = 0) and changing of the state is allowed ($\mathbf{P_2}$(i,j) = 1). For exact solution the feedback template must be space variant at the borders. An approximate result can be obtained by using space invariant network. Further information about the space variant network is available in [27] and [29].

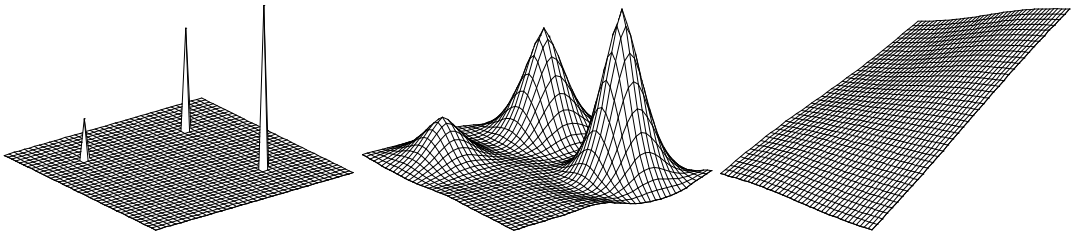
II. Examples

Example 1: Ball surface reconstruction (10% of the points is known). Image names: interp1.bmp, interp2.bmp; image size: 80x80; template name: interp.tem .



initial state fixed state mask output

Example 2: Fitting a surface on three given points. Image size: 80x80.



initial state intermediate step output

JunctionExtractor: ***Extracts the junctions of a skeleton*** **[22]**

Old names: JUNCTION

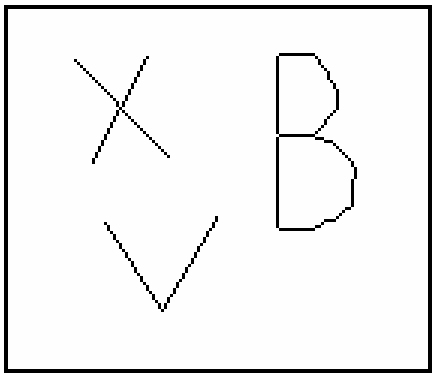
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad z = \boxed{-3}$$

I. Global Task

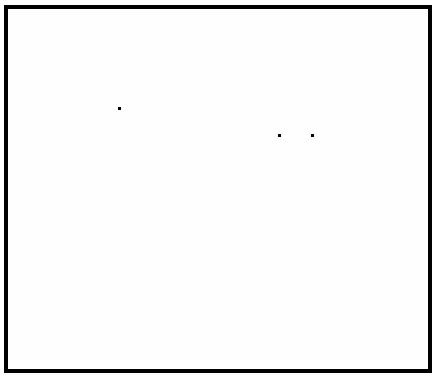
Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X(0) = P**
Boundary Conditions:	Fixed type, u_{ij} = -1 for all virtual cells, denoted by [U] = -1
Output:	**Y(t)⇒Y(∞)** = Binary image showing the junctions of a skeleton.
Template robustness:	ρ = 0.15 .

Remark:
A black pixel is considered to be a junction if it has at least 3 black neighbors.

II. Example: image name: junction.bmp, image size: 140x120; template name: junction.tem .



input



output

JunctionExtractor1: Finding the intersection points of thin (one-pixel thick) lines from two binary images

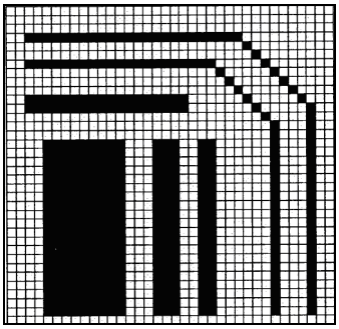
$$\mathbf{A} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ -0.5 & 3 & -0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ -0.5 & 3 & -0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix} \quad z = \boxed{-8.5}$$

I. Global Task

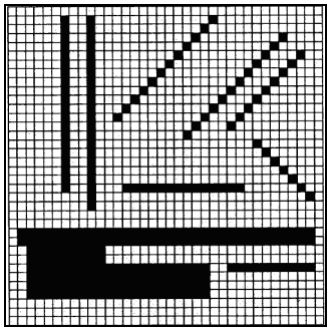
Given:	*two* static binary images **P1** and **P2** containing thin (one-pixel thick) lines or curves, among other (compact) objects
Input:	**U(t) = P1**
Initial State:	**X**(0) **= P2**
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = Binary image containing all the intersection points between the thin lines contained in the binary images **P1** and **P2**

Remarks: *The two binary images can be interchanged, i.e. we can apply* **P2** *at the input and load* **P1** *as initial state. The feedback and control templates are identical. Even if other (compact) objects are present in the two images, their overlapping is not detected, except intersection points of thin lines.*

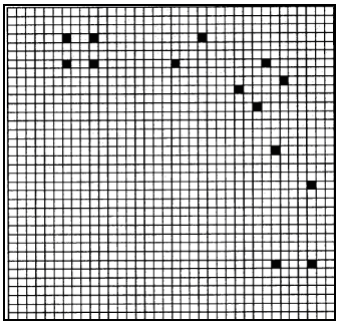
II. Example: image size: 36x36.



Initial state



Intermediate state



Output

LocalConcavePlaceDetector: ***Local concave place detector*** **[11]**

Old names: LCP

$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 2 \\ 1 & -2 & 1 \end{bmatrix}$ $z = \boxed{-7}$

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

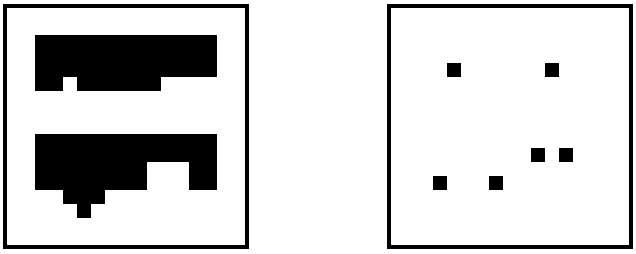
Output: **Y(t)⇒Y(∞)** = Binary image showing the local concave places of **P**.

Template robustness: $\rho = 0.24$.

Remark:

Local concave places of the image are pixels having no southern neighbor, but both eastern, western and either a south-western or (permissive) a south-eastern one.

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



input output

LE7pixelVerticalLineRemover: ***Deletes vertical lines not longer than 7 pixels*** **[10]**

Old names: LINCUT7V, CUT7V

A =

0	0	1	0	0
0	0	0.5	0	0
0	0	2	0	0
0	0	0.5	0	0
0	0	1	0	0

B =

0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

z = -5.5

I. Global Task

Given: static binary image **P**

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

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

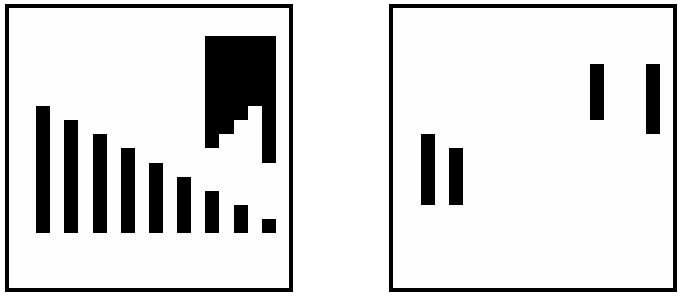
Boundary Conditions: Fixed type, u_{ij} = -1, y_{ij} = -1 for all virtual cells, denoted by [U]=[Y]= -1

Output: **Y(t)⇒Y(∞)** = Binary image where black pixels identify the vertical lines with a length of 8 or more pixels in **P**.

Remark:

The template deletes even those parts of the objects that could be interpreted as vertical lines not longer than 7 pixels.

II. Example: image name: lincut7v.bmp, image size: 20x20; template name: lincut7v.tem



input output

GrayscaleLineDetector: ***Grayscale line detector template***

Old names: LINE3060

A =

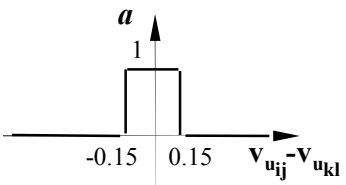
0	0	0
0	1.5	0
0	0	0

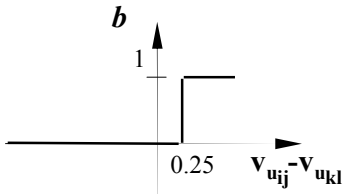
B =

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**.

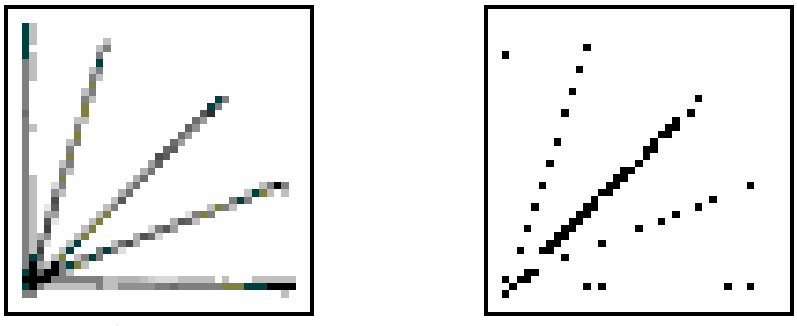
Remark:

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

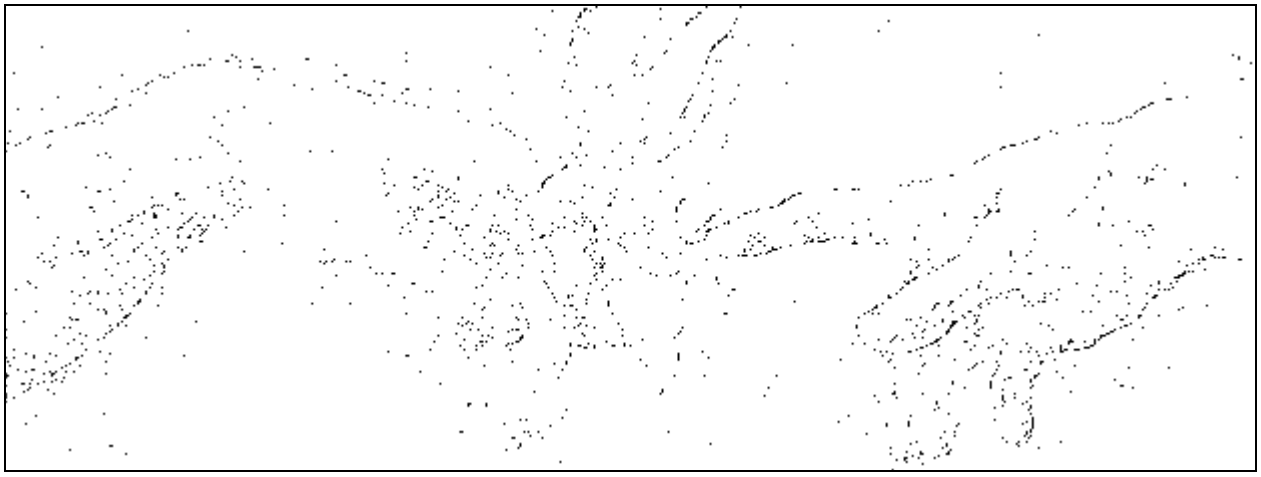
Example 1 (simple): image name: line3060.bmp, image size: 41x42; template name: line3060.tem .



input output

Example 2 (complex): image name: michelan.bmp, image size: 625x400; template name: line3060.tem .

input



output

LE3pixelLineDetector: ***Lines-not-longer-than-3-pixels-detector template*** **[13]**

Old names: LINEXTR3, LGTHTUNE

A =

0	0	0	0	0
0	0.3	0.3	0.3	0
0	0.3	3	0.3	0
0	0.3	0.3	0.3	0
0	0	0	0	0

B =

-1	0	-1	0	-1
0	-1	-1	-1	0
-1	-1	4	-1	-1
0	-1	-1	-1	0
-1	0	-1	0	-1

z = -2

I. Global Task

Given: static binary image **P** where the background is set to 0

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

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

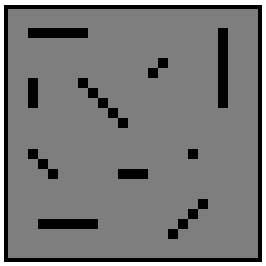
Boundary Conditions: Fixed type, $u_{ij} = 0$, $y_{ij} = 0$ for all virtual cells, denoted by [U]=[Y]=0

Output: **Y(t)⇒Y(∞)** = Binary image representing only lines not longer than 3 pixels in **P**.

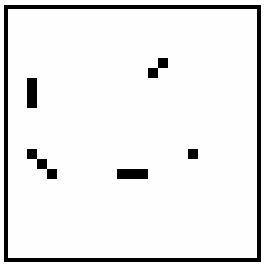
Remark:

The image **P** should contain only 1 pixel wide non-crossing lines. Otherwise the template may not work properly.

II. Example: image name: linextr3.bmp, image size: 25x25; template name: linextr3.tem .



input



output

PixelSearch: ***Pixel search in a given range*** **[72]**

A =

0	0	0
0	2	0
0	0	0

B=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0

z = -1

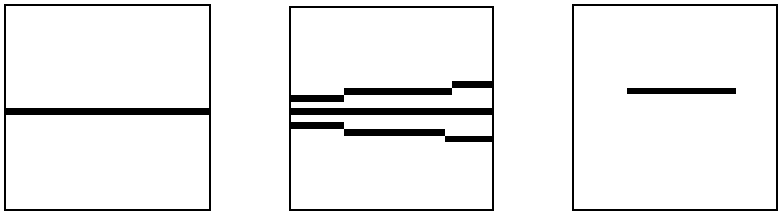
I. Global Task

Given:	static binary image **P, Q**
Input:	**U(t) = P** the reference image
Initial State:	**X(0) = Q** pixel whose distance is measured to the reference
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = Binary image representing the pixels being at the specified distance from the reference.

Remark:

This operation keeps those pixels of the initial state where there is a black pixel on the input at the place of the nonzero element of **B**. Based on the above example arbitrary pixel search operations can be implemented by extending the applied template to *NxN*. Although the operation requires *NxN* templates (7x7 in the example), members of this template class can be decomposed into a sequence of 3x3 linear templates (see [73]).

II. Example: image names: input_reference.bmp, initial_available.bmp; image size: 30x30; template name: pixelsearch.tem .



input initial state output

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

Old names: 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

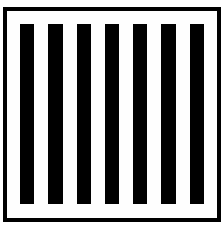
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

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

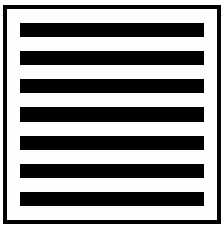
Initial State: $\mathbf{X(0) = P_2}$

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.

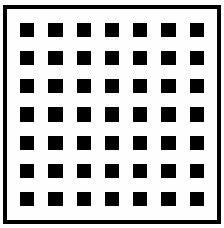
II. Example: image names: logic01.bmp, logic02.bmp; image size: 44x44; template name: logand.tem .



input



initial state



output

LogicDifference1: ***Logic Difference and Relative Set Complement ($P_2 \setminus P_1 = P_2 - P_1$) template* [7]**

Old names: LOGDIF, PA-PB

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{-1}$$

I. Global Task

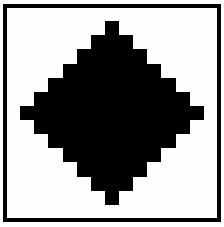
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

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

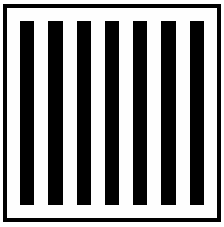
Initial State: $\mathbf{X}(0) = \mathbf{P_2}$

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Binary image representing the set-theoretic, or logic complement of $\mathbf{P_2}$ relative to $\mathbf{P_1}$. In set-theoretic or logic notation, $\mathbf{Y(\infty) = P_2 \setminus P_1}$, or $\mathbf{Y(\infty) = P_2 - P_1}$, i.e., $\mathbf{P_2}$ minus $\mathbf{P_1}$.

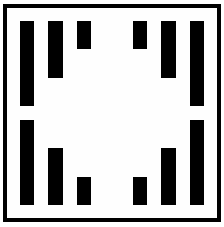
II. Example: image names: logic05.bmp, logic01.bmp; image size: 44x44; template name: logdif.tem .



input



initial state



output

LogicNOTOperation: ***Logic NOT and Set Complementation ($P \rightarrow \overline{P} = P^c$) template***

Old names: LogicNOT, LOGNOT, INV

A = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | -2 | 0 |
| 0 | 0 | 0 |

z = | 0 |
| --- |

I. Global Task

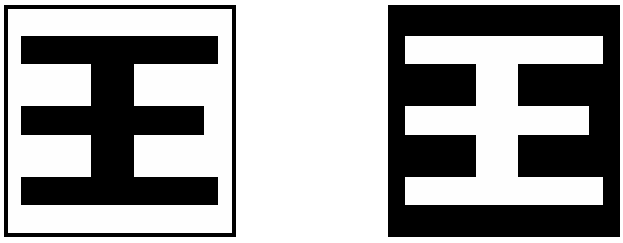
Given: static binary image **P**

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

Initial State: **X(0) =** Arbitrary or as a default X(0)=0

Output: **$Y(t) \Rightarrow Y(\infty)$** = Binary image where each black pixel in **P** becomes white, and vice versa. In set-theoretic or logic notation: **$Y(\infty) = P^c = \overline{P}$,** where the bar denotes the "Complement" or "Negation" operator.

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



input output

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

Old names: LogicOR, LOGOR, OR

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 2 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$z =$ | 1 |
|---|

I. Global Task

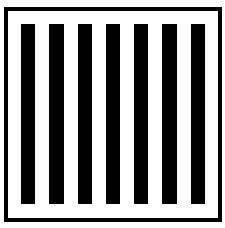
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

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

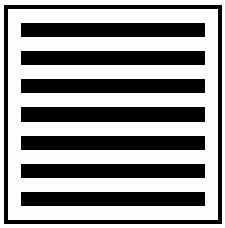
Initial State: $\mathbf{X}(0) = \mathbf{P_2}$

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.

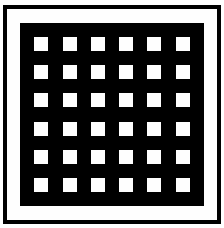
II. Example: image names: logic01.bmp, logic02.bmp; image size: 44x44; template name: logor.tem .



input



initial state



output

LogicORwithNOT: ***Logic "OR" function of the initial state and logic "NOT" function of the input* [24]**

Old names: LOGORN, INV-OR

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{1}$$

I. Global Task

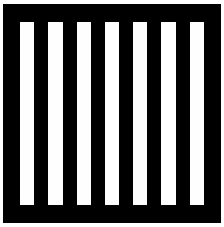
Given: two static binary images $\mathbf{P_1}$ and $\mathbf{P_2}$

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

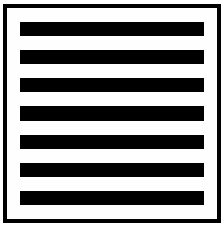
Initial State: $\mathbf{X}(0) = \mathbf{P_2}$

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = binary output of the logic operation **OR** between $\overline{P}_1$ and $\mathbf{P_2}$. In logic notation, $\mathbf{Y}(\infty) = \overline{P}_1 \vee \mathbf{P_2}$, where $\vee$ denotes the "disjunction" operator.

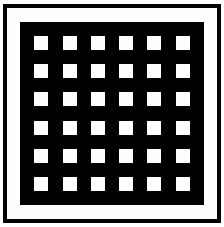
II. Example: image names: logic06.bmp, logic02.bmp; image size: 44x44; template name: logorn.tem .



input



initial state



output

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

Old names: PATCHMAK

$\mathbf{A} =$

0	1	0
1	2	1
0	1	0

$\mathbf{B} =$

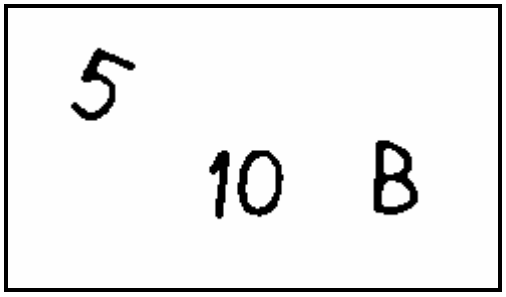
0	0	0
0	1	0
0	0	0

$z =$ 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** $\rightarrow \infty$ all pixels will be driven to black.

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



input

output

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

Old names: SMKILLER

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

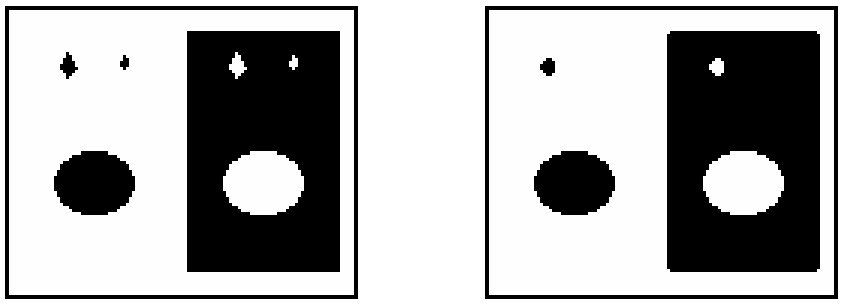
I. Global Task

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

BipolarWave: ***Generates black and white waves*** **[52]**

$$\mathbf{A} = \begin{bmatrix} 0.3 & 0.3 & 0.3 \\ 0.3 & 0.8 & 0.3 \\ 0.3 & 0.3 & 0.3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \begin{bmatrix} 0 \end{bmatrix}$$

I. Global Task

Given: image **P** containing three gray levels: +1, 0, -1 (black, gray, white)

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

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

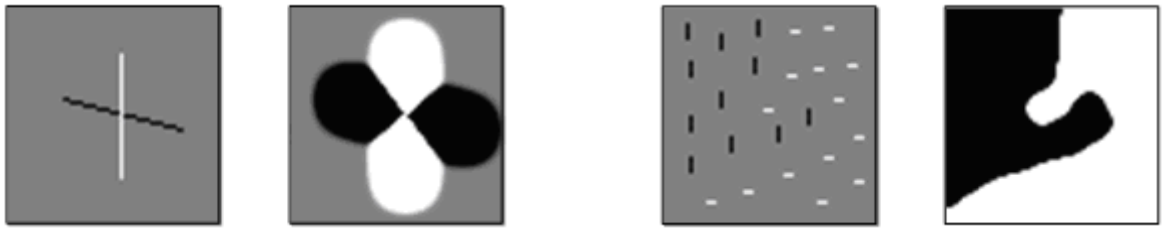
Boundary Conditions: Zero-flux boundary condition (duplicate)

Output: **Y(t)⇒Y(∞)** = Black and white areas, the boundary of which is located at positions where the waves collided.

Remark:

The wave starts from black and white pixels and propagates on cells which have zero state (gray color). The final image will contain black and white areas.

II. Example: image size: 64x64; template name: bipolar.tem .



input output input output

1.4. TEXTURE SEGMENTATION AND DETECTION

5x5TextureSegmentation1: Segmentation of four textures by a 5*5 template **[17]**

Old names: TX_HCLC

A =

-3.44	0.86	-1.64	-0.16	-1.02
-1.09	0.16	-2.19	-3.2	3.51
2.50	1.56	3.91	2.66	2.42
0.55	2.89	-0.62	0.47	3.67
-1.80	-0.55	2.50	-0.23	2.34

B =

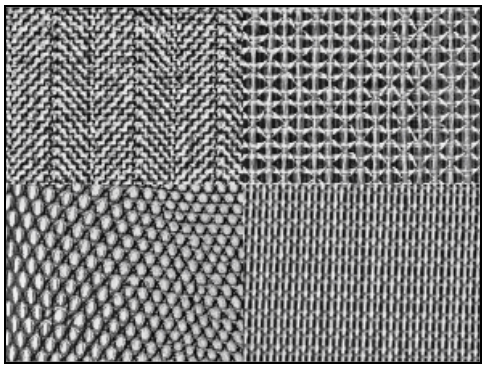
-2.19	-0.23	0.16	-0.63	-0.78
1.64	2.27	-3.2	1.09	2.03
0.08	0.55	0.86	3.52	0.08
0.39	-3.83	-3.12	-2.34	-2.11
0.78	-2.66	-1.17	-1.41	1.02

z = 3.28

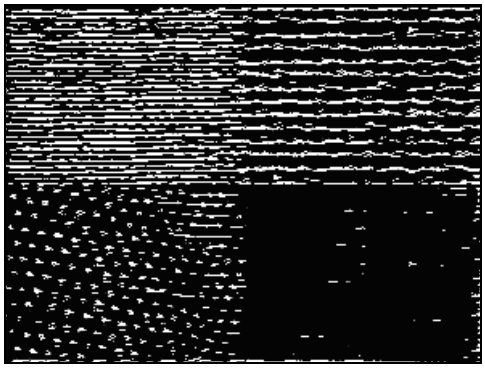
I. Global Task

Given:	static grayscale image **P** representing four textures (herringbone clothes, cloth, lizard skin, cloth) having the same flat grayscale histograms with the same gray average value
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.

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



input



output

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

Old names: 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**

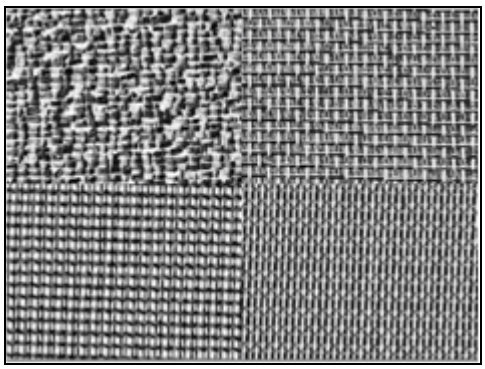
Initial State: **X(0) = P**

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

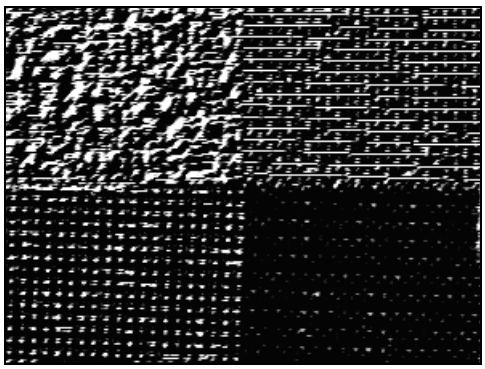
Output: **Y(t)⇒Y(T)** = Nearly binary image representing four patterns that differ in average gray-levels.

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

5x5TextureSegmentation2: Segmentation of four textures by a 5*5 template **[17]**

Old names: TX_RACC5

A =

4.21	-1.56	1.56	3.36	0.62
-2.89	4.53	-0.23	3.12	-2.89
2.65	2.18	-4.68	-3.43	-2.81
3.98	1.56	-1.17	-3.12	-3.20
-3.75	-2.18	3.28	2.19	-0.62

B =

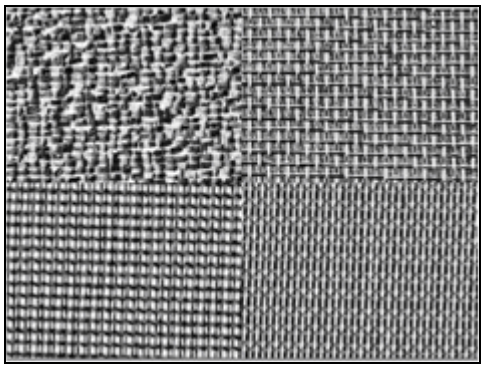
4.06	-5	0.39	2.11	-1.87
3.90	0.31	-1.95	4.84	-0.31
0	-4.06	0.93	-0.31	0.46
-0.62	-5	2.34	0.62	-1.87
3.59	-0.93	0.15	2.81	-1.87

z = -5

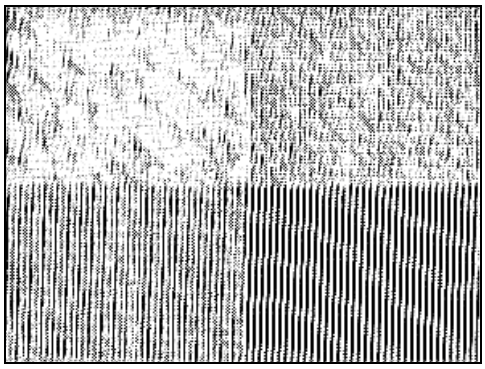
I. Global Task

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.

II. Example: image name: tx_racc.bmp, image size: 296x222; template name: tx_racc5.tem .



input



output

TEXTURE DETECTION

TextureDetector1 (T1_RACC3)

A = | 2.27 | 1.80 | 3.36 |
| -0.70 | -4.45 | 1.41 |
| 3.20 | 3.98 | -0.31 |

B = | -3.91 | 1.25 | 3.05 |
| 0.86 | -3.05 | 3.36 |
| 1.72 | -0.63 | -4.61 |

z = -1.64

TextureDetector2 (T2_RACC3)

A = | 1.56 | 4.38 | 2.42 |
| 4.69 | -3.13 | 1.41 |
| 2.19 | -5 | 0.86 |

B = | -2.81 | 2.42 | -3.75 |
| -5 | -0.39 | -5 |
| 3.67 | 4.22 | 3.13 |

z = -3.20

TextureDetector3 (T3_RACC3)

A = | 1.64 | -1.02 | 1.33 |
| 1.88 | -4.61 | 2.89 |
| 3.28 | 2.03 | 3.75 |

B = | -3.91 | -2.66 | -3.13 |
| 0.94 | 1.48 | -3.13 |
| 1.33 | 0.55 | 2.34 |

z = -2.42

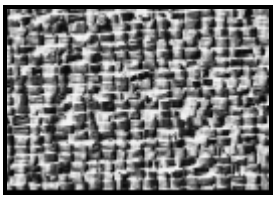
TextureDetector4 (T4_RACC3)

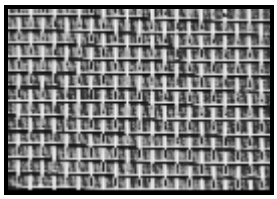
A = | 3.13 | 4.30 | 2.19 |
| -2.81 | 3.13 | 0.16 |
| 1.88 | 4.92 | 4.53 |

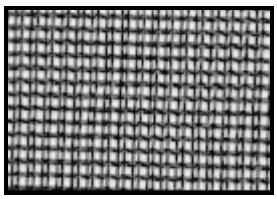
B = | -3.52 | 4.38 | -5 |
| -0.94 | -3.05 | -3.67 |
| 1.41 | -0.63 | -4.38 |

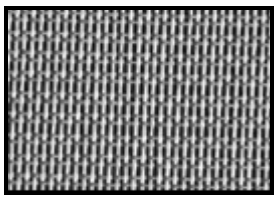
z = -2.42

Textures being detected by these templates


TextureDetector1


TextureDetector2


TextureDetector3


TextureDetector4

I. Global Task

Given: static grayscale image **P** representing textures having the same flat grayscale histograms. One of them is identical to a texture shown above.

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

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

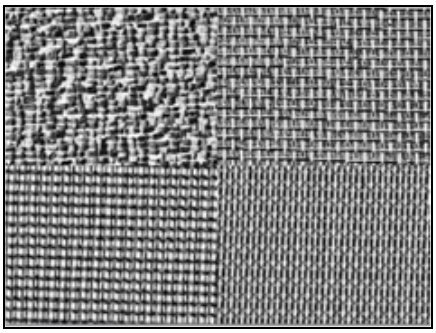
Boundary Conditions: Fixed type, $u_{ij} = 0$, $y_{ij} = 0$ for all virtual cells, denoted by [U]=[Y]=0

Output: **Y(t)⇒Y(T)** = Nearly binary image where the detected texture becomes darker than the others.

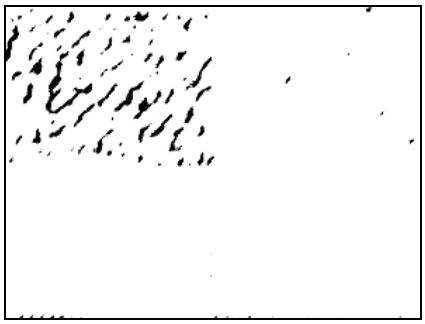
These templates can be used in a classification problem when the number of different examined textures is, for instance, more than 10 and the input textures have the same flat grayscale histograms.

II. Examples image name: tx_racc.bmp; image size: 296x222.

Example 1: Texture detection with the *TextureDetector1* (t1_racc3.tem) template.

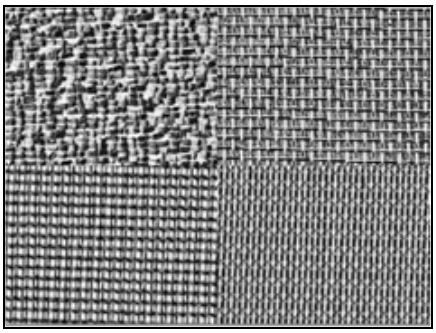


input

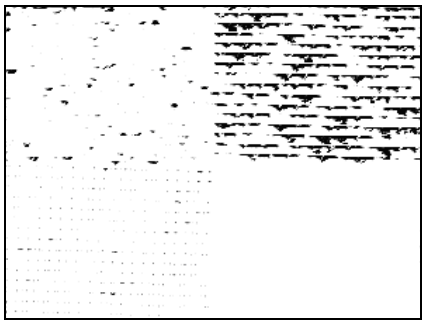


output

Example 2: Texture detection with the *TextureDetector2* (t2_racc3.tem) template.

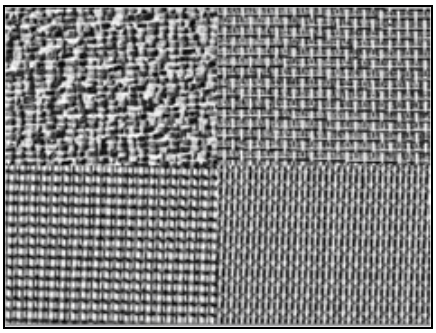


input

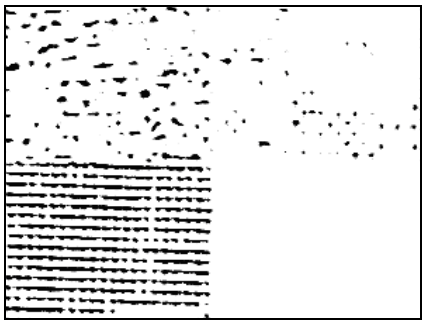


output

Example 3: Texture detection with the *TextureDetector3* (t3_racc3.tem) template.

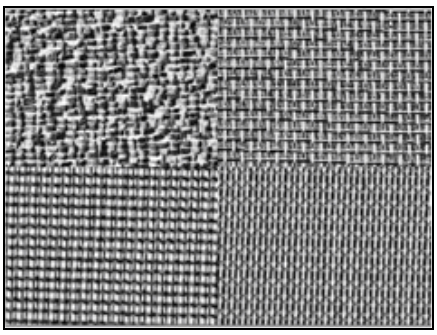


input

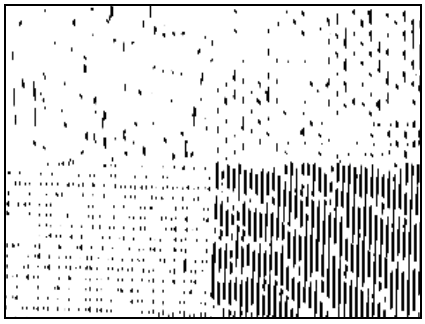


output

Example 4: Texture detection with the *TextureDetector4* (t4_racc3.tem) template.



input



output

1.5. MOTION

ImageDifferenceComputation: ***Logic difference between the initial state and the input pictures with noise filtering*** **[7]**

Old names: LogicDifference2, LOGDIFNF, PA-PB_F1

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0.25 | 0.25 | 0.25 |
|---|---|---|
| 0.25 | 2 | 0.25 |
| 0.25 | 0.25 | 0.25 |

$z =$ -4.75

$\mathbf{A}^{\tau} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}^{\tau} =$ | -0.25 | -0.25 | -0.25 |
|---|---|---|
| -0.25 | -2 | -0.25 |
| -0.25 | -0.25 | -0.25 |

$\tau = 10\ \tau_{CNN}$

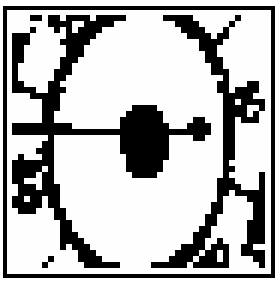
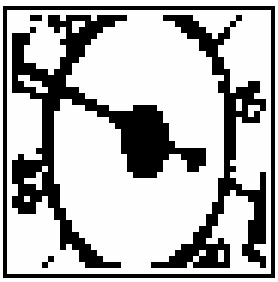
I. Global Task

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(T)** = Binary image where black pixels identify the moving parts of **P**.

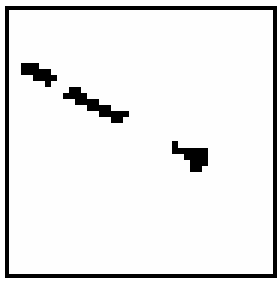
Remark:

Takes the logic difference of the delayed and actual input in continuous mode together with noise filtering. Moving parts of an image can be extracted.

II. Example: image names: logdnfi0.bmp, logdnfi1.bmp; image size: 44x44; template name: logdifnf.tem .

two consecutive steps of the input stream



output

MotionDetection: ***Direction and speed dependent motion detection*** **[12]**

Old names: MotionDetection1, MOTDEPEN, MOVEHOR

$\mathbf{A} =$ | -0.1 | -0.1 | -0.1 |
|---|---|---|
| -0.1 | 0 | -0.1 |
| -0.1 | -0.1 | -0.1 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1.5 | 0 |
| 0 | 0 | 0 |

$z =$ -2

$\mathbf{A}^{\tau} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

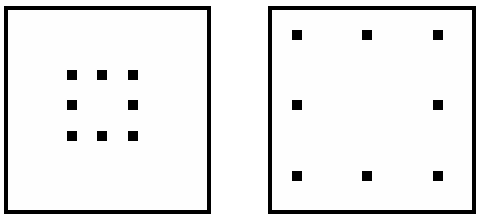
$\mathbf{B}^{\tau} =$ | 0 | 0 | 0 |
|---|---|---|
| 1.5 | 0 | 0 |
| 0 | 0 | 0 |

$\tau = 10\ \tau_{CNN}$

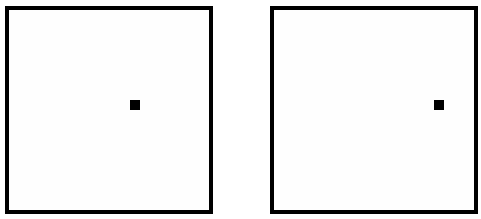
I. Global Task

Given:	static binary 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(T)** = Binary image representing only objects of **P** moving horizontally to the right with a speed of 1 pixel/delay time.

II. Example: image names: motdep1.bmp, motdep2.bmp; image size: 20x20; template name: motdepen.tem .



two consecutive steps of the input stream



corresponding outputs

***SpeedDetection: Direction independent motion detection* [7]**

Old names: MotionDetection2, MOTINDEP, MD_CONT

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 6 | 0 |
| 0 | 0 | 0 |

$z =$ -2

$\mathbf{A}^{\tau} =$ | 0.68 | 0.68 | 0.68 |
|---|---|---|
| 0.68 | 0.68 | 0.68 |
| 0.68 | 0.68 | 0.68 |

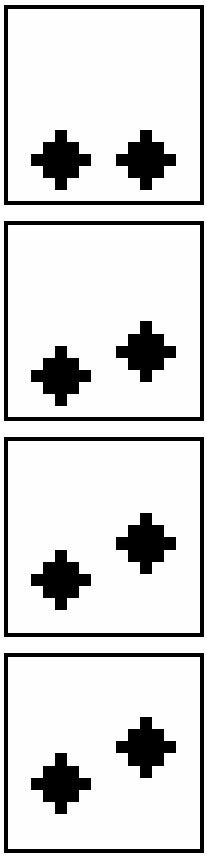
$\mathbf{B}^{\tau} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$\tau = 10\ \tau_{CNN}$

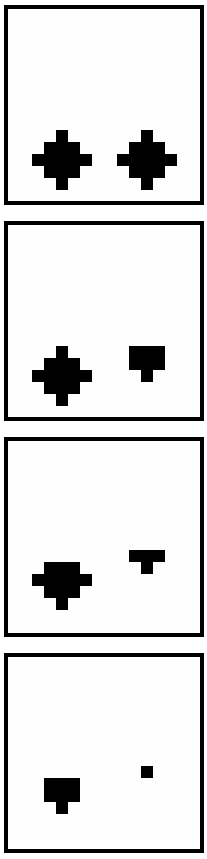
I. Global Task

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(T)** = Binary image representing only objects of **P** moving slower than 1 pixel/delay time in arbitrary directions.

II. Example: image names: motind1.bmp, motind2.bmp, motind3.bmp, motind4.bmp; image size: 16x16; template name: motindep.tem .



4 consecutive steps of the input stream

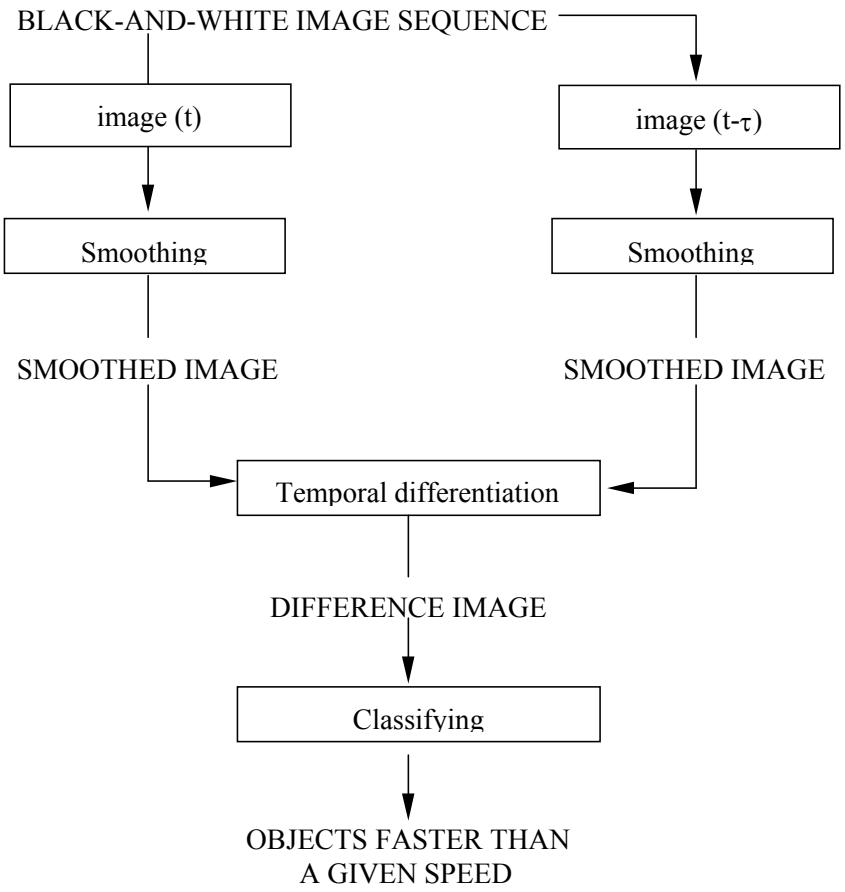


corresponding outputs

SPEED CLASSIFICATION

The algorithm is capable of classifying the speed of black-and-white objects moving parallel to the image plane. It extracts objects moving faster than a given speed determined by the current of the threshold template. Grayscale image sequences can be converted to black-and-white using the *Smoothing* template.

The flow-chart of the algorithm:



Smoothing:

$$\mathbf{B}_{smoothing} = \begin{bmatrix} 0.06 & 0.13 & 0.06 \\ 0.13 & 0.24 & 0.13 \\ 0.06 & 0.13 & 0.06 \end{bmatrix}$$

Temporal differentiation:

$$\mathbf{A}_{diff} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B}_{diff} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \square = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Classifying speed:

$$\mathbf{A}_{class} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B}_{class} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_{class} = \boxed{\textit{-threshold}}$$

Recalling objects:

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

0.3	0.3	0.3
0.3	4	0.3
0.3	0.3	0.3

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

0	0	0
0	5.1	0
0	0	0

Example: Input and output pictures. Image names: speed1.bmp, speed2.bmp; image size: 163x105.

PathTracing: ***Traces the path of moving objects on black-and-white images***

Old names: TRACE

$\mathbf{A}_{11} =$ | 0.3 | 0.3 | 0.3 |
|---|---|---|
| 0.3 | 4 | 0.3 |
| 0.3 | 0.3 | 0.3 |

$\mathbf{B}_{11} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 5.1 | 0 |
| 0 | 0 | 0 |

$z_1 =$ | 0 |
|---|

$\mathbf{A}_{22} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 2 | 0 |
| 0 | 0 | 0 |

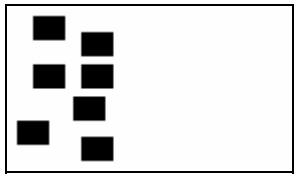
$\mathbf{A}_{21} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 3 | 0 |
| 0 | 0 | 0 |

$z_2 =$ | 2 |
|---|

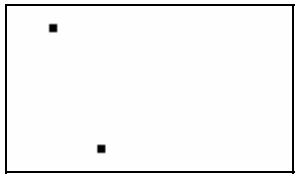
I. Global Task

Given:	a binary image sequence $\mathbf{P_1}$ and a binary image $\mathbf{P_2}$. $\mathbf{P_1}$ represents the objects to be traced, $\mathbf{P_2}$ consists of black pixels marking the objects to be traced.
Input:	$\mathbf{U_1(t) = P_1}$
Initial State:	$\mathbf{X_1}(0) = \mathbf{P_2}$
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	$\mathbf{Y_1(t) \Rightarrow Y(T)}$ = **B**inary image representing the actual position of the marked objects $\mathbf{Y_2(t) \Rightarrow Y(\infty)}$ = **B**inary image showing the whole path of the marked objects

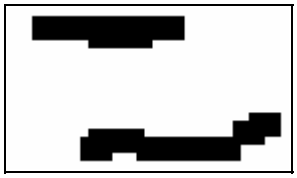
II. Example: image names: trace1.bmp, trace2.bmp; image size: 140x80; template name: trace.tem



starting snapshot of the input sequence



initial state

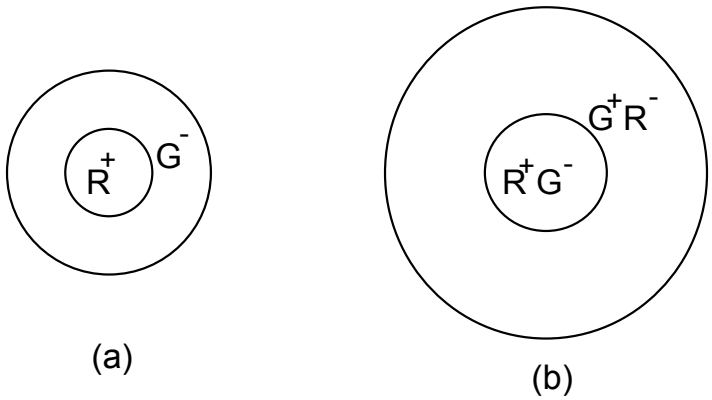


output of the 2. layer

1.6. COLOR

CNN MODELS OF SOME COLOR VISION PHENOMENA: SINGLE AND DOUBLE OPPONENCIES

In the retina, and the visual cortex, there are single and double color opponent cells [23]. Their receptive fields are as follows:



where (a) belongs to the single and (b) belongs to the double opponent cell. The template simulating the single opponent cell has two layers. The input of the first layer is the monochromatic red map, while the second layer gets the green map. The result appears on the second layer. The template is the following:

$\mathbf{B}_{12} =$

0	0	0
0	2	0
0	0	0

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

-0.25	-0.25	-0.25
-0.25	0	-0.25
-0.25	-0.25	-0.25

By swapping the layers we get the template generating the G^+R^- single opponents. The output of the R^+G^- and G^+R^- layers provide the input for the first and second layer of the double opponent structure, respectively. The output appears on the second layer. The template is as follows:

$\mathbf{B}_{12} =$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	2	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

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

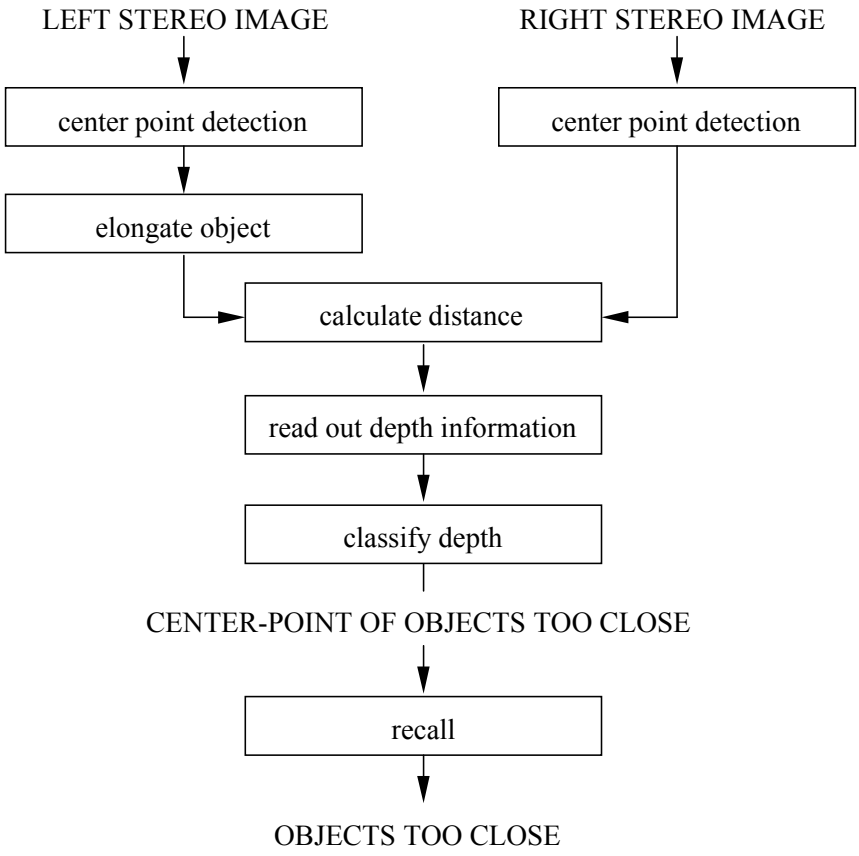
0.02	0.02	0.02	0.02	0.02	0.02	0.02
0.02	0	0	0	0	0	0.02
0.02	0	0	0	0	0	0.02
0.02	0	0	0	0	0	0.02
0.02	0	0	0	0	0	0.02
0.02	0	0	0	0	0	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.02

1.7. DEPTH

DEPTH CLASSIFICATION

The algorithm determines the depth of black-and-white objects based on a pair of stereo images. It determines whether an object is closer than a given distance or not. The first step of the algorithm is to reduce the objects in both input images to a single pixel; then the distance between corresponding points is calculated. The distance can be thresholded to determine whether the object is too close or not. As a first preprocessing step, grey-scale images can be converted into black-and-white using the *Smoothing* template.

The flow-chart of the algorithm:



Templates:

Elongate objects: add pixels to the top and bottom of each object (use the left image as input)

$\mathbf{A}_{\text{elongate}} =$

0	0	0
0	1	0
0	0	0

$\mathbf{B}_{\text{elongate}} =$

0	3	0
0	3	0
0	3	0

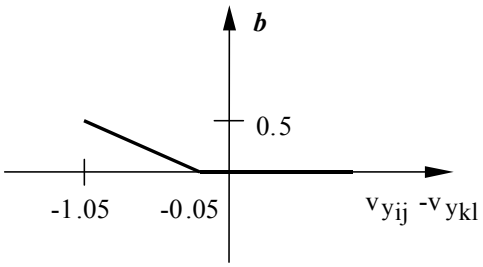
$z_{\text{elongate}} =$ 4.5

Calculate depth: (use the elongated left image as initial state)

$\mathbf{A}_{\text{distance}} =$

0	0	0
b	1	***b***
0	0	0

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



Read out depth: (use the right center points as a fixed state map)

$\mathbf{A}_{\text{read}} =$

0	0	0
0	1	0
0	0	0

$z_{\text{read}} =$ -2

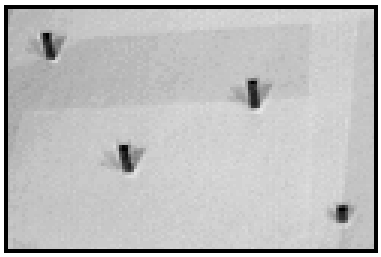
Classify depth:

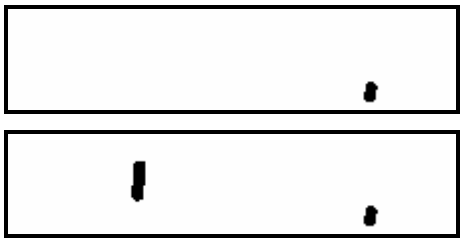
$\mathbf{A}_{\text{class}} =$

0	0	0
0	2	0
0	0	0

$z_{\text{class}} =$ *-threshold*

Example: image name: depth0.bmp; image size: 120x80.





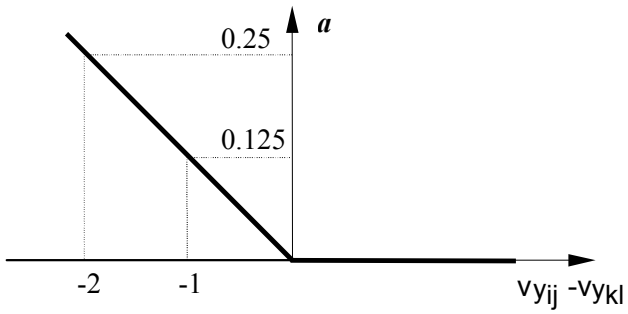
1.8. OPTIMIZATION

GlobalMaximumFinder: ***Finds the global maximum*** **[33]**

Old names: GLOBMAX

$$\mathbf{A} = \begin{bmatrix} a & a & a \\ a & 1 & a \\ a & a & a \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{0}$$

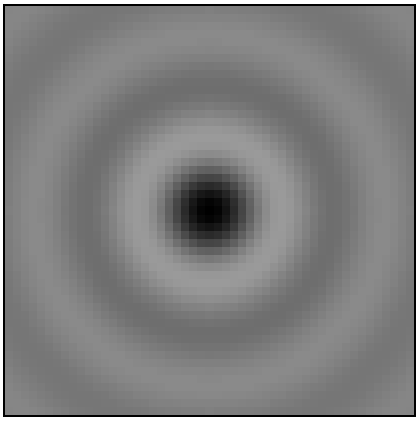
where ***a*** is defined by the following nonlinear function:



I. Global Task

Given:	static grayscale image **P**
Input:	**U(t)** = Arbitrary
Initial State:	**X**(0) = **P**
Boundary Conditions:	Zero-flux boundary condition (duplicate)
Output:	**Y(t)⇒Y(∞)** = An image where intensities of the pixels are identical and equal to the maximum intensity of pixels in **P**. In other words, the output is filled up with the global maximum of **P**.

II. Example: image name: globmax.bmp, image size: 51x51; template name: globmax.tem .



input

output

1.9. GAME OF LIFE AND COMBINATORICS

HistogramGeneration: ***Generates the one-dimensional histogram of a black-and-white image*** **[20]**

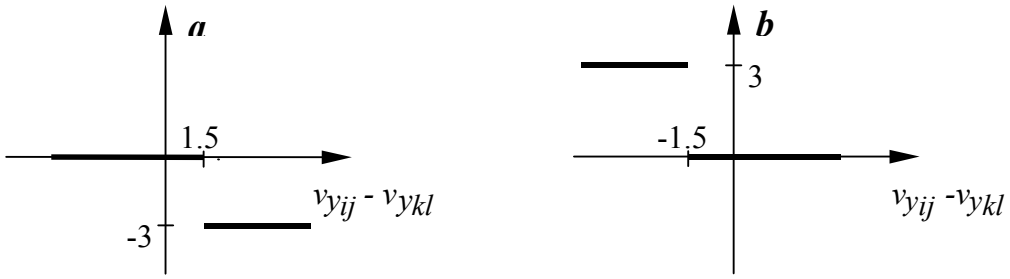
Old names: HistogramComputation, HISTOGR

A = | 0 | 0 | 0 |
| --- | --- | --- |
| ***a*** | 1 | ***b*** |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = | 0 |
| --- |

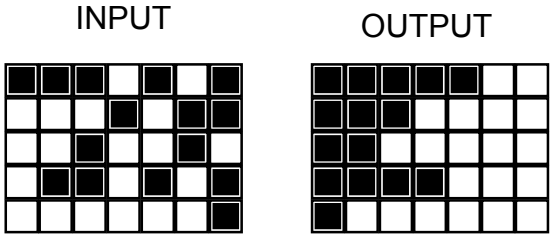
where ***a*** and ***b*** are defined by the following nonlinear functions:



I. Global Task

Given:	static binary image **P**
Input:	**U(t) =** Arbitrary
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 all black pixels are shifted to the left.

II. Example: image name: histogr.bmp, image size: 7x5; template name: histogr.tem .



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 |

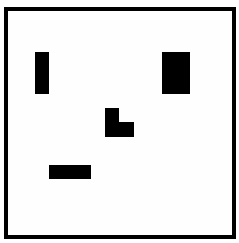
$\mathbf{B}_{21} =$ | -1 | -1 | -1 |
|---|---|---|
| -1 | -1 | -1 |
| -1 | -1 | -1 |

$z =$ | -4 |
|---|

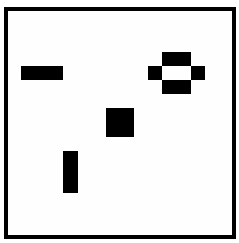
I. Global Task

Given:	static binary image **P**
Inputs:	$\mathbf{U_1(t) = P}$, $\mathbf{U_2(t)}$ = Arbitrary
Initial States:	$\mathbf{X_1}(0) = \mathbf{X_2}(0)$ = Arbitrary
Boundary Conditions:	Fixed type, u_{ij} = -1 for all virtual cells, denoted by [U]= -1
Outputs:	$\mathbf{Y_1(t)}$, $\mathbf{Y_2(t)} \Rightarrow \mathbf{Y_1(\infty)}$,$\mathbf{Y_2(\infty)}$ = 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



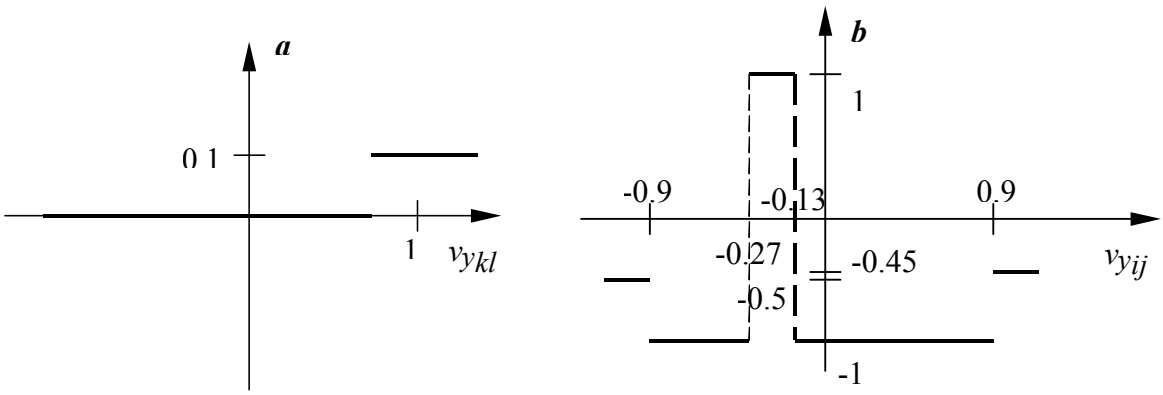
output

GameofLifeDTCNN1: ***Simulates the game of life on a single-layer DTCNN with piecewise-linear thresholding* [11]**

Old names: LIFE_1L

$$\mathbf{A} = \begin{bmatrix} a & a & a \\ a & b & a \\ a & a & a \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{0}$$

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



I. Global Task

Given:	static binary image **P**
Input:	**U(t) =** Arbitrary
Initial State:	**X**(0) **= P**
Boundary Conditions:	Fixed type, y_{ij} = 0 for all virtual cells, denoted by [Y]=0
Outputs:	$\mathbf{Y(t) \Rightarrow Y(2*t_1)}$, $\mathbf{t_1}$= 1,2,… Binary images representing the game of life. The new state appears after every second iteration.

Remark:

The DTCNN with piecewise-linear thresholding is a CNN approximated by the forward Euler integration form using time step 1.

II. Example

See the example of the *GameofLife1Step* template (template name: life_1l.tem).

GameofLifeDTCNN2: ***Simulates the game of life on a 3-layer DTCNN*** **[11]**

Old names: LIFE_DT

$\mathbf{A}_{31} =$

0	0	0
0	1	0
0	0	0

$\mathbf{A}_{13} =$

0.3	0.3	0.3
0.3	0.3	0.3
0.3	0.3	0.3

$z_1 =$ | 1 |

$\mathbf{A}_{32} =$

0	0	0
0	1	0
0	0	0

$\mathbf{A}_{23} =$

-0.6	-0.6	-0.6
-0.6	0	-0.6
-0.6	-0.6	-0.6

$z_2 =$ | -0.8 |

$z_3 =$ | -1.5 |

I. Global Task

Given:	static binary image **P**
Inputs:	$\mathbf{U_1(t) = U_2(t) = U_3(t)}$ = Arbitrary
Initial States:	$\mathbf{X_1}(0) = \mathbf{X_2}(0) = \mathbf{X_3}(0) = \mathbf{P}$
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Outputs:	$\mathbf{Y(t) \Rightarrow Y_3(3,5,7,9,\ldots)}$ = Binary images representing the game of life.

II. Example

See the example of the *GameofLife1Step* template (template name: life_dt.tem).

***MajorityVoteTaker: Majority vote-taker* [20]**

Old names: MAJVOT

The goal of the one dimensional majority vote-taker template is to decide whether a row of an input image contains more black or white pixels, or their number is equal. The effect is realized in two phases. The first template (setting the initial state to 0) gives rise to an image, where the sign of the rightmost pixel corresponds to the dominant color. Namely, it is positive, if there are more black pixels than white ones; it is negative in the opposite case, and is 0 in the case of equality. By using the second template this information can be extracted, which drives the whole network into black or white, depending on the dominant color, or leaves the rightmost pixel unchanged otherwise. The method can easily be extended to two or even three dimensions.

First template

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0.05 | 0 |
| 0 | 0 | 0 |

$z =$ | 0 |
|---|

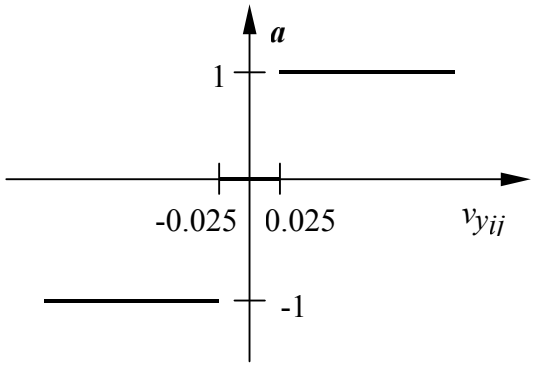
Second template

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | ***a*** | 2 |
| 0 | 0 | 0 |

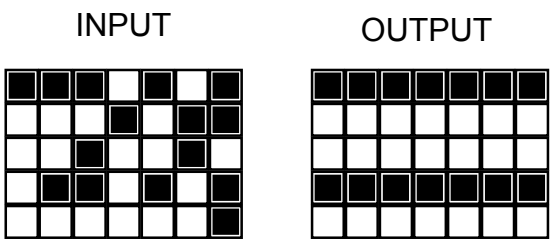
$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$z =$ | 0 |
|---|

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



Example: image name: histogr.bmp, image size: 7x5; template names: majvot1.tem, majvot2.tem.



ParityCounting1: ***Determines the parity of a row of the input image*** **[20]**

Old names: PARITY

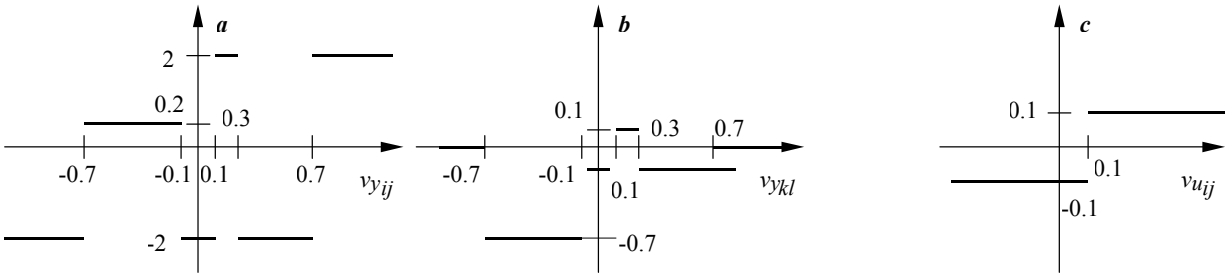
The template determines whether the number of black pixels in a row is even or odd. As a result, the leftmost pixel in the output image corresponds to the parity of the row, namely, black represents odd, while white means even parity. It is also true that each pixel codes the parity of the pixels right to it, together with the pixel itself. Naturally, the parity of a column or a diagonal can be counted in the same manner. The parity of an array can also be determined if columnwise parity is counted on the result of the rowwise parity operation. The initial state should be set to -0.5.

A = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | ***a*** | ***b*** |
| 0 | 0 | 0 |

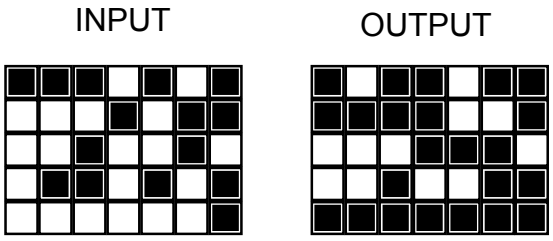
B = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | ***c*** | 0 |
| 0 | 0 | 0 |

z = | 0 |
| --- |

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



Example: image name: histogr.bmp, image size: 7x5; template name: parity1.tem .



ParityCounting2: ***Computes the parity of rows in a black-and-white image***

A = | 0 | 0 | 0 |
    | 0 | 1 | 0 |
    | 0 | 0 | 0 |

D = | 0 | 0 | 0 |
       | ***d*** | 0 | 0 |
       | 0 | 0 | 0 |

z = | 0 |

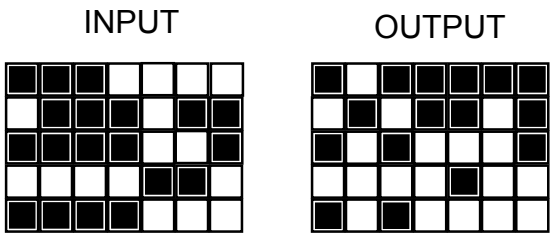
where $\boldsymbol{d} = - v_{u_{ij}} v_{y_{kl}}$.

I. Global Task

Given:	static binary image **P**
Input:	**U(t) = P**
Initial State:	**X**(0) **= P**
Boundary Conditions:	Fixed type, $y_{ij} = -1$ for all virtual cells, denoted by [Y]=-1
Output:	**Y(t)⇒Y(∞)** = Binary image representing the parity of rows in P

Remark:
A particular pixel in the output is black if an odd number of black pixels can be found at the left of the particular pixel in the input (including the position of the pixel itself).

II. Example: image name: parity2.bmp, image size: 7x5; template name: parity2.tem .



1-DArraySorting: ***Sorts a one dimensional array*** **[20]**

Old names: SORTING

A one-dimensional array of n values in the [-1,+1] interval can be sorted in descending order in n steps with the following time- and space-varying template. In each odd step, the templates should be applied in the $\mathbf{A}_R\mathbf{A}_L\mathbf{A}_R\mathbf{A}_L\mathbf{A}_R\mathbf{A}_L$ ... pattern, while in each even step in the $\mathbf{A}_L\mathbf{A}_R\mathbf{A}_L\mathbf{A}_R\mathbf{A}_L\mathbf{A}_R$ ... pattern. To suppress side effects, the left and right boundaries should be set to +1 and -1, respectively.

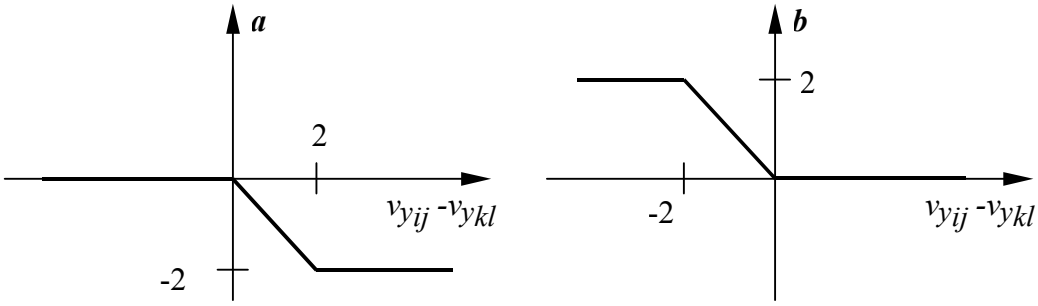
$\mathbf{A}_L =$

0	0	0
a	1	0
0	0	0

$\mathbf{A}_R =$

0	0	0
0	1	***b***
0	0	0

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



Example: image name: sorting.bmp; image size: 12x1.

Input

Output

1.10. PATTERN FORMATION

SPATIO-TEMPORAL PATTERN FORMATION IN TWO-LAYER OSCILLATORY CNN

Spatio-temporal pattern formation in two-layer oscillatory CNN is studied in [56] and showed e.g. that some exotic types of spiral waves exist on this type of network. As an example, spiral waves might consist of not only two types of motif (black and white patches) but also, for instance, checkerboard patterns. These three types of motifs propagate like spiral waves and transform continuously into each other.

I. Global Task

Task: Generate oscillatory Turing patterns

Given:

Initial States:	$\mathbf{X_1}(0)$, $\mathbf{X_2}(0)$ = small signal random pattern, otherwise arbitrary
Boundary Conditions:	Zero-flux or periodic boundary condition
Outputs:	$\mathbf{Y_1(t)}$, $\mathbf{Y_2(t)}$ spatio-temporal oscillatory Turing patterns.

II. CNN architecture

Consider the dynamics of a two-layer autonomous CNN:

$$\left.\begin{aligned}\frac{dx_{1,ij}}{dt} &= -x_{1,ij} + \mathbf{A}_{1,1} * \mathbf{y}_1 + \beta_1 y_{2,ij} \\ \frac{dx_{2,ij}}{dt} &= -x_{2,ij} + \mathbf{A}_{2,2} * \mathbf{y}_2 + \beta_2 y_{1,ij}\end{aligned}\right\},$$ where β_1 and β_2 are the coupling parameters between the two layers.

III. Templates

Turing pattern generating templates extended with coupling parameter (β_1 and β_2). As an example, template A_{11} generates cow patches while template A_{22} generates checker board pattern. Range of the coupling parameter is β = [0.5, 3.0].

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

1	0.1	1
0.1	-2	0.1
1	0.1	1

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

0	0	0
0	0	0
0	0	0

$z =$ 0 $\quad \beta_1 =$ β

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

1	-0.1	1
-0.1	-2	-0.1
1	-0.1	1

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

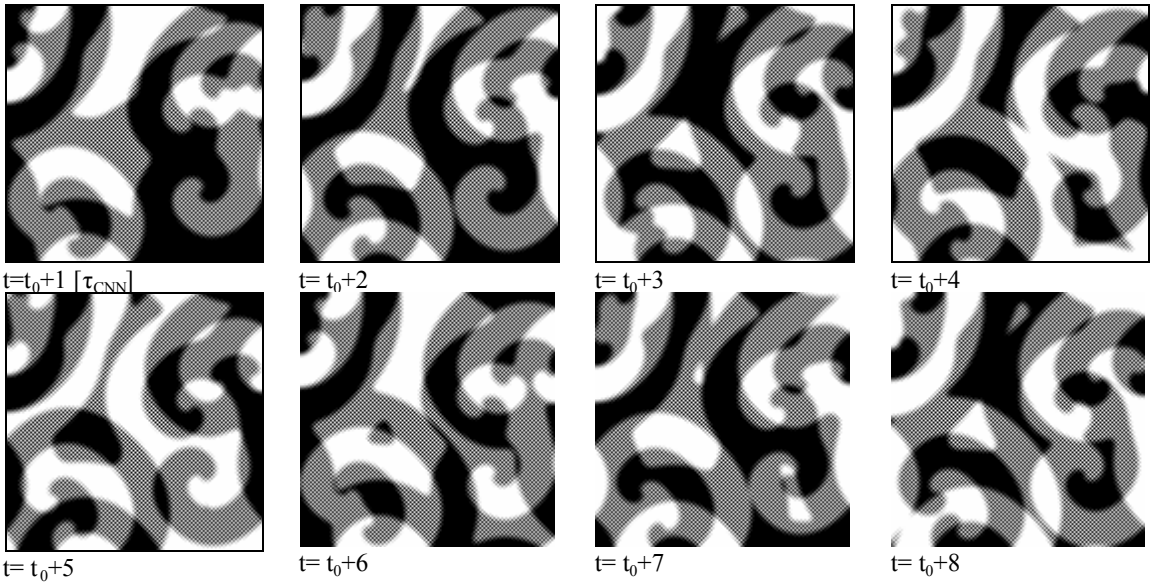
0	0	0
0	0	0
0	0	0

$z =$ 0 $\quad \beta_2 =$ $-\beta$

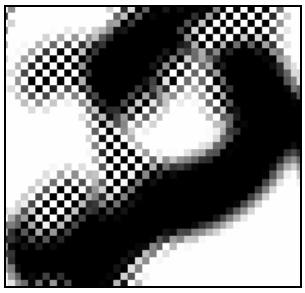
IV. Example

Image size = 128x128, $\beta_1 = -\beta_2 = 1$, and zero-flux boundary condition. The oscillation frequency is ~ 10 [τ_{CNN}].

Output of the 1st layer (2nd layer behaves in a very similar way):



A magnified part of the image makes visible the checkerboard patterns:



SPATIO-TEMPORAL PATTERNS OF AN ASYMMETRIC TEMPLATE CLASS

The template class analyzed in [57] produces novel spatio-temporal patterns that exhibit complex dynamics. The character of these propagating patterns depends on the self-feedback and on the sign of the coupling below the self-feedback template element.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ s & p & q \\ 0 & r & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{z} = \boxed{z}$$

I. Global Task

Given: static binary image **P**

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

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

Boundary Conditions: Zero-flux boundary condition (duplicate)

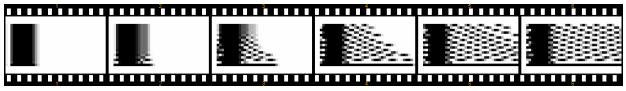
Output: Generated spatio-temporal pattern the structure of which depend on the sign of the extra template element *r*.

Remark:
The full range CNN model is used.

II. Examples

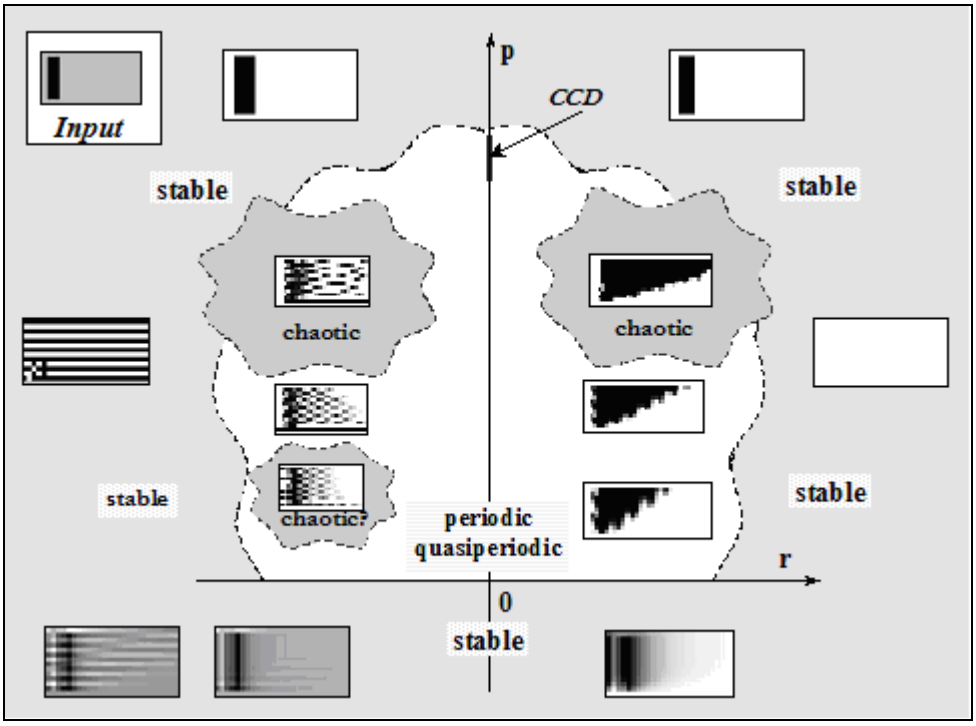
Examples are generated for the *sign-antisymmetric* case having $sq<0$ ($s = -q$).
Patterns generated depend on the sign of the extra template element *r*:

- if $r>0$ a pattern is formed which is solid inside, however its right border is oscillating.
- if $r<0$ a texture-like oscillating pattern is formed.

Coupling sign	*Propagating pattern*	*Snapshots*	*Time evolution*
positive: r > 0	solid inner part, oscillating border cells		
negative: r < 0	texture like oscillating pattern		

The effect of the central template element

If self-feedback is increased the generated pattern becomes more and more irregular and it can become chaotic. The *r-p* plane can be divided into *stable-periodic-chaotic* sub-regions as shown below.



The input & initial state is shown in the upper left corner. It is a three-pixel wide bar. The pictures in the different regions show few typical snapshots of outputs belonging to that region. The arrangement and size of the different regions gives only qualitative information.

1.11. OTHERS

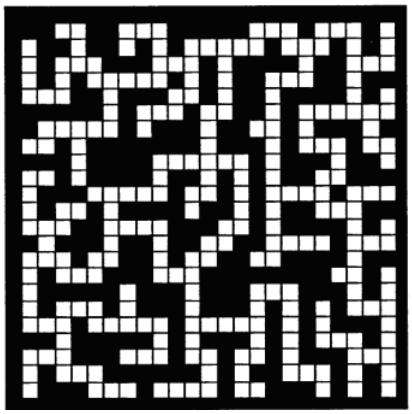
PathFinder: ***Finding all paths between two selected points through a labyrinth*** **[61]**

$$\mathbf{A} = \begin{bmatrix} 0.5 & 4 & 0.5 \\ 4 & 12 & 4 \\ 0.5 & 4 & 0.5 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{8}$$

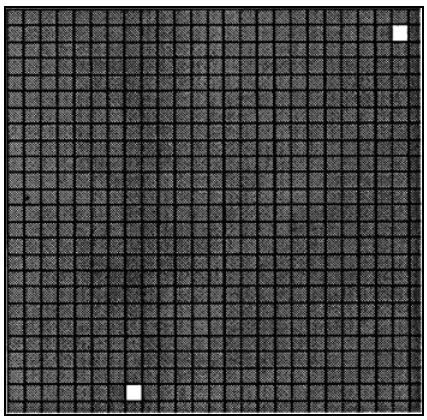
I. Global Task

Given:	static binary image **P** representing a labyrinth made of one-pixel thick white curves on a black background
Input:	**U(t) = 0**, except for two white pixels which mark the desired points in the labyrinth
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 containing all the paths connecting the marked points (made of white curves against a black background)

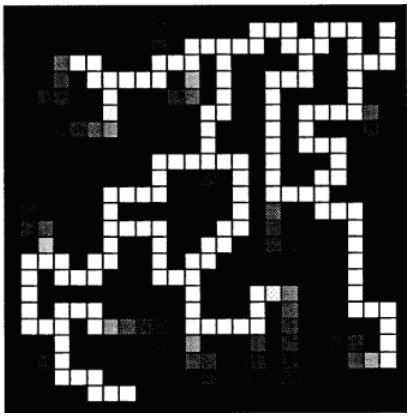
II. Example: image size: 25x25.



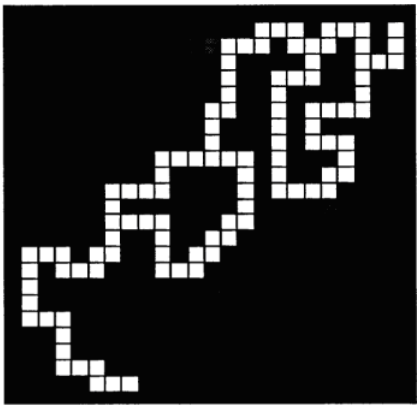
Initial state



Input



Intermediate result



Output

ImageInpainting: Interpolation-based image restoration [58]

Old names: NEL_AINTPOL3

$$\mathbf{A} = \begin{bmatrix} -0.05 & 0.3 & -0.05 \\ 0.3 & 0 & 0.3 \\ -0.05 & 0.3 & -0.05 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 0 & d & 0 \\ d & 0 & d \\ 0 & d & 0 \end{bmatrix}$$

where $d = \lambda sign(y_{ij} - x_{kl}); \lambda \in [-1, 0]$, with (**B**=0, z=0).

I. Global Task

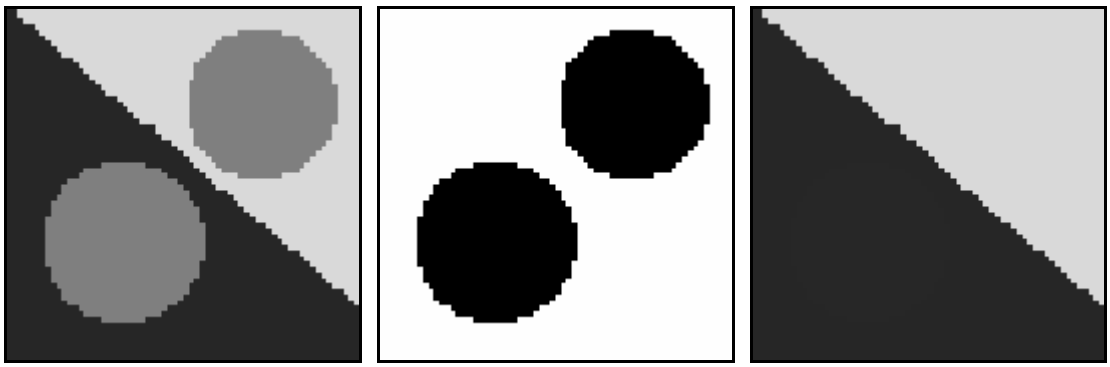
Given:	static grayscale image **P** (image to be restored = missing or damaged image) and static binary image **M**
Input:	**U(t) =** Arbitrary (in the examples we choose **U(t)= P**)
Initial State:	**X(0) = P**
Boundary Conditions:	Zero-flux boundary condition (duplicate)
Fixed State Mask:	$\mathbf{X_{fix}}$**=M.** It is necessary to use a mask image that does not change the elements of the image that are known at the beginning, but allow the computing of unknown elements. The existence of a mask image presumes that the user knows the positions of the elements that need to be computed.
Output:	**Y(t)⇒Y(∞)** = Grayscale image representing the restored missing or damage image.

Remark:

Image inpainting is an interpolation problem where an image with missing or damaged parts is restored.

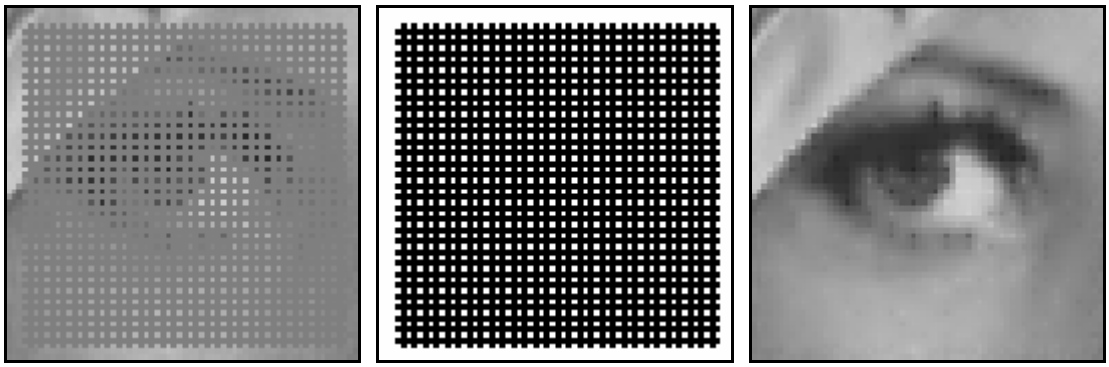
II. Examples

Example 1: image name: inphole.bmp, image size: 64x64; template name: nel_aintpol3.tem .



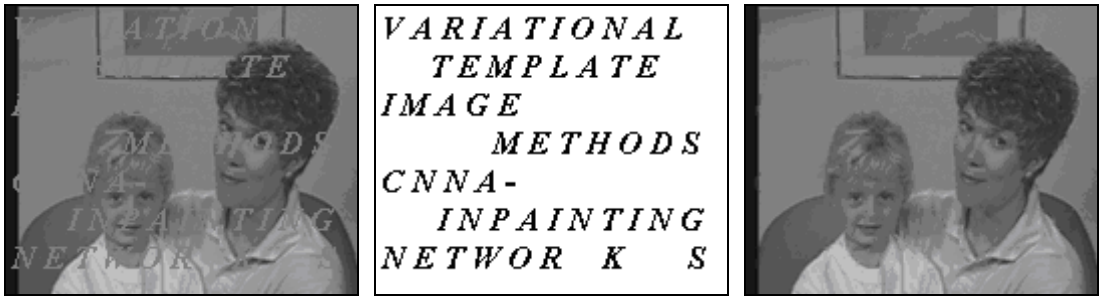
input mask output

Example 2: image name: inpeye.bmp, image size: 64x64; template name: nel_aintpol3.tem .



input mask output

Example 3: image name: inpaintig.bmp, template name: nel_aintpol3.tem .



input mask output

ImageDenoising: ***Image denoising based on the total variational (TV) model of Rudin-Osher-Fatemi [59, 60]***

A =

0	*a*	0
a	1	*a*
0	*a*	0

D =

0	0	0
0	*d*	0
0	0	0

where $a = \lambda sign\left(x_{ij} - x_{kl}\right)$ $\lambda \in [-1, 0]$ and $d = 2\alpha\left(x_{ij} - u_{ij}\right)$ $\alpha \in [0, 1]$, with (**B**=0, z=0).

I. Global Task

Given: static grayscale image **P** (image to be denoising)

Input: **U(t) =** Arbitrary (in the examples we choose **U(t)= P**)

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

Boundary Conditions: Zero-flux boundary condition (duplicate)

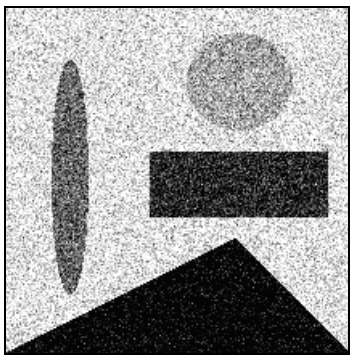
Output: **Y(t)⇒Y(∞)** = Grayscale image representing the denoising image.

Remark:

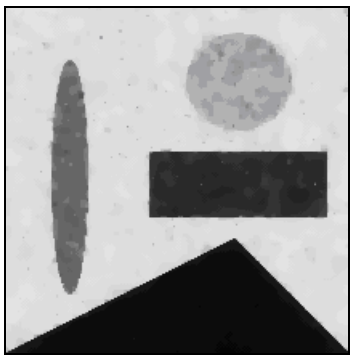
The noise is white Gaussian and additive.

II. Examples

Example 1: image name: osrufa5.bmp, image size: 214x216; templates name: osrufa2.tem.



input



output

Example 2: image name: cameraman10.bmp, image size: 256x256; templates name: osrufa.tem.

input

output

Orientation-SelectiveLinearFilter: ***IIR linear filter with orientation-selective low-pass (LP) frequency response, oriented at an angle*** φ ***with respect to an axis of the frequency plane*** **[62]**

Given the orientation angle φ, first the 3×3 *orientation matrix* O_φ is obtained:

$$O_\varphi = -\frac{1}{2}\left(\sin 2\varphi \cdot \begin{bmatrix} 0 & 1 & -1 \\ 1 & -2 & 1 \\ -1 & 1 & 0 \end{bmatrix} + \cos 2\varphi \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right)$$

then another two templates depending only on φ are obtained:

$$B_\varphi = O_0 - 0.4532 \cdot O'_\varphi + 0.0194 \cdot O_\varphi * O_\varphi$$

$$A_\varphi = O_0 + 0.0468 \cdot O'_\varphi + 0.0012 \cdot O_\varphi * O_\varphi$$

where "$*$" = matrix convolution; O_0 is a 5×5 zero matrix with the central element of value 1; the 5×5 matrix O'_φ is the 3×3 matrix O_φ bordered with zeros in order to be summed with O_0 and $O_\varphi * O_\varphi$. In the two templates A_φ, B_φ (5×5), the marginal elements are generally negligible and can be discarded, so A_φ, B_φ can be reduced to size 3×3 with a minimum error.

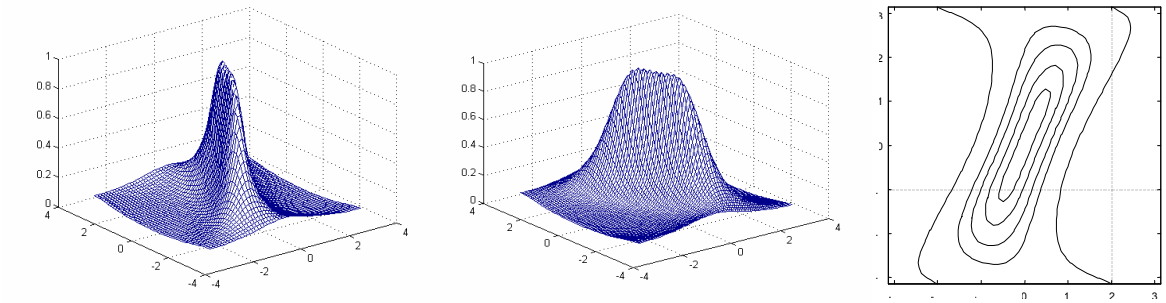
An LP oriented filter with the general spatial transfer function $H_\varphi(\omega_1,\omega_2) = \dfrac{B(\omega_1,\omega_2)}{A(\omega_1,\omega_2)}$ can be designed using the templates, with parameter $p<0$ specifying selectivity:

$$B(\omega_1,\omega_2) = A_\varphi(\omega_1,\omega_2)$$

$$A(\omega_1,\omega_2) = -(p+1)\cdot A_\varphi(\omega_1,\omega_2) + p \cdot B_\varphi(\omega_1,\omega_2)$$

Example: An LP oriented filter with $p_a = -37.6$ and $\varphi = \pi/8$ is realized with:

$$A = \begin{bmatrix} -0.1413 & 8.4406 & 5.8977 \\ -3.2964 & -23.6541 & -3.2964 \\ 5.8977 & 8.4406 & -0.1413 \end{bmatrix}; \quad B = \begin{bmatrix} 0.0005 & 0.0464 & 0.0331 \\ -0.0199 & -1.9623 & -0.0199 \\ 0.0331 & 0.0464 & 0.0005 \end{bmatrix}$$



Frequency response of a selective oriented LP filter ($p_a = -37.6$ and $\varphi = \pi/8$) viewed from two angles and constant level contours

Complex-Gabor: ***Filtering with a complex-valued Gabor-type filter*** **[53]**

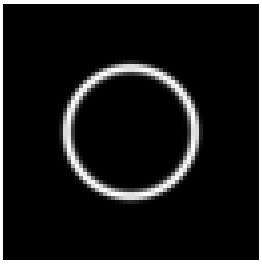
$$\mathbf{A} = \begin{bmatrix} 0 & e^{-j\Omega_y} & 0 \\ e^{j\Omega_x} & -\left(3+\lambda^2\right) & e^{-j\Omega_x} \\ 0 & e^{j\Omega_y} & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{0}$$

where Ω_x and Ω_y control the spatial frequency tuning of the filter and λ controls the bandwidth. Note that the off-center elements are complex valued $\left(j = \sqrt{-1}\right)$. The state is also assumed to be complex valued. Note that the center element of the template presented here differs from that presented in [53] by 1 because we assume the standard CNN equation here, whereas [53] used an equation without the resistive loss term.

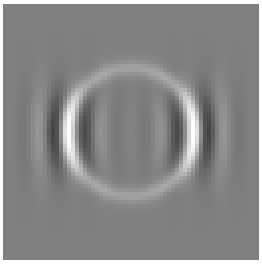
I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) = P**
Initial State:	**X**(0) **=** Arbitrary or as a default *X*(t)=0. Note that the state is assumed to be complex valued.
Boundary Conditions:	Fixed type, y_{ij} = 0 for all virtual cells, denoted by [Y]=0
Output:	**Y(t)⇒Y(∞)** = grayscale image representing the output of the Gabor-type filter. The real part of the output is the output of an even-symmetric Gabor-type filter. The imaginary part of the output is the output of an odd symmetric Gabor type filter.

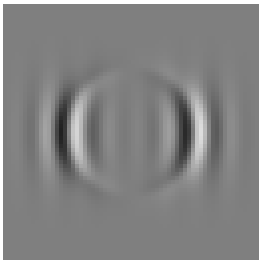
II. Example: image name: annulus.bmp, image size: 64x64; template name: cgabor.tem .



P (input)



Real(Y)



Imag(Y)

where $\lambda = 0.2$, $\Omega_x = 2\pi/8$, $\Omega_y = 0$.

Two-Layer Gabor: ***Two-layer template implementing even and odd Gabor-type filters***

$$\mathbf{A_{11}} = \begin{bmatrix} 0 & \cos(\Omega_y) & 0 \\ \cos(\Omega_x) & -(3+\lambda^2) & \cos(\Omega_x) \\ 0 & \cos(\Omega_y) & 0 \end{bmatrix} \quad \mathbf{B_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = \boxed{0}$$

$$\mathbf{A_{22}} = \begin{bmatrix} 0 & \cos(\Omega_y) & 0 \\ \cos(\Omega_x) & -(3+\lambda^2) & \cos(\Omega_x) \\ 0 & \cos(\Omega_y) & 0 \end{bmatrix} \quad \mathbf{B_2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_2 = \boxed{0}$$

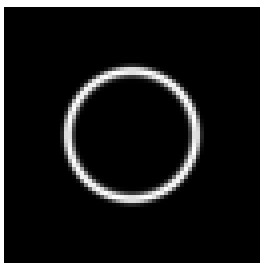
$$\mathbf{A_{12}} = \begin{bmatrix} 0 & \sin(\Omega_y) & 0 \\ -\sin(\Omega_x) & 0 & \sin(\Omega_x) \\ 0 & -\sin(\Omega_y) & 0 \end{bmatrix} \quad \mathbf{A_{21}} = \begin{bmatrix} 0 & -\sin(\Omega_y) & 0 \\ \sin(\Omega_x) & 0 & -\sin(\Omega_x) \\ 0 & \sin(\Omega_y) & 0 \end{bmatrix}$$

where Ω_x and Ω_y control the spatial frequency tuning of the filter and λ controls the bandwidth. This template is equivalent to the Complex-Gabor template, where we have separated the real and imaginary parts to two layers.

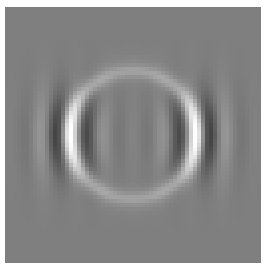
I. Global Task

Given:	static grayscale image **P**
Input:	**U(t) = P**
Initial State:	$\mathbf{X_1(0) = X_2(0) =}$ Arbitrary or as a default $X_1(t)= X_2(t)=0$.
Boundary Conditions:	Fixed type, $y_{ij} = 0$ for all virtual cells, denoted by [Y]=0
Output:	$\mathbf{Y_1(t) \Rightarrow Y_1(\infty)}$ = grayscale image representing the output of the even-symmetric Gabor-type filter.
	$\mathbf{Y_2(t) \Rightarrow Y_2(\infty)}$ = grayscale image representing the output of the odd-symmetric Gabor-type filter.

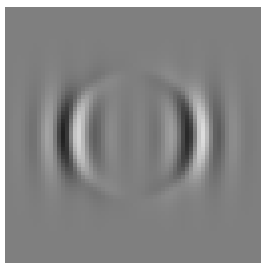
II. Example: image name: annulus.bmp, image size: 64x64; template name: cgabor.tem .



P (input)



Y_1



Y_2

where $\lambda = 0.2$, $\Omega_x = 2\pi / 8$, $\Omega_y = 0$.

LinearTemplateInverse: ***Inverse of a linear template operation using dense support of input pixels [55]***

A Linear Template to be inverted

$\mathbf{A} =$

0	0	0
0	0	0
0	0	0

$\mathbf{B} =$

-0.03	-0.10	-0.02
0	0.50	-0.20
-0.03	-0.10	-0.02

$z =$ 0

Example: image names: LenaS.bmp; image size: 128x128; template name: CS2.tem.

TEST INPUT

TEST OUTPUT

Old names: Linear Template Inverse (A_i=1-B; B_i=1-A; A and B see above)

$\mathbf{A_i} =$

0.03	0.10	0.02
0	0.50	0.20
0.03	0.10	0.02

$\mathbf{B_i} =$

0	0	0
0	1	0
0	0	0

$z =$ 0

I. Global Task

Given: a linear template as well as two static gray scale images $\mathbf{P_1}$ (result of the linear template operation (see the test template above and its output) and $\mathbf{P_2}$. (masked version of the original image). $\mathbf{P_3}$ is a binary version of $\mathbf{P_2}$ providing the fixed state mask for CNN operation. $\mathbf{P_3}$ indicates the positions of supporting pixels where the interpolation is fixed.. The result of the inverse of a linear template operation is computed rapidly using masked diffusion even if the template cannot be inverted (linear template – convolution kernel - have zero Eigen values).

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

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

Fixed State Mask: $\mathbf{X_{fix} = P_3}$

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

Output: $\mathbf{Y(t) \Rightarrow Y(\infty)}$ = Gray scale image containing the inverse of the B template operation (B=1-A).

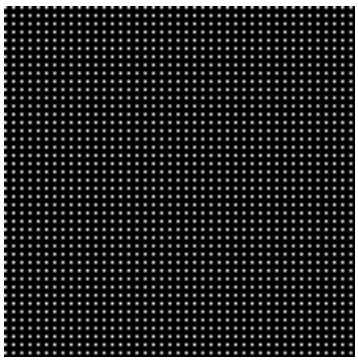
Remark:

The inverse operation of a linear template can be easily performed by using a dense support even if the theoretical inverse converges too slowly or if theoretically the template operation cannot be inverted.

II. Example: image names: LenaSCs.bmp, LenaSMask.bmp, MaskS.bmp; image size: 128x128; template name: DiffM2.tem .

INPUT (P_1)

INITIAL STATE (P_2)



MASK (P_3)

OUTPUT

Translation(dx,dy): *Translation by a fraction of pixel (dx,dy) with -1 ≤ dx ≤ 1 and -1 ≤ dy ≤ 1*

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} a\,b\,\mathrm{H}\,\mathrm{V} & (1-a)\,b\,\mathrm{H} & a\,b\,\mathrm{H}\,(1-\mathrm{V}) \\ a\,(1-b)\,\mathrm{V} & (1-a)\,(1-b) & a\,(1-b)\,(1-\mathrm{V}) \\ a\,b\,(1-\mathrm{H})\,\mathrm{V} & (1-a)\,b\,(1-\mathrm{H}) & a\,b\,(1-\mathrm{H})\,(1-\mathrm{V}) \end{bmatrix} \quad z = \boxed{0}$$

where

- $H = 1$, if $dy > 0$; $H = 0$, otherwise
- $V = 1$, if $dx > 0$; $V = 0$, otherwise
- $a = |dx|$ $b = |dy|$

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, $u_{ij} = 1$ for all virtual cells, denoted by [U]=1
Output:	**Y(t)⇒Y(∞)** = Grayscale image **P** shifted horizontally by *dx* and vertically by *dy*.

Remark:

Translations by a value greater than one pixel can be achieved by applying the same template many times. For example, if $dx > dy > 1$:

ny = trunc(*dy*); *fy* = *dy* – *ny*;
nx = trunc(*dx*); *fx* = *dx* – *nx*;
- Apply template *Translation(fx,fy)*
- repeat *nx-ny* times *Translation (1,1)*
- repeat *nx* times *Translation (1,0)*

II. Example image name: lenna.bmp, image size: 256x256; dx = -10.5; dy = 4.7

INPUT

OUTPUT

Rotation: ***Image rotation by angle ϕ around (O_x, O_y)***

The *Rotation* algorithm is based on a space-variant version of the *Translation(dx,dy)* template:

A =

0	0	0
0	0	0
0	0	0

B =

$a\,b$ H V	(1 - a) b H	$a\,b$ H (1 - V)
a (1 - b) V	(1 - a) (1 - b)	a (1 - b) (1 - V)
$a\,b$ (1 - H) V	(1 - a) b (1 - H)	$a\,b$ (1 - H) (1 - V)

z =

0

where

- $dx(i,j) = O_x + (j - O_x)\cos\varphi - (i - O_y)\sin\varphi - j$
- $dy(i,j) = O_y + (j - O_x)\sin\varphi + (i - O_y)\cos\varphi - i$
- $H = 1$, if $dy(i,j) > 0$; $H = 0$, otherwise
- $V = 1$, if $dx(i,j) > 0$; $V = 0$, otherwise
- $a = |dx(i,j)|$ $b = |dy(i,j)|$

If $|dx|>1$ or $|dy|>1$ for some i,j, an exact rotation is not possible with a neighborhood order 1.

We can perform an approximation with an error depending on ϕ, by applying the *Translation* template many times with the following algorithm:

- Compute the integer and fractional parts:
  $nx(i,j) = \text{trunc}(dx(i,j))$; $fx(i,j) = dx(i,j) - nx(i,j)$
  $ny(i,j) = \text{trunc}(dy(i,j))$; $fy(i,j) = dy(i,j) - ny(i,j)$
  $m = \max((nx(i,j), ny(i,j))$

- Apply space-variant template *Translation(fx,fy)*

- for k = 1 to m:

  if $|nx(i,j)| - k > 0$ then $kx(i,j) = \text{sign}(nx(i,j))$ else $kx(i,j) = 0$
  if $|nx(i,j)| - k > 0$ then $ky(i,j) = \text{sign}(ny(i,j))$ else $ky(i,j) = 0$
  Apply the space-variant template *Translation(kx, kx)*

- Apply the low-pass template:

A =

0	0	0
0	0	0
0	0	0

B =

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

z =

0

- Apply the hi-pass template:

A =

0	0	0
0	0	0
0	0	0

B =

-0.04	-0.12	-0.04
-0.12	1.57	-0.12
-0.04	-0.12	-0.04

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:	Fixed type, $u_{ij} = 1$ for all virtual cells, denoted by [U]=1
Output:	**Y(t)⇒Y(∞)** = Grayscale image **P** rotated by ϕ around (O_x, O_y).

II. Example: image name: lenna.bmp, image size: 256x256; $\phi = -10°$; $O_x = 0$; $O_y = 0$

INPUT

OUTPUT

Chapter 2. Subroutines

BLACK AND WHITE SKELETONIZATION

Old names: SKELBW

Task description and algorithm

The algorithm finds the skeleton of a black-and-white object. The 8 templates should be applied circularly, always feeding the output back to the input before using the next template [19].

The templates of the algorithm:

SKELBW1:

$\mathbf{A}_1 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_1 =$ | 1 | 1 | 0 |
|---|---|---|
| 1 | 5 | -1 |
| 0 | -1 | 0 |

$z_1 =$ -1

SKELBW2:

$\mathbf{A}_2 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_2 =$ | 2 | 2 | 2 |
|---|---|---|
| 0 | 9 | 0 |
| -1 | -2 | -1 |

$z_2 =$ -2

SKELBW3:

$\mathbf{A}_3 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_3 =$ | 0 | 1 | 1 |
|---|---|---|
| -1 | 5 | 1 |
| 0 | -1 | 0 |

$z_3 =$ -1

SKELBW4:

$\mathbf{A}_4 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_4 =$ | -1 | 0 | 2 |
|---|---|---|
| -2 | 9 | 2 |
| -1 | 0 | 2 |

$z_4 =$ -2

SKELBW5:

$\mathbf{A}_5 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_5 =$ | 0 | -1 | 0 |
|---|---|---|
| -1 | 5 | 1 |
| 0 | 1 | 1 |

$z_5 =$ -1

SKELBW6:

$\mathbf{A}_6 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_6 =$ | -1 | -2 | -1 |
|---|---|---|
| 0 | 9 | 0 |
| 2 | 2 | 2 |

$z_6 =$ -2

SKELBW7:

$\mathbf{A}_7 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_7 =$ | 0 | -1 | 0 |
|---|---|---|
| 1 | 5 | -1 |
| 1 | 1 | 0 |

$z_7 =$ -1

SKELBW8:

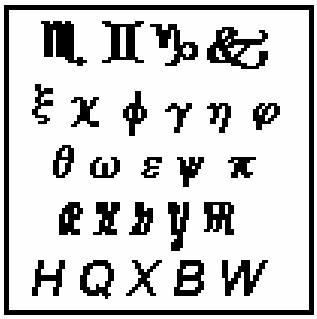
$\mathbf{A}_8 =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_8 =$ | 2 | 0 | -1 |
|---|---|---|
| 2 | 9 | -2 |
| 2 | 0 | -1 |

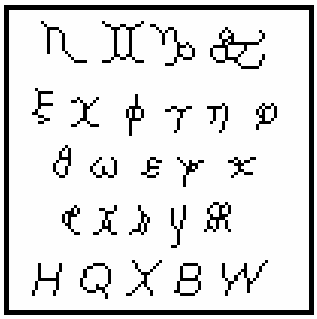
$z_8 =$ -2

The robustness of templates SKELBW1 and SKELBW2 are ρ(SKELBW1) = 0.18 and ρ(SKELBW2) = 0.1, respectively. Other templates are the rotated versions of SKELBW1 and SKELBW2, thus their robustness values are equal to the mentioned ones.

Example: image name: skelbwi.bmp, image size: 100x100; template names: skelbw1.tem, skelbw2.tem, …, skelbw8.tem.

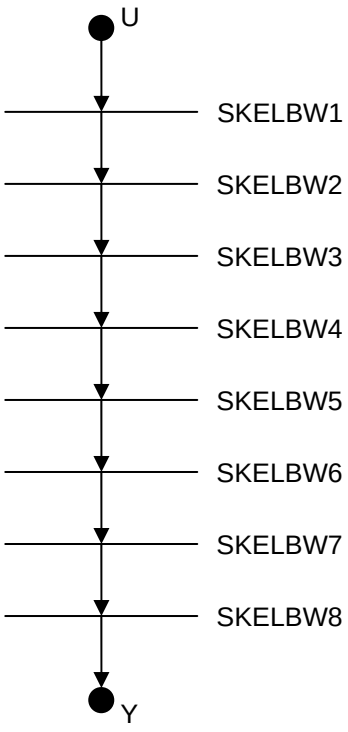


input



output

UMF diagram



GRAYSCALE SKELETONIZATION

Old names: SKELGS

Task description and algorithm

The algorithm finds the skeleton of grayscale objects. The algorithm uses 8 sets of templates, skeletonizing the objects circularly. Each step contains two templates. First, appropriate pixels are selected by the *selection* templates; afterwards these are used as fixed state masks for the *replacement*. The result of the replacement is fed back to the input. Only 4 of the 8 required templates are shown, the others can easily be generated by rotating functions *a* and *b*, as indicated in the first 3 steps [19].

The templates of the algorithm:

Selection templates: *Replacement:*

$\mathbf{A}_1$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_1$ = | ***a*** | ***a*** | 0 |
|---|---|---|
| ***a*** | 0 | ***b*** |
| 0 | ***b*** | 0 |

z_1 = -4.5

$\mathbf{A}_1$ = | ***c*** | ***c*** | 0 |
|---|---|---|
| ***c*** | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{A}_2$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_2$ = | ***a*** | ***a*** | ***a*** |
|---|---|---|
| 0 | 0 | 0 |
| ***b*** | ***b*** | 0 |

z_2 = -4.5

$\mathbf{A}_2$ = | ***c*** | ***c*** | ***c*** |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{A}_3$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_3$ = | 0 | ***a*** | ***a*** |
|---|---|---|
| ***b*** | 0 | ***a*** |
| 0 | ***b*** | 0 |

z_3 = -4.5

$\mathbf{A}_3$ = | 0 | ***c*** | ***c*** |
|---|---|---|
| 0 | 1 | ***c*** |
| 0 | 0 | 0 |

. . .

. . .

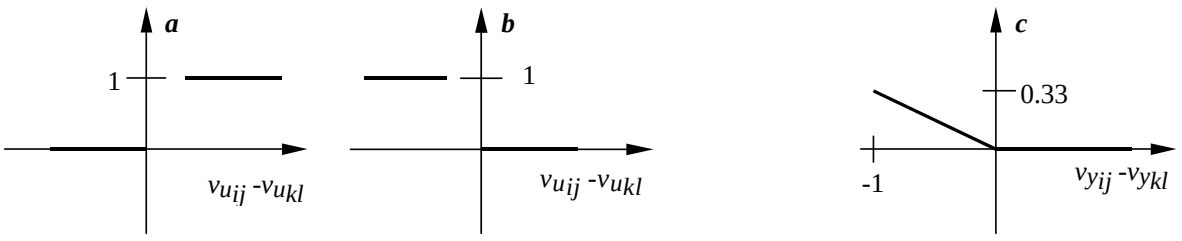
$\mathbf{A}_8$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_8$ = | ***a*** | 0 | 0 |
|---|---|---|
| ***a*** | 0 | ***b*** |
| ***a*** | 0 | ***b*** |

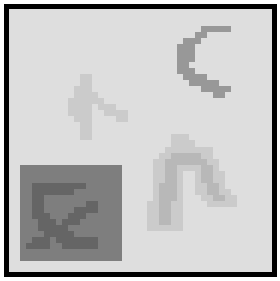
z_8 = -4.5

$\mathbf{A}_8$ = | ***c*** | 0 | 0 |
|---|---|---|
| ***c*** | 1 | 0 |
| ***c*** | 0 | 0 |

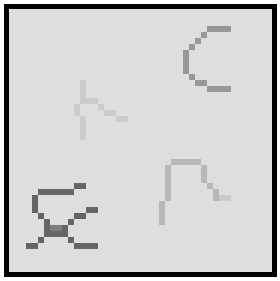
where ***a***, ***b*** and ***c*** are defined by the following nonlinear functions:



Example 1: image name: skelg2i.bmp; image size: 44x44.

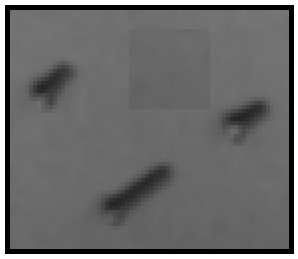


input

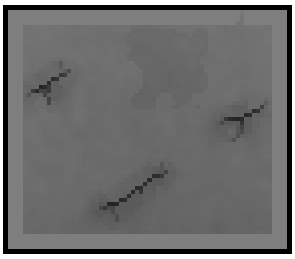


output

Example 2: image name: skelg1.bmp; image size: 70x60.



input



output

GRADIENT CONTROLLED DIFFUSION

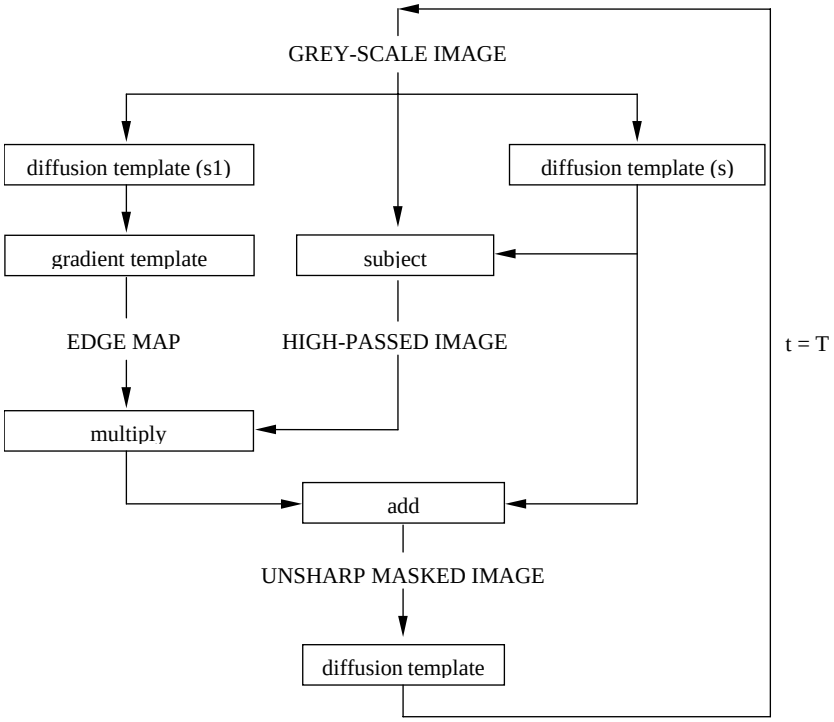
Task description and algorithm

Performs edge-enhancement during noise-elimination [17,25,30]. The equation used for filtering is as follows:

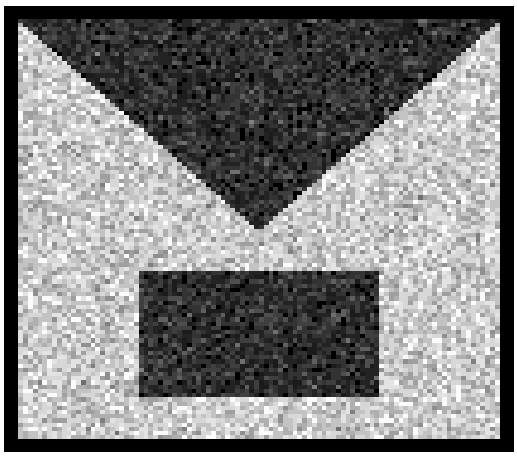
$$\frac{\partial I}{\partial t} = \Delta\left[I(x,y,t) \cdot \left(1 - k \cdot \left| grad\left(G(s) * I(x,y,t)\right)\right|\right)\right]$$

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.

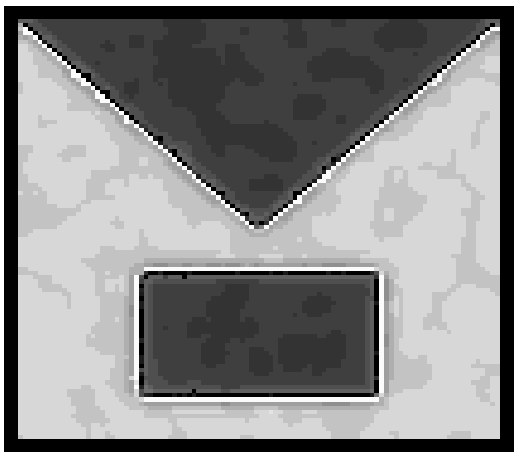
The flow-chart of the algorithm:



Example: image name: laplace.bmp; image size: 100x100.



input



output

SHORTEST PATH

Task description and algorithm

Two points given, the subroutine finds the shortest path connecting them. A labyrinth can also be defined, where the walls (black pixels) denote forbidden cells. The algorithm runs in a time proportional to the length of the shortest path. The algorithm can further be generalized to connect several points. It was also used to design the layout of printed circuit boards.
The subroutine contains two phases. In the first phase, the black-and-white labyrinth should be applied to the input, and one of the two points to the initial state (a white point against a black background). In the second step, when the shortest route has been selected, the output of the first step serves as input, and the other point to be connected is the initial state (black point on a white background). In this step four templates are to be applied cyclically.

The templates of the algorithm:

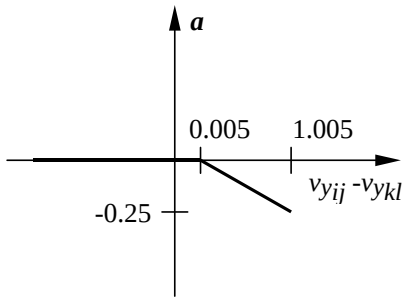
Explore:

$\mathbf{A}_{explore}$ = | 0 | ***a*** | 0 |
|---|---|---|
| ***a*** | 1 | ***a*** |
| 0 | ***a*** | 0 |

$\mathbf{B}_{explore}$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 3 | 0 |
| 0 | 0 | 0 |

$z_{explore}$ = | 3 |
|---|

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



Select:

$\mathbf{A}_{right}$ = | 0 | 0 | 0 |
|---|---|---|
| 1 | 3 | 0 |
| 0 | 0 | 0 |

$\mathbf{A}_{down}$ = | 0 | 1 | 0 |
|---|---|---|
| 0 | 3 | 0 |
| 0 | 0 | 0 |

$\mathbf{A}_{left}$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 3 | 1 |
| 0 | 0 | 0 |

$\mathbf{A}_{up}$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 3 | 0 |
| 0 | 1 | 0 |

$\mathbf{B}_{right}$ = | 0 | 0 | 0 |
|---|---|---|
| ***b*** | 0 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_{down}$ = | 0 | ***b*** | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

$\mathbf{B}_{left}$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | ***b*** |
| 0 | 0 | 0 |

$\mathbf{B}_{up}$ = | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | ***b*** | 0 |

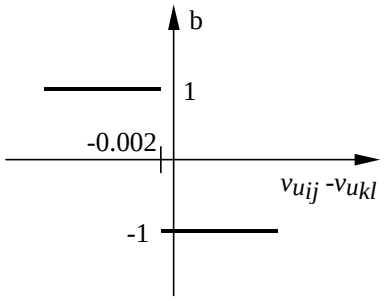
z_{right} = | 1 |
|---|

z_{down} = | 1 |
|---|

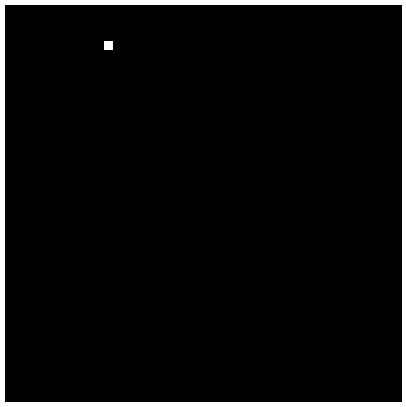
z_{left} = | 1 |
|---|

z_{up} = | 1 |
|---|

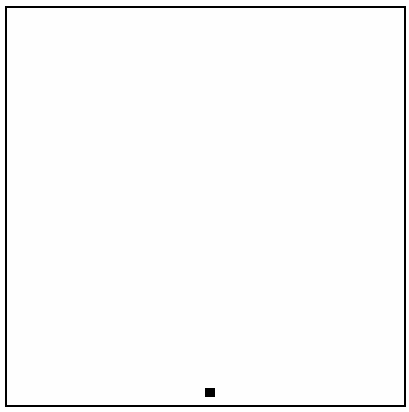
where ***b*** is defined by the following nonlinear function:



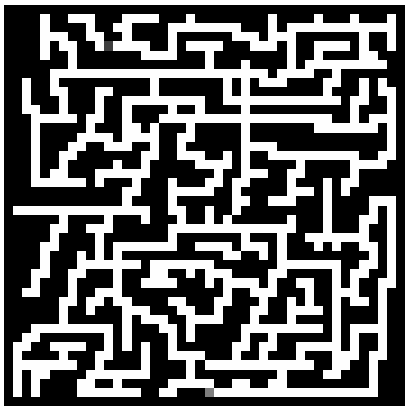
Example : image names: shpath1.bmp, shpath2.bmp, shpath3.bmp; image size: 44x44.



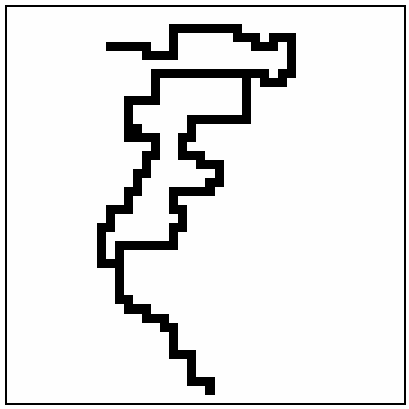
initial state of the 1. step



initial state of the 2. step



labyrinth

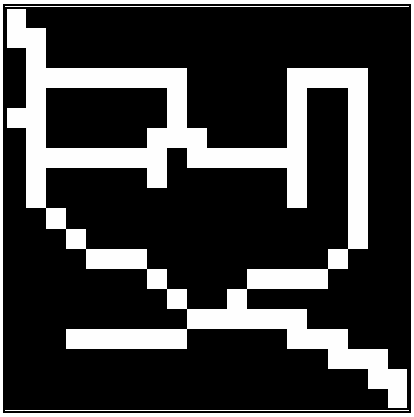


shortest paths

J-FUNCTION OF SHORTEST PATH

Task description and algorithm

A labyrinth is defined as a binary picture, where the white points (-1) represent the free cells, and black pixels (+1) represent the obstacles.



The task is to find the shortest path from a given startpoint to a given endpoint. The task can be solved through a so-called J-function, which gives for every point the length of the shortest path from the startpoint to the given point in some appropriate measure (the length of the path is scaled down into the interval [-1,+1] in the CNN solution).
The minimum in a given neighborhood can be computed through a difference-controlled template. Thus the computation can be carried out through a two layer nonlinear CNN. The first layer represents the J-function, and on the second layer the actual value of the J-function increased by the unit length is computed. The startpoint is given by a binary image, where a white pixel represents the startpoint and the other points are black. The J-function is obtained as an output picture, which values are related to the value of J. Obstacles have J=1, i.e. they are black.

The templates of the algorithm:

Layer1 : Minimum selection:

$\mathbf{A_{11}}=$

0	0	0
0	1	0
0	0	0

$\mathbf{D_{12}}=$

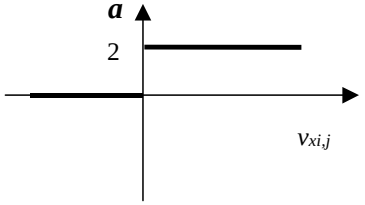
d	***d***	***d***
d	0	***d***
d	***d***	***d***

$\mathbf{B_{11}}=$

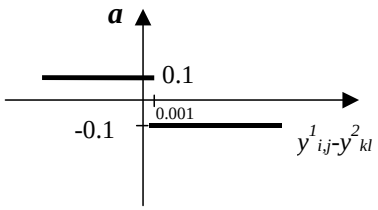
0	0	0
0	***a***	0
0	0	0

$z=$ -0.8

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



and ***d*** is defined by the following function:



Layer2: Increased J-function:

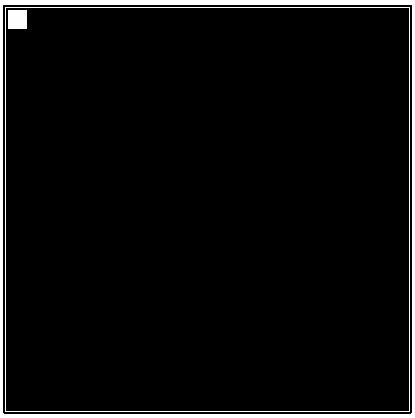
$\mathbf{A_{21}}$=

0	0	0
0	1	0
0	0	0

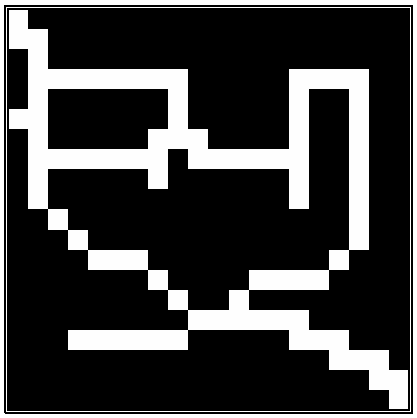
z = ***c***

where c=2/maximal path length

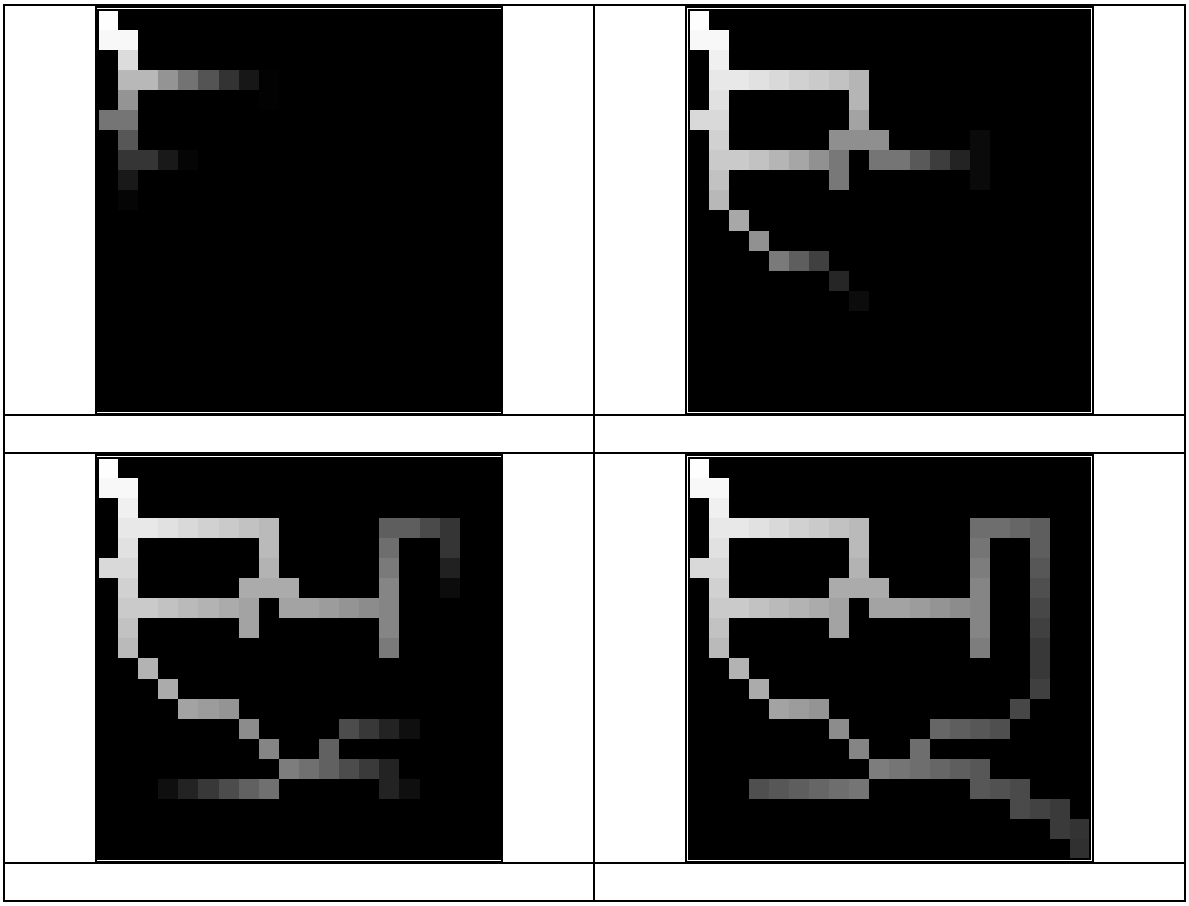
Example: image names: init.bmp, mask.bmp; image sizes: 20x20.



initial state



labirinth

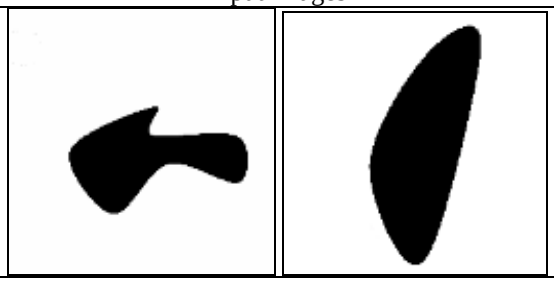
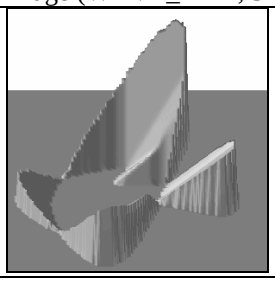


Transient sampled at different time values

NONLINEAR WAVE METRIC COMPUTATION

Task description and algorithm

This algorithm implements a wave metric for object comparison based on nonlinear spatio-temporal wave process. The method uses nonlinear wave propagation to explore the properties of objects. From the intersections of object-pairs wave fronts propagate through the unions of objects and the time evolution of the process is recorded on a special wave map. This wave map contains aggregated information about the dynamics of the wave process; and several different measures can be extracted from this map focusing on the desired aspect of the comparison. The key steps of metric computations are wave based transformation of objects, wave map generation where the associated gray-scale values are related to the time required for the wave to reach a given position, and distance calculation from the wave map. The detailed description of the method can be found in [46]. Two types of implementation are presented, namely, a dynamic solution where wave map is generated on a two-layer CNN architecture in a single transient, using nonlinear template interactions (two-layer UMF machine), and an iterative type of method which was implemented on the 64x64 I/O CNN-UM chip [47]. The main advantage of the method is that several object pairs can be compared to each other at the same time.

Input images	Output image (WAVE_MAP, 3D view)
	

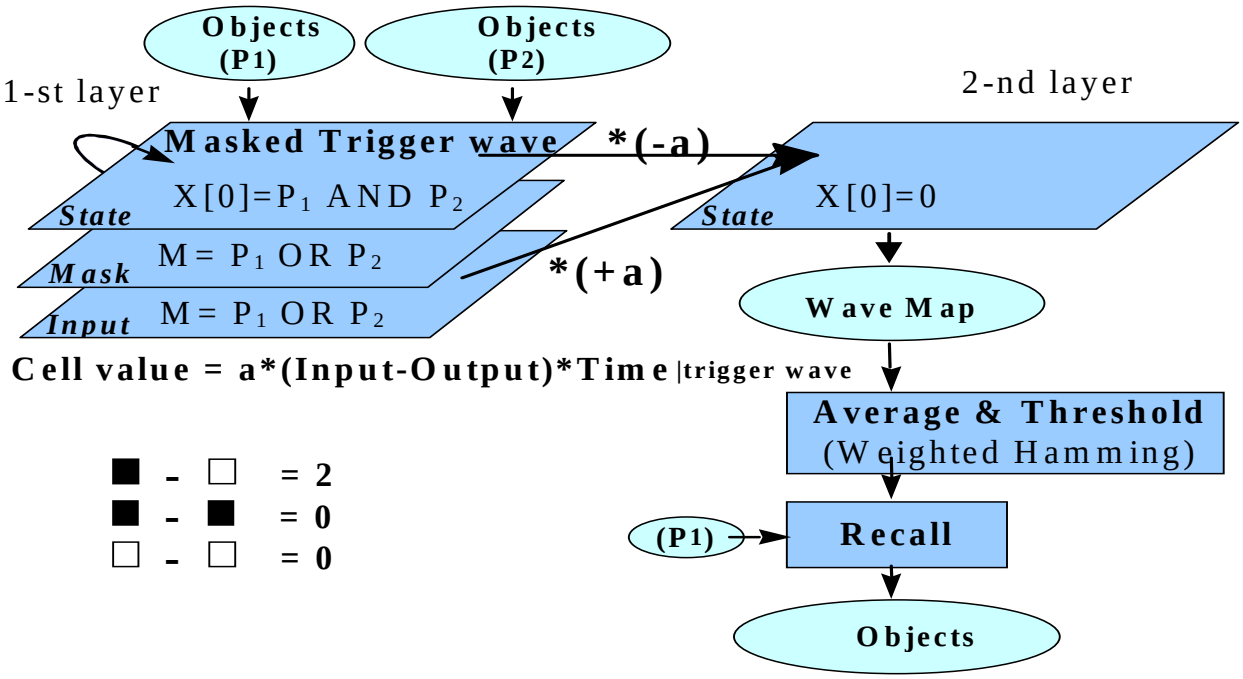
Input Parameters

U_1, U_2	Input binary images (object set A and set B)
B_WAVE	Binary wave propagation
WAVE_MAPPING	Depending on the machine it can be a nonlinear template or a simple linear
WAVE_MAP	Spatial map encoding dynamics of wave propagation
DIFFUS and THRESHOLD	Approximation of Integrated Hausdorff metric [1]

Output Parameters

MARKERS	Result of comparison: marker points of selected objects. These are to be used for further processing. Application example can be found in [2].

Dynamic implementation



Template for trigger wave generation

$\mathbf{A}_{1,1}$ =

0.41	0.59	0.41
0.59	2	0.59
0.41	0.59	0.41

B =

0	0	0
0	0	0
0	0	0

z = 4.5

Template for current filling

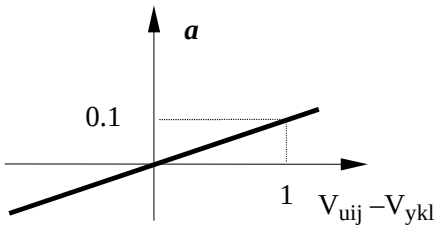
$\mathbf{A}_{1,2}$ =

0	0	0
0	*a*	0
0	0	0

B =

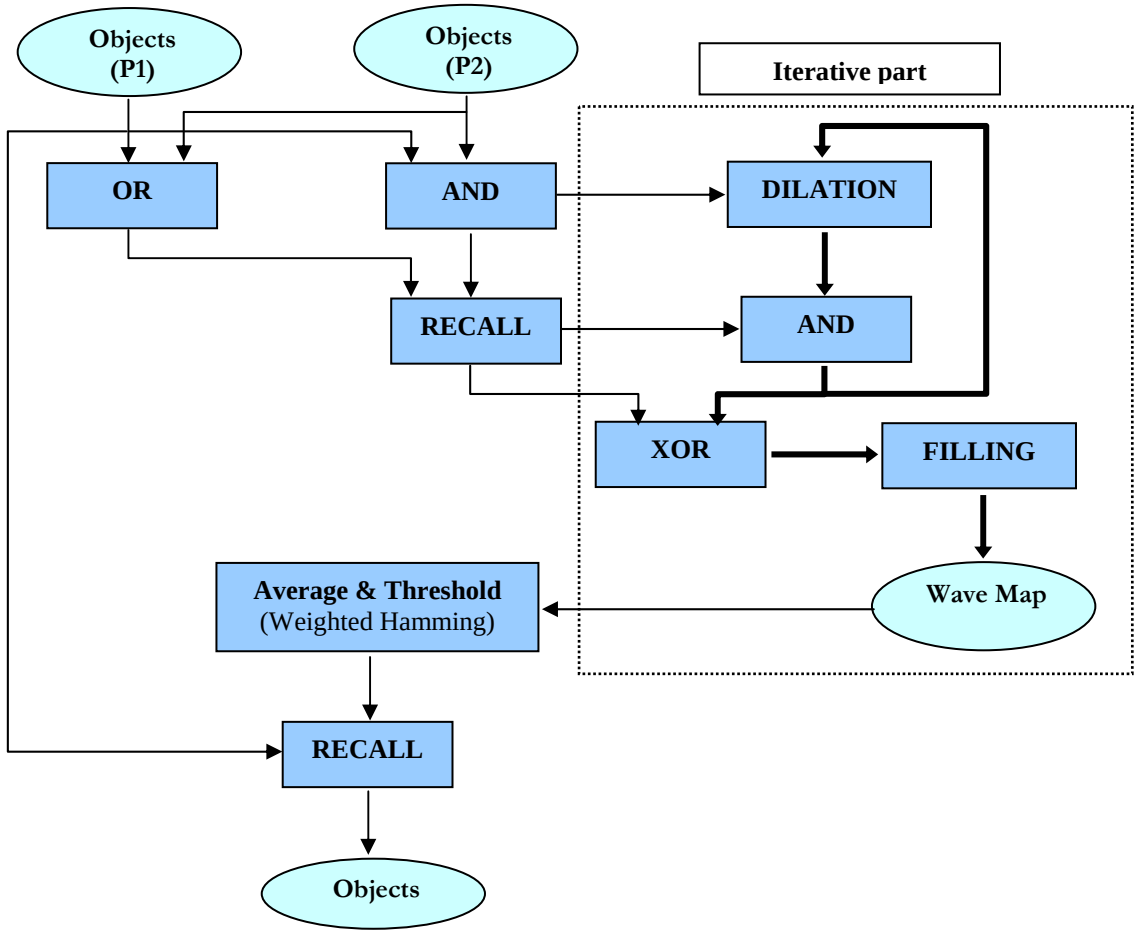
0	0	0
0	0	0
0	0	0

z = 0



Iterative type implementation

The dilation template



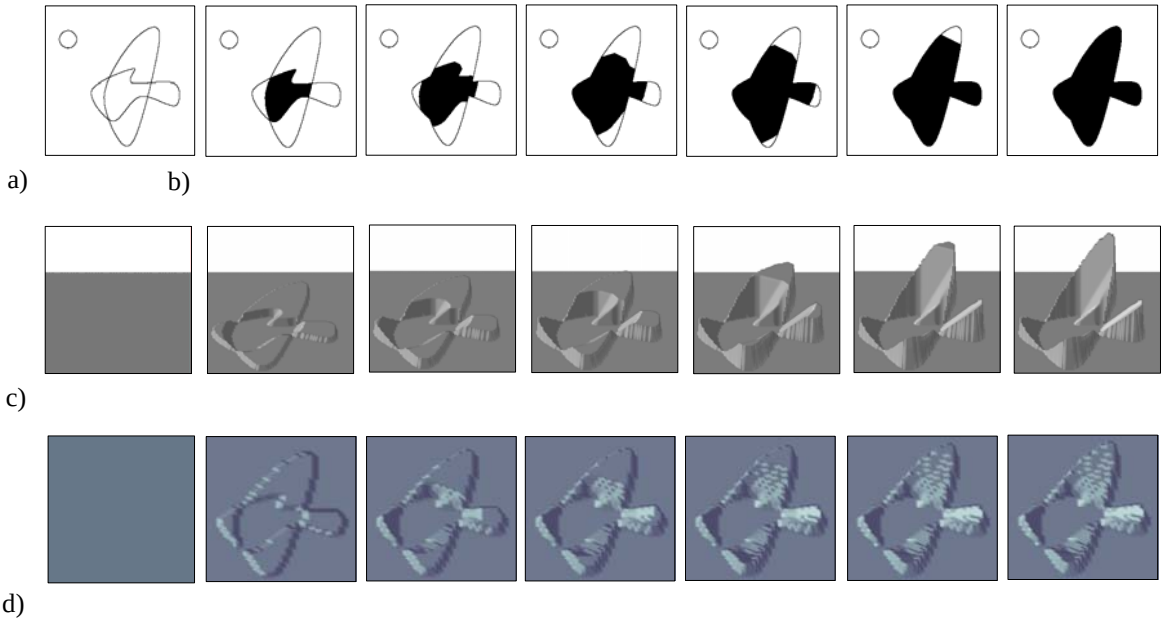
A = | 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

B = | 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

z = 8.5

The other cited templates can be found in this template library.

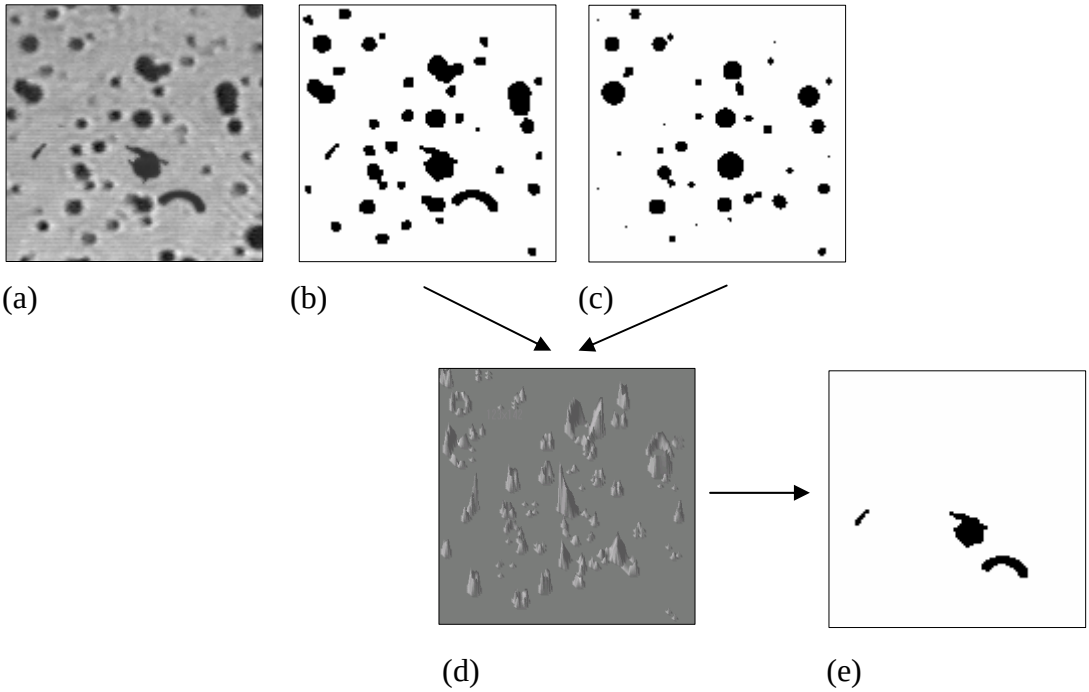
Example – 1



Wave Map generation. a) Outlines of two partially overlapping point sets, b) Trigger wave spreads from the intersection through the union of contiguous parts of point sets until all the points become triggered, c) Wave map generated by increasing intensities of pixels until trigger wave reaches them, simulation result d) Consecutive steps of generating Wave Map on the 64x64 I/O CNN-UM chip.

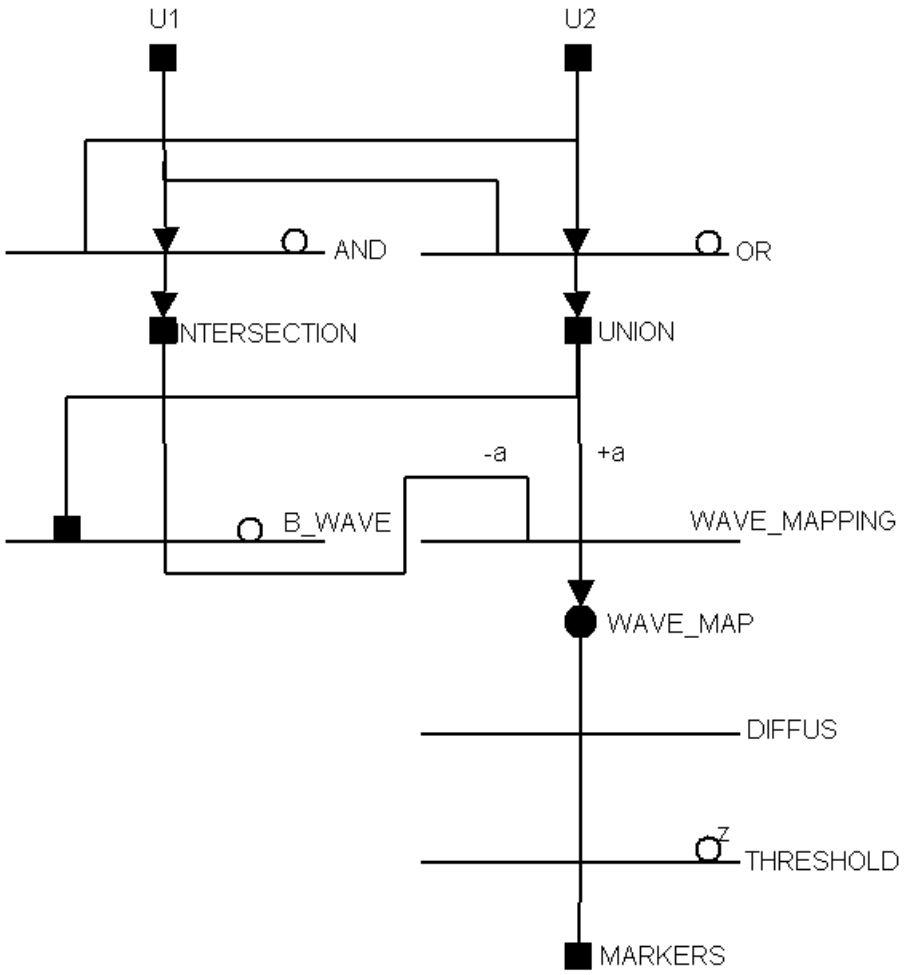
Example – 2

A possible application of the nonlinear wave metric was presented in [48]. The study addressed the problem of conditional basement maintenance of engines, concerning the on-line monitoring system that should forecast engine malfunction. Optical sensing of the oil flow of the engine was applied and debris particles were detected by using model-based object recognition. The comparison of object-model pairs was performed via nonlinear wave metric.



Bubble-debris classification algorithm (a) original gray-scale image, (b) adaptive threshold, (c) bubble models, (d) wave map, (e) detected debris particles

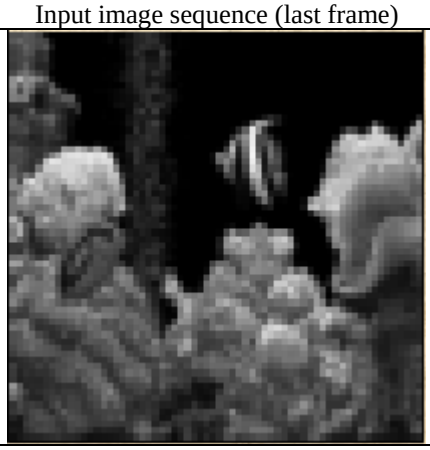
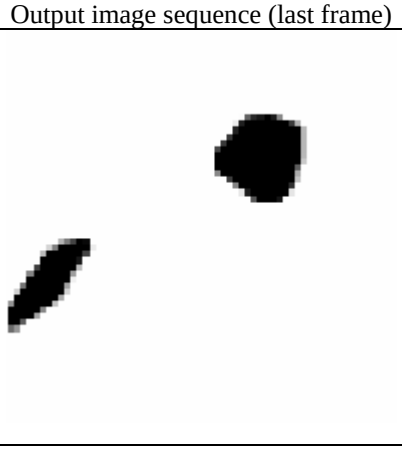
UMF diagram


U1
U2
AND
OR
INTERSECTION
UNION
-a
+a
B_WAVE
WAVE_MAPPING
WAVE_MAP
DIFFUS
z
THRESHOLD
MARKERS

MULTIPLE TARGET TRACKING

Task description and algorithm

A simple thresholded-difference method is required to maintain both hardware and processing time complexity low. The history of detected targets is used to assure the quality of tracking until target is disappeared from the scene [63].

Input image sequence (last frame)	Output image sequence (last frame)
	

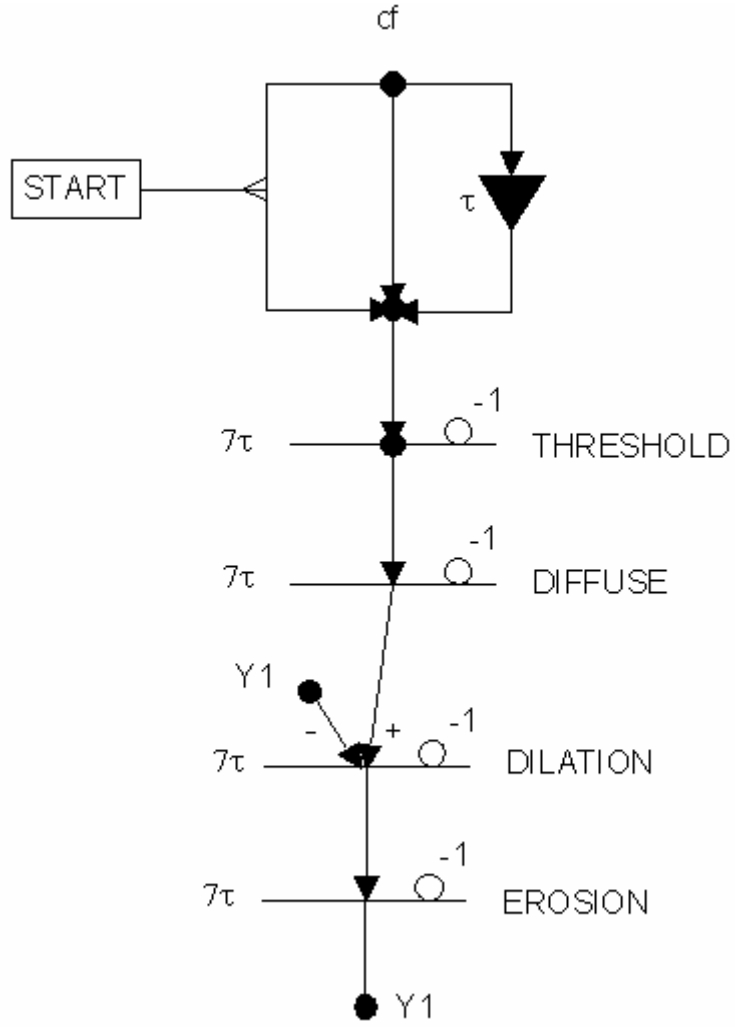
Input Parameters

cf	Current frame (input image)

Output Parameters

Y1	Detected targets in the present frame (output image)

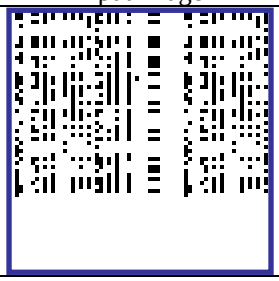
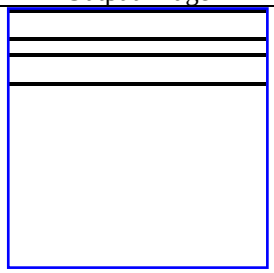
UMF diagram


cf
START
τ
-1
7τ
THRESHOLD
-1
7τ
DIFFUSE
Y1
-
+
-1
7τ
DILATION
-1
7τ
EROSION
Y1

MAXIMUM ROW(S) SELECTION

Task description and algorithm

This algorithm detects rows where the number of black pixels is the greatest [64]. Two different implementations are presented: a longer one with simple linear templates and binary storage (A), and a shorter one with non-linear templates (B).

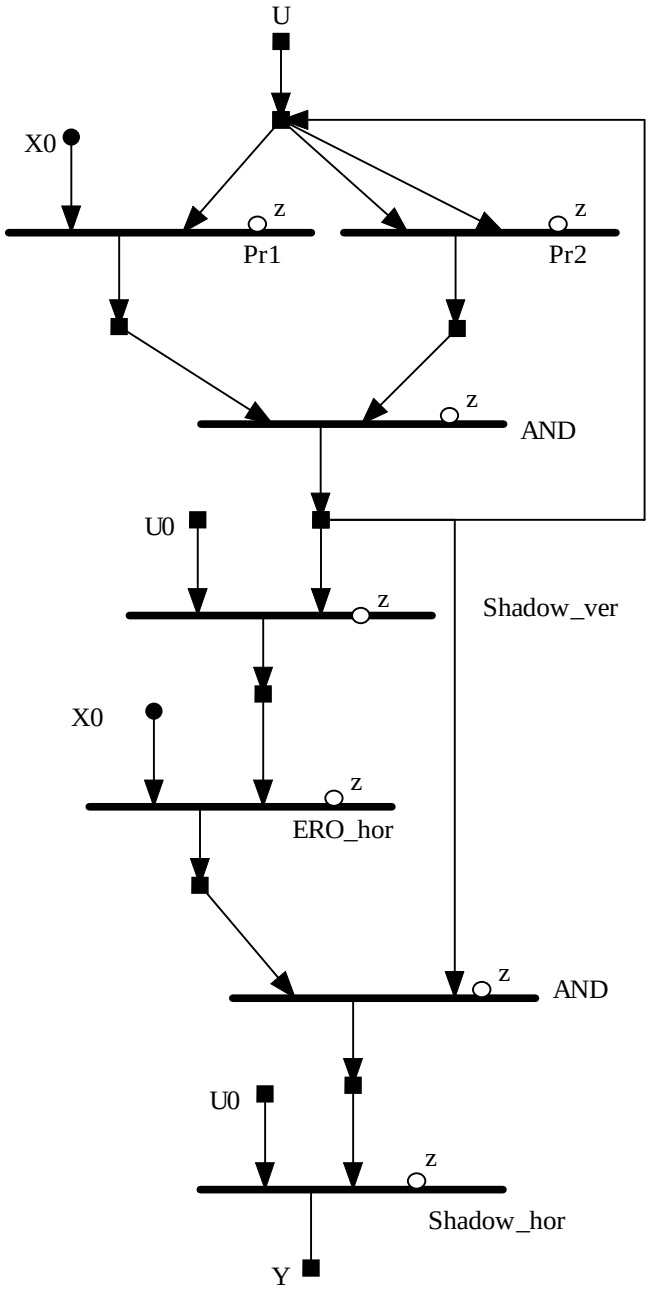
Input image	Output image
	

Input Parameters: –

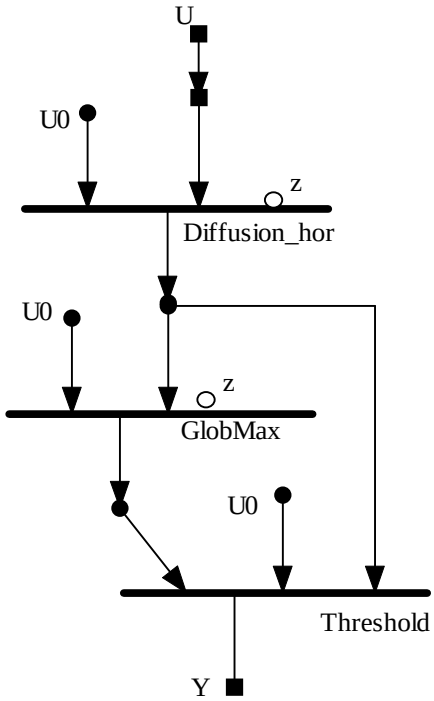
Remarks:

- The hardware complexity (chip area) of the (A) solution is much lower than the (B) solution
- The theoretical scalability in space (increasing the array size) is better in the (B) solution
- The practical scalability in space (tiling the input) is better in the (A) solution
- The computational time (speed and power consumption) depends on the available boundary conditions

UMF diagram



A

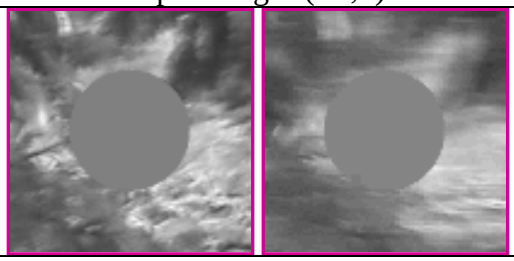
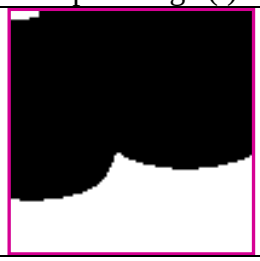


B

SUDDEN ABRUPT CHANGE DETECTION

Task description and algorithm

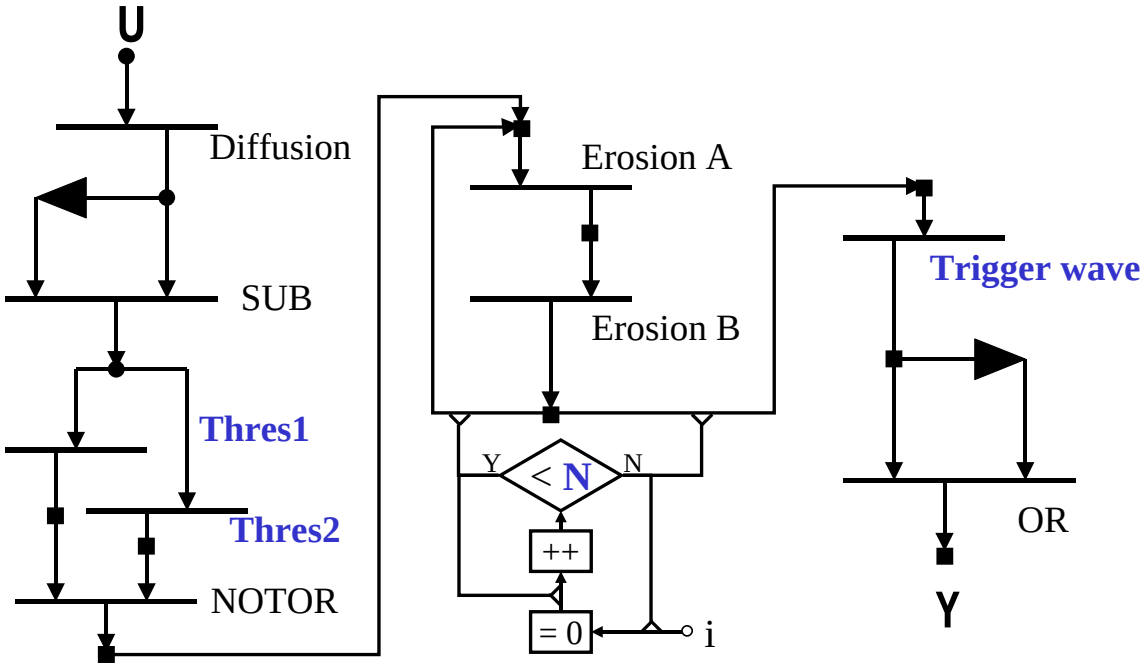
This algorithm detects the places and time instances in a video flow where a sudden abrupt change is happened based on neurobiological measurements [65]. The generated map can be used to re-initialize the history-based processing in a general image flow processing application [66].

Input images (t-1, t)	Output image (t)
	

Input Parameters

U	Input grayscale image flow
Thres	Threshold level
N	Number of the erosion steps
Trigger wave	Expansion size ~ trigger wave running time

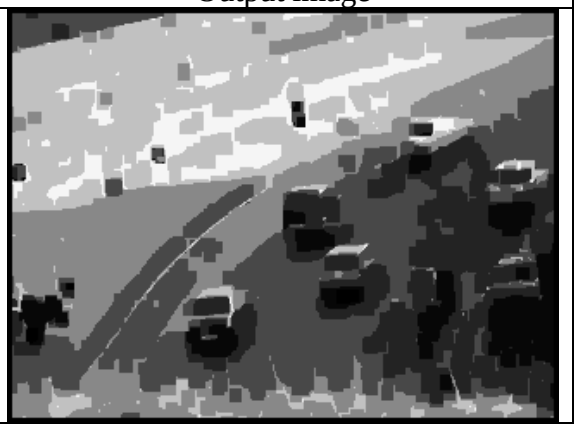
UMF diagram


U
Diffusion
SUB
Thres1
Thres2
NOTOR
Erosion A
Erosion B
Y
< N
N
++
= 0
i
Trigger wave
OR
Y

HISTOGRAM MODIFICATION WITH EMBEDDED MORPHOLOGICAL PROCESSING OF THE LEVEL-SETS

Task description and algorithm

This algorithm is designed for simultaneous contrast enhancement, noise suppression and shape enhancement [67]. An adaptive multi-thresholded output with shape enhancement can be obtained if a morphological processing is embedded at each gray-scale level considered. This could be implemented either through a multi-step erosion and dilation operations or using trigger-waves that approximate a continuous-scale binary morphology with flat structuring elements

Input image	Output image
	

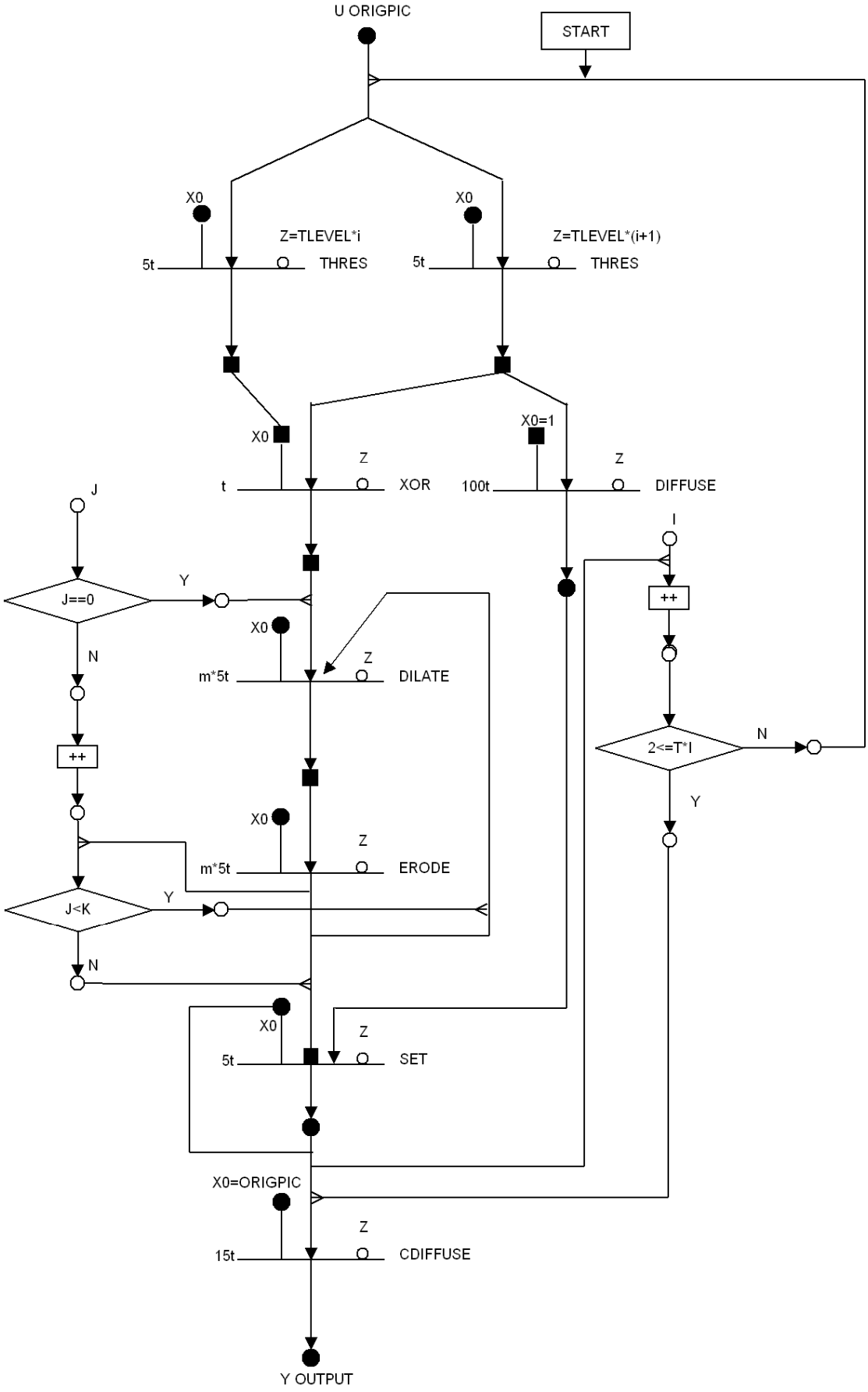
Input Parameters

U	Input image
I,J	Indexes
K	Number of morphological steps
M	(Def = 1) Heuristic value
T(LEVEL)	Number of levels

Output Parameters

Y	Output image

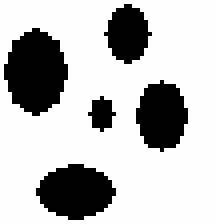
UMF diagram


U ORIGPIC
START
X0
Z=TLEVEL*i
5t
THRES
X0
Z=TLEVEL*(i+1)
5t
THRES
X0
Z
t
XOR
X0=1
Z
100t
DIFFUSE
J
J==0
Y
N
X0
Z
m*5t
DILATE
++
X0
Z
m*5t
ERODE
J<K
Y
N
I
++
2<=T*I
N
Y
X0
Z
5t
SET
X0=ORIGPIC
Z
15t
CDIFFUSE
Y OUTPUT

OBJECT COUNTER

Task description and algorithm

This algorithm counts the distinct objects in a binary image. The routine is a member of the iterative structure family. The algorithm iteratively selects and removes one single object at a time from a set of objects in a picture until no black pixel is present. One counts the number of the necessary iterations can tell the number of the objects in the original input picture.

Input image	Output (scalar)
	5 objects

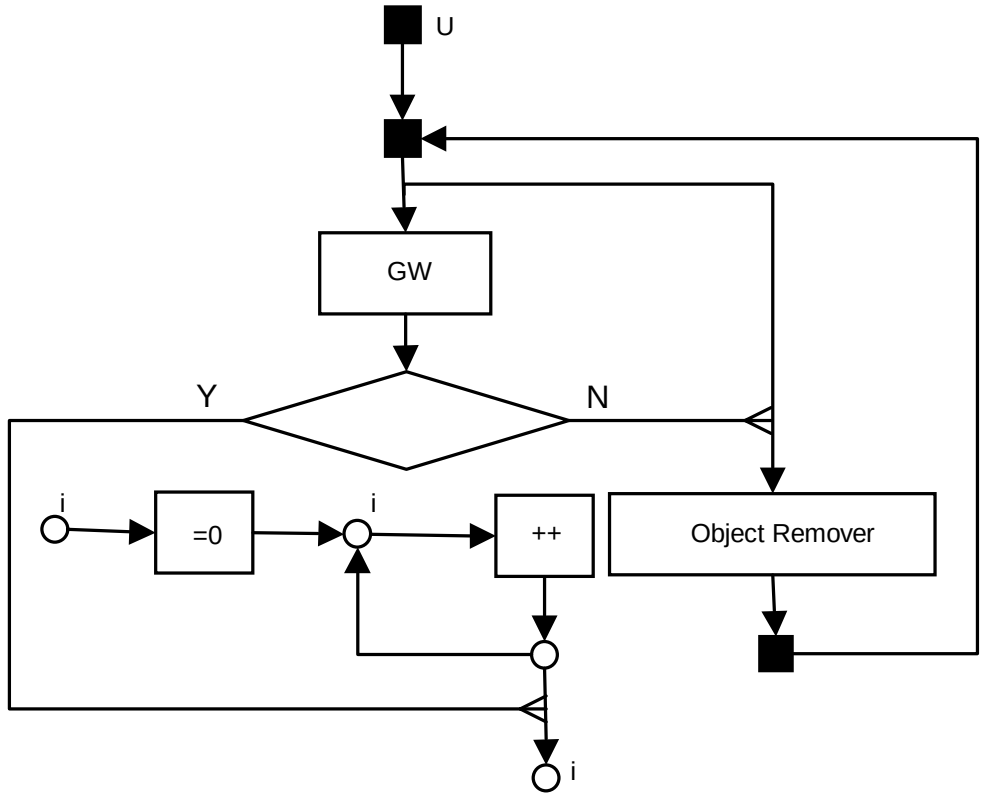
Input Parameters

U	Input image

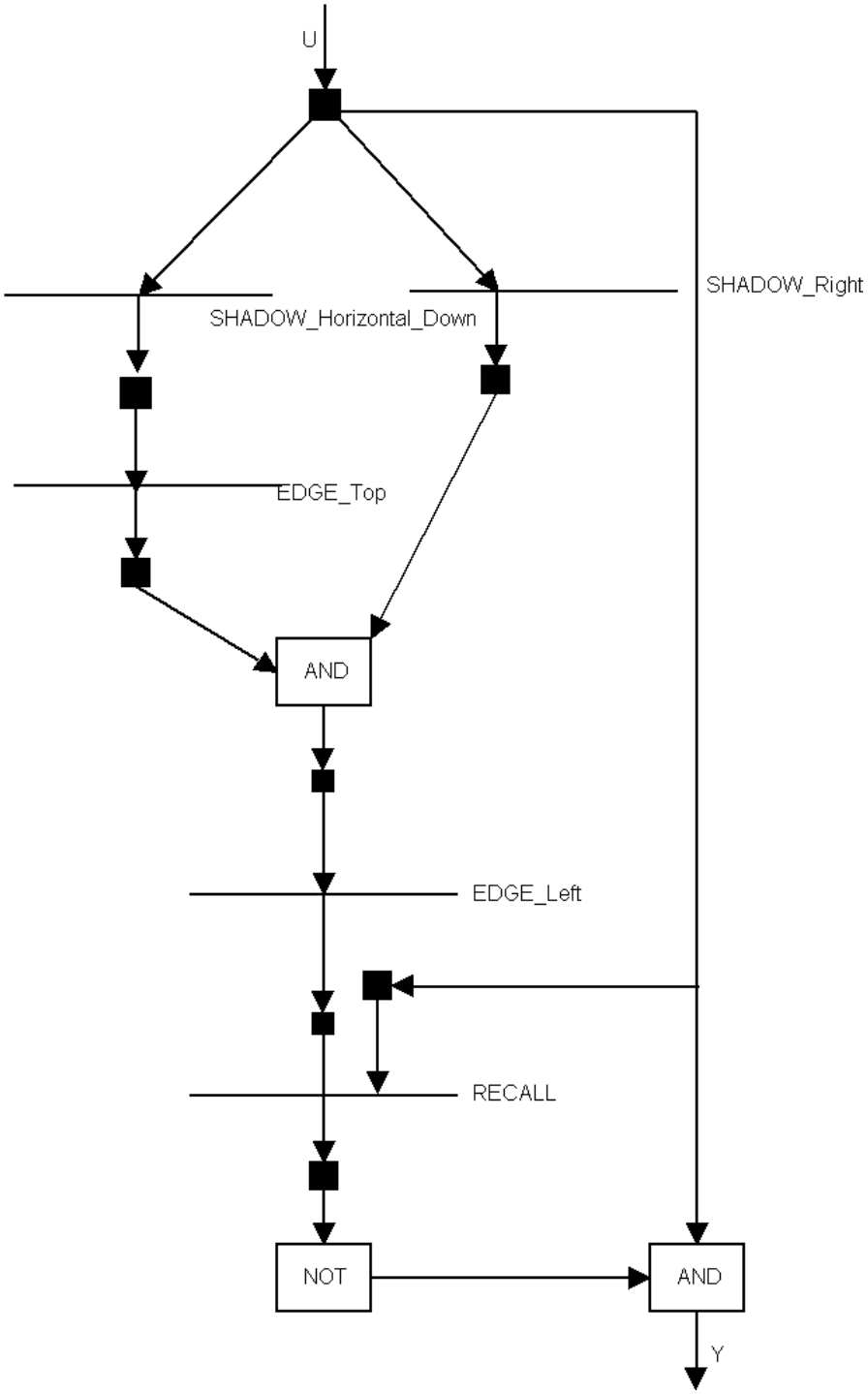
Output Parameters

I	Number of detected objects

UMF diagram



Subroutine **Object Remover**

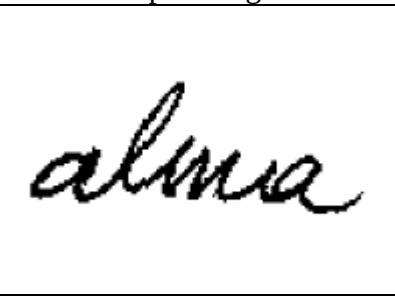


HOLE DETECTION IN HANDWRITTEN WORD IMAGES

Task description and algorithm

This algorithm [68] detects the holes in a handwritten word image and classifies them into four classes that are used as features in handwriting recognition:

- holes (holes like the one in letter ‘o’)
- small holes (holes like the one in letter ‘e’)
- upper holes (holes in the ascender region)
- lower holes (holes in the descender region)

Input image	Output image
	

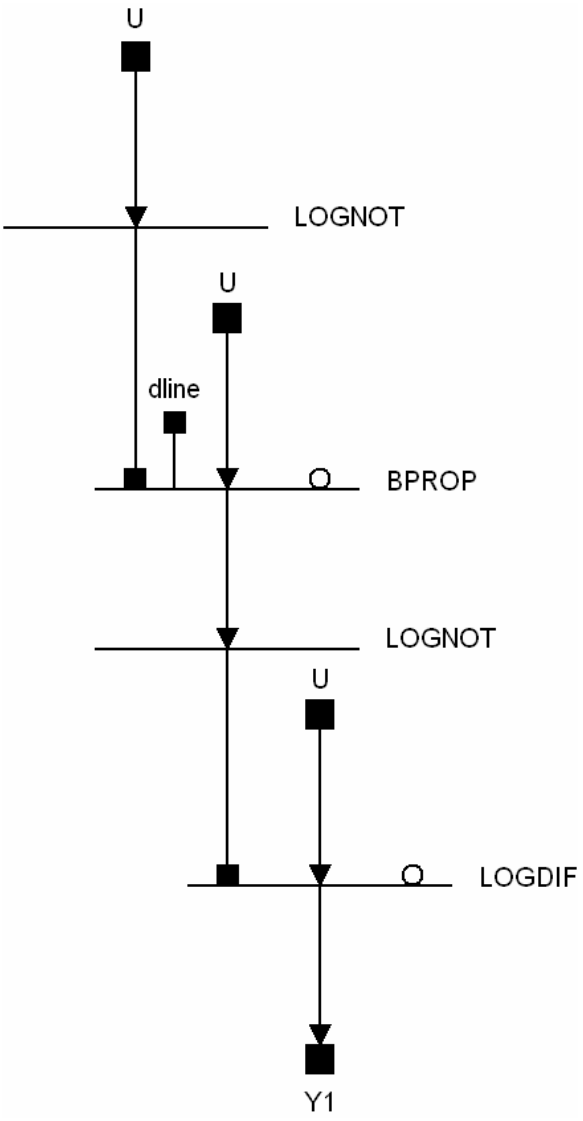
Input Parameters

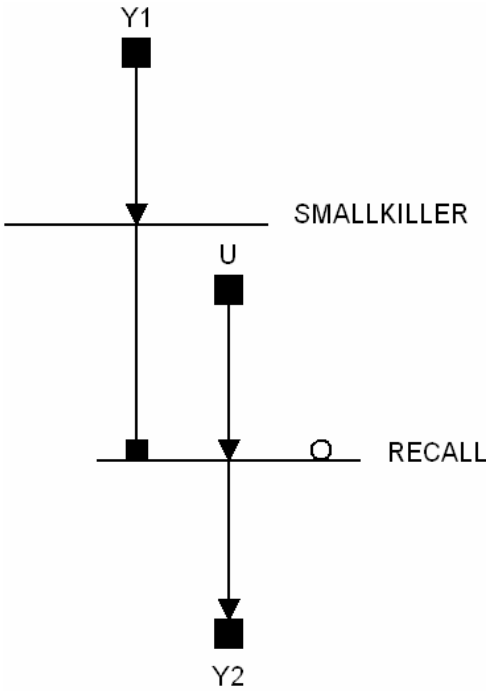
U	Input image
uline / dline	Upper / lower baselines

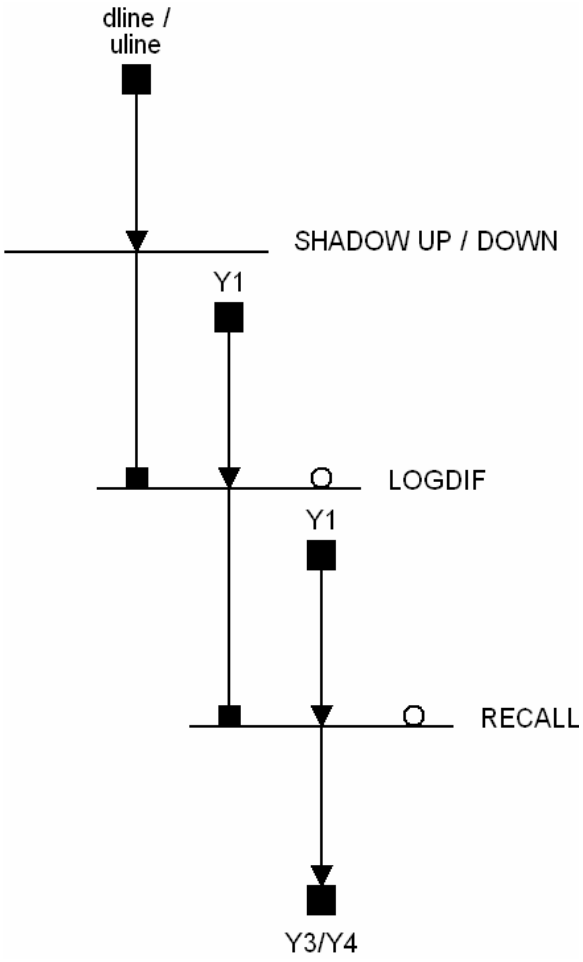
Output Parameters

Y1	Holes detected
Y2	Small holes
Y3 / Y4	Upper / lower holes

UMF diagram


U
LOGNOT
U
dline
BPROP
LOGNOT
U
LOGDIF
Y1


Y1
SMALLKILLER
U
RECALL
Y2


dline /
uline
SHADOW UP / DOWN
Y1
LOGDIF
Y1
RECALL
Y3/Y4

AXIS OF SYMMETRY DETECTION ON FACE IMAGES

Task description and algorithm

This algorithm detects axis of symmetry on face images [69]. For estimating the axis of symmetry we take an assumption that the region of nose on a face is the most vertically detailed region of face

Input image	Output image
	

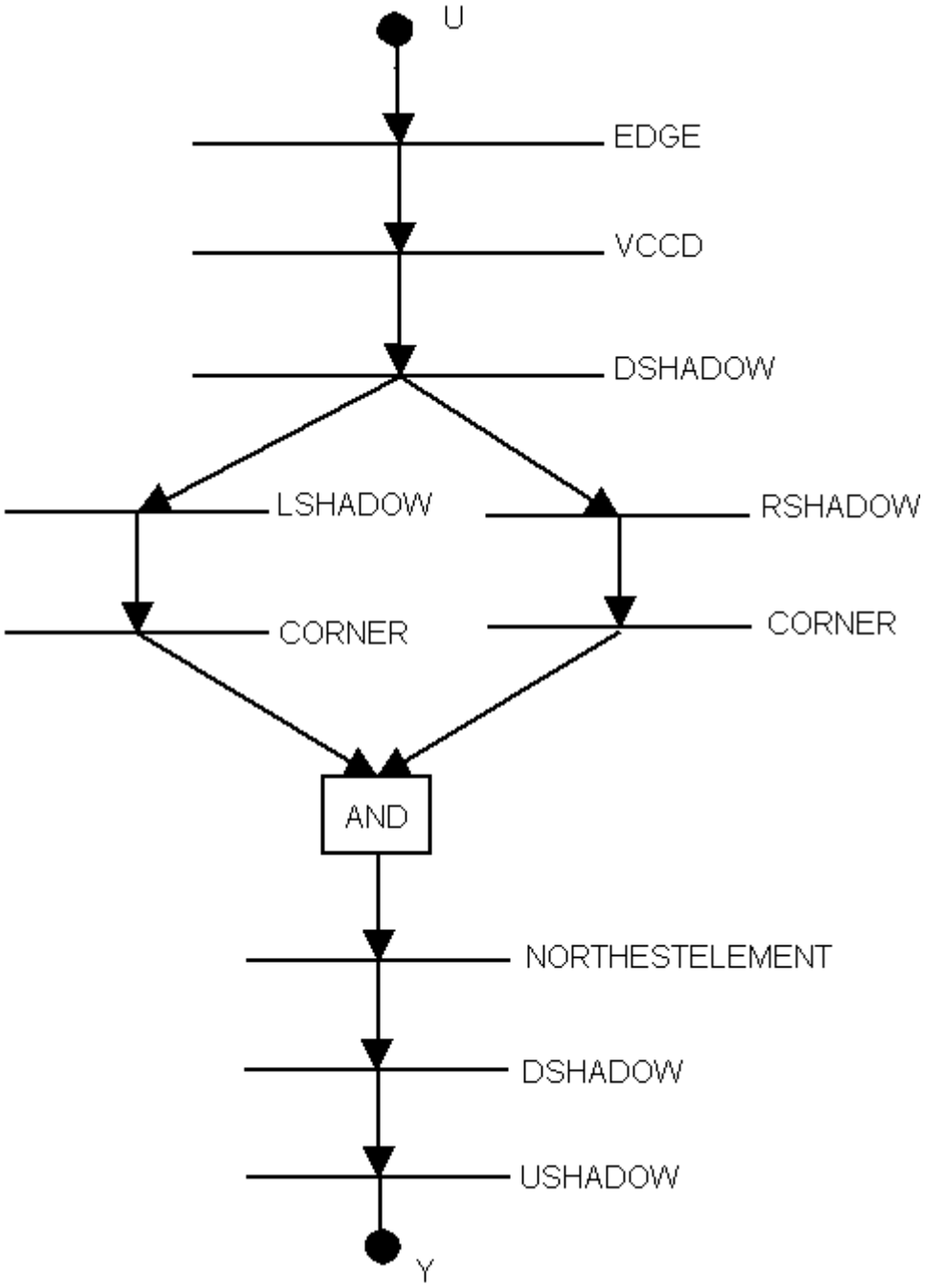
Input Parameters

U	Face image

Output Parameters

Y	Axis of symmetry detected

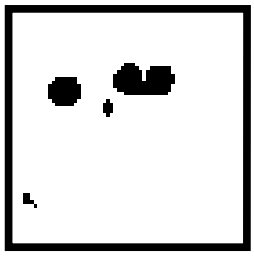
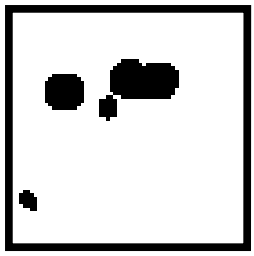
UMF diagram


U
EDGE
VCCD
DSHADOW
LSHADOW
RSHADOW
CORNER
CORNER
AND
NORTHESTELEMENT
DSHADOW
USHADOW
Y

ISOTROPIC SPATIO-TEMPORAL PREDICTION CALCULATION BASED ON PREVIOUS DETECTION RESULTS

Task description and algorithm

This algorithm calculates the likely position of moving targets based on their current detected position [70].

Input image	Output image
	

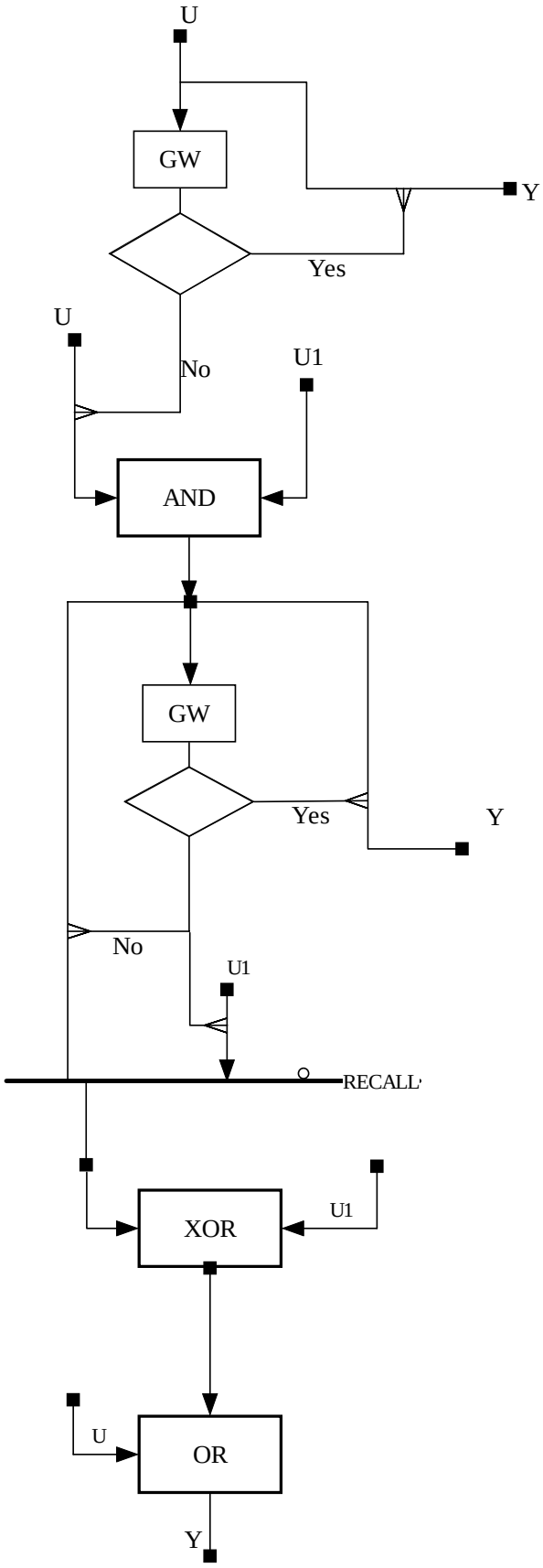
Input Parameters

U	Input image
U1	Previous prediction image

Output Parameters

Y	Current prediction

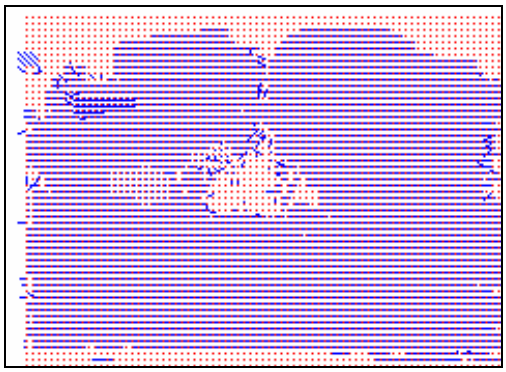
UMF diagram



MULTI SCALE OPTICAL FLOW

Task description and algorithm

This algorithm selects the most likely motion vectors in each pixel based upon two subsequent frames of a video sequence [71].
The algorithm consists of three recursively embedded steps operating on the current and previous frames of the sequence. The outer step is responsible for cycling on scales, which factorizes the "scale" from "Iscale_low" up to the limit in each cycle. This scalar value is provided for the Probability Distribution Function (PDF) generator (PDFFull(scale)) which in turn creates 2Nx2N images where N is the maximal displacement of frames to be detected, measured in pixels. The schema depicts the case in which *N=1*. Finally, the most embedded subroutine (PDF(scale)) is responsible for generating a single PDF in a single direction. In the operation, it realizes L_1 norm between its two inputs, diffuses the result and finally puts in a *exp(-x)* function pixel-wise.

Input images	Output image
 Ucurr  Uprev	 Y composed of Y_PDFFull_x and Y_PDFFull_x

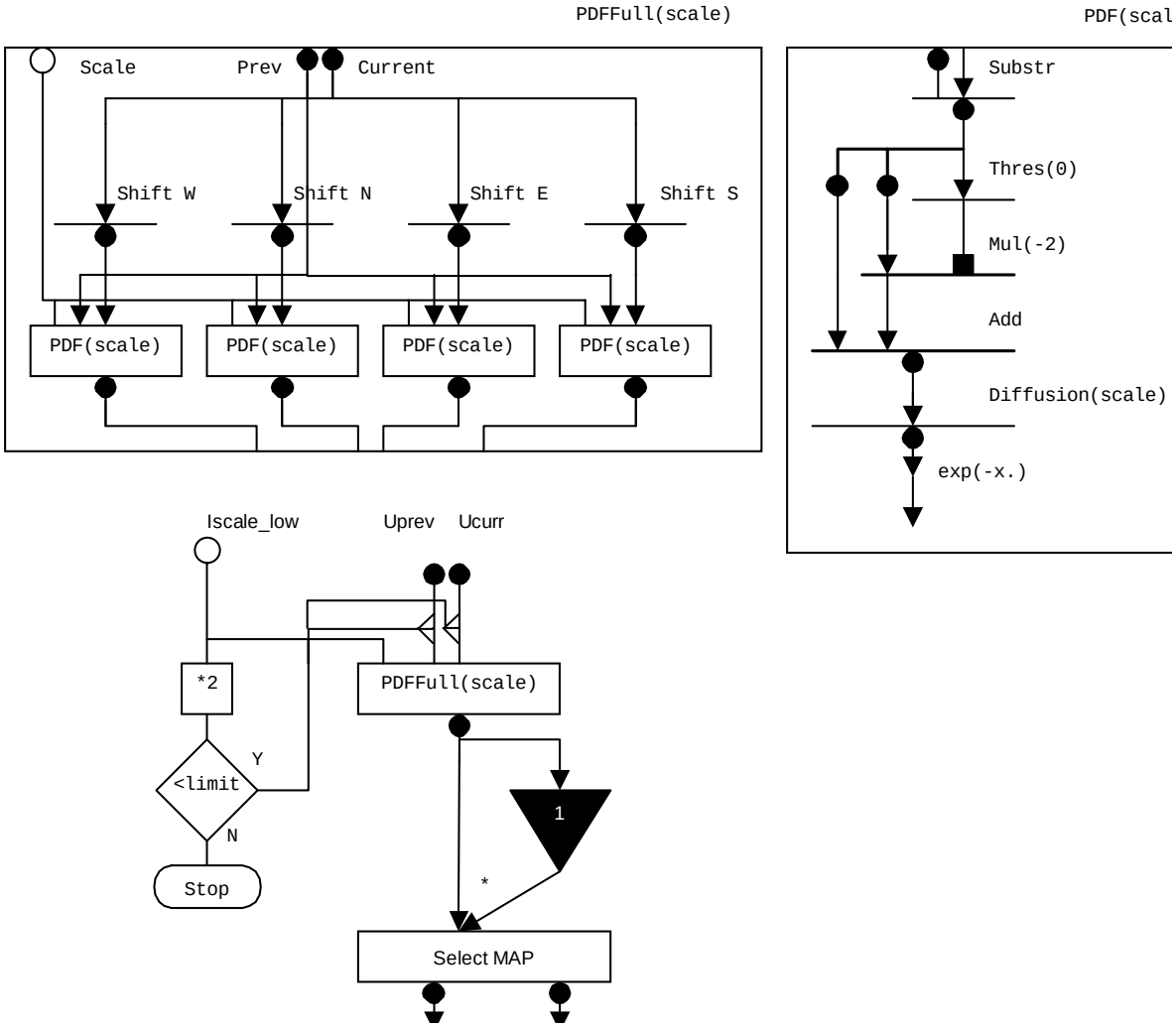
Input Parameters

U	Input image
Ucurr	The second frame of the pair
Uprev	The first frame of the pair

Output Parameters

Y	Output image
Y_PDFFull_x	X coordinate of the most likely motion vectors
Y_PDFFull_y	Y coordinate of the most likely motion vectors

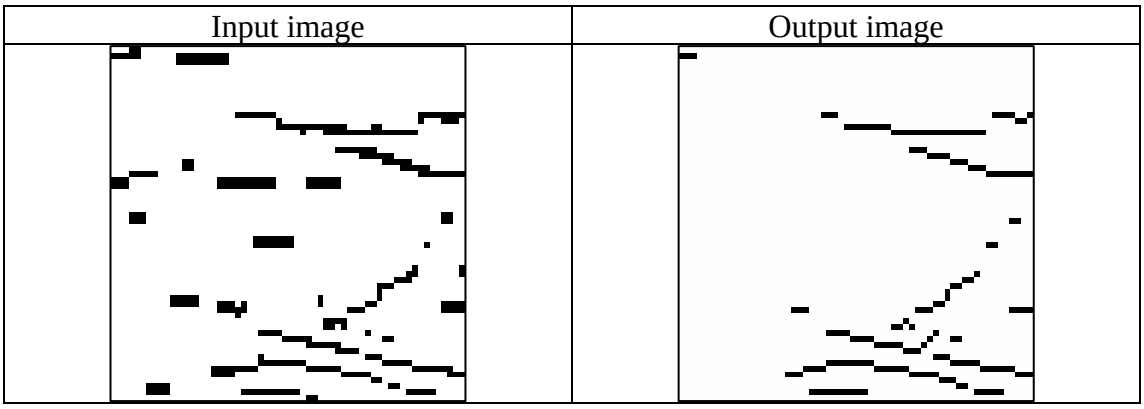
UMF diagram


PDFFull(scale)
PDF(scal
Scale
Prev
Current
Shift W
Shift N
Shift E
Shift S
PDF(scale)
PDF(scale)
PDF(scale)
PDF(scale)
Substr
Thres(0)
Mul(-2)
Add
Diffusion(scale)
exp(-x.)
Iscale_low
Uprev
Ucurr
*2
PDFFull(scale)
Y
<limit
N
Stop
1
*
Select MAP

BROKEN LINE CONNECTOR

Task description and algorithm

This algorithm fill gaps between neighboring object which onsets or offsets is closer than a given threshold [72]. The size of filled gaps can be controlled through the iteration of dilation template operation.



Input Parameters

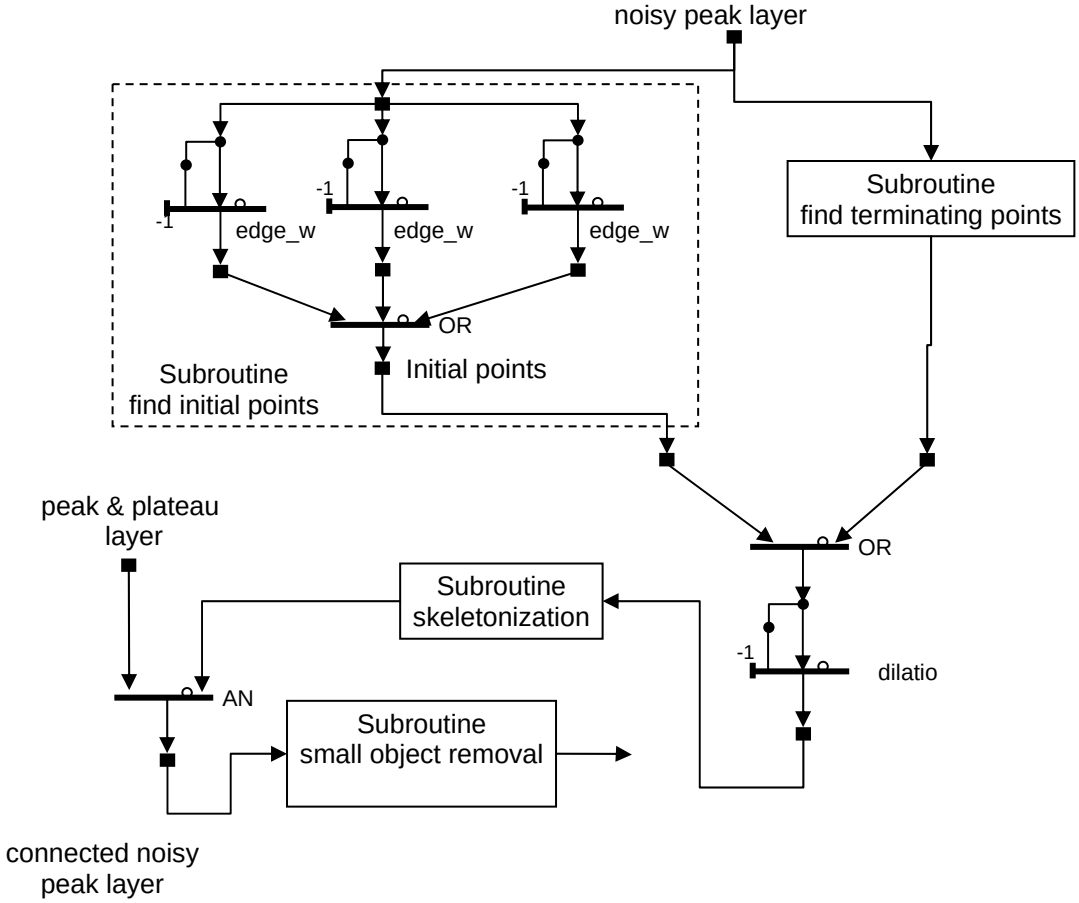
noisy peak layer	Input image

Output Parameters

Connected peak layer	Connected objects

Remarks

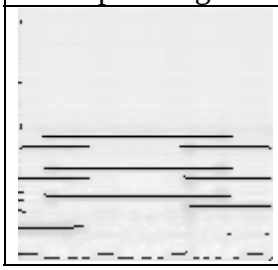
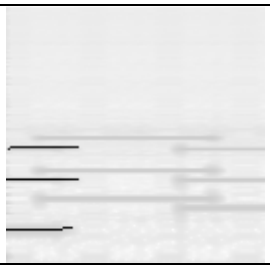
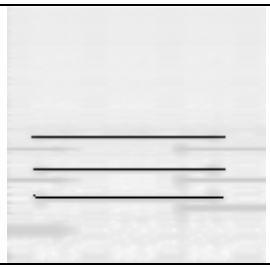
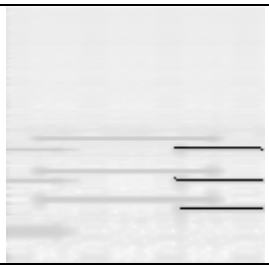
UMF diagram


noisy peak layer
Subroutine
find terminating points
-1
edge_w
-1
edge_w
-1
edge_w
OR
Initial points
Subroutine
find initial points
peak & plateau
layer
OR
Subroutine
skeletonization
-1
dilatio
AN
Subroutine
small object removal
connected noisy
peak layer

COMMON AM

Task description and algorithm

The algorithm extracts frequency bands to a distinct layer which onsets and offsets are synchronized [72]. This algorithm produces similar groups like the common amplitude-modulation group that was observed by psychoacoustic experiments of the human hearing.

Input image	Y1	Y20	Y 87
			

Input Parameters

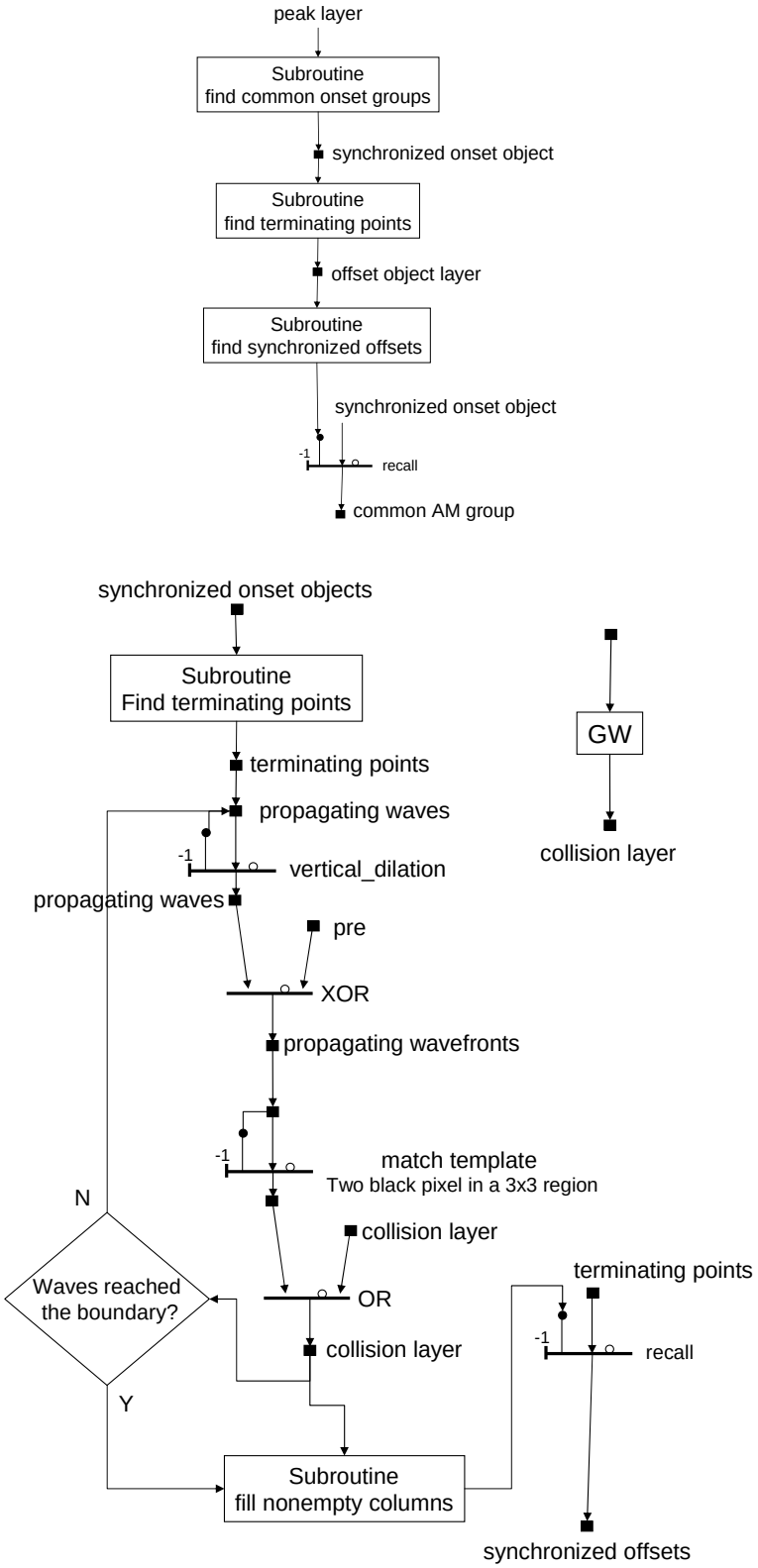
peak layer	Layer containing the representative frequency bands (peak layer)

Output Parameters

common AM layer	Bands with synchronized onset and offset from

Remarks
The time tolerance of offsets can be controlled through the iteration number of the vertical wave propagation step producing the *propagating waves* layer.

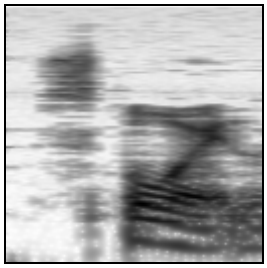
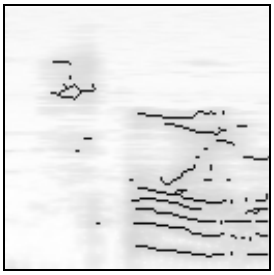
UMF diagram


peak layer
Subroutine
find common onset groups
synchronized onset object
Subroutine
find terminating points
offset object layer
Subroutine
find synchronized offsets
synchronized onset object
-1
recall
common AM group
synchronized onset objects
Subroutine
Find terminating points
terminating points
propagating waves
-1
vertical_dilation
propagating waves
pre
XOR
propagating wavefronts
-1
match template
Two black pixel in a 3x3 region
N
collision layer
Waves reached
the boundary?
OR
collision layer
Y
Subroutine
fill nonempty columns
terminating points
-1
recall
synchronized offsets
GW
collision layer

FIND OF COMMON FM GROUP

Task description and algorithm

Psychoacoustic experiments shows that human hearing system handles object with common frequency modulation as coming from the same source. This algorithm finds objects that is modulated by the same frequency i.e. their vertical distance is constant [72].

Original image	Representative frequency bands (input image)
	
Reference curve	Common FM group
	

Input Parameters

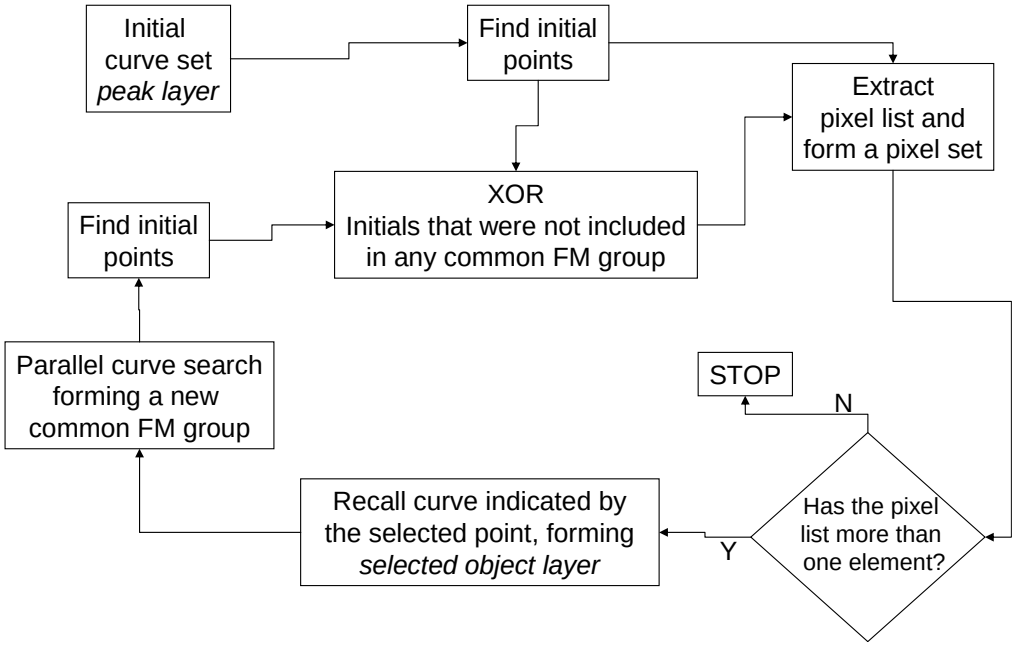
peak layer	Input image with lines in which some of them vertical distance is constant. (parallel curve)

Output Parameters

common FM group	Curves parallel to a given reference

Remarks
The registration of curves is processed by a digital computer. The digital computer selects the reference curves which initials are still in the maintained pixel list.

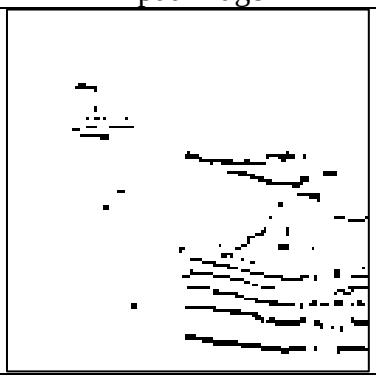
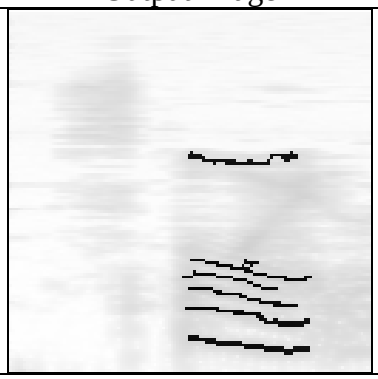
UMF diagram


Initial
curve set
peak layer
Find initial
points
Extract
pixel list and
form a pixel set
XOR
Initials that were not included
in any common FM group
Find initial
points
Parallel curve search
forming a new
common FM group
STOP
N
Has the pixel
list more than
one element?
Y
Recall curve indicated by
the selected point, forming
selected object layer

FIND COMMON ONSET/OFFSET GROUPS

Task description and algorithm

This algorithm produces series of layers containing objects with synchronized onset [72].

Input image	Output image
	

Input Parameters

peak layer	Input image

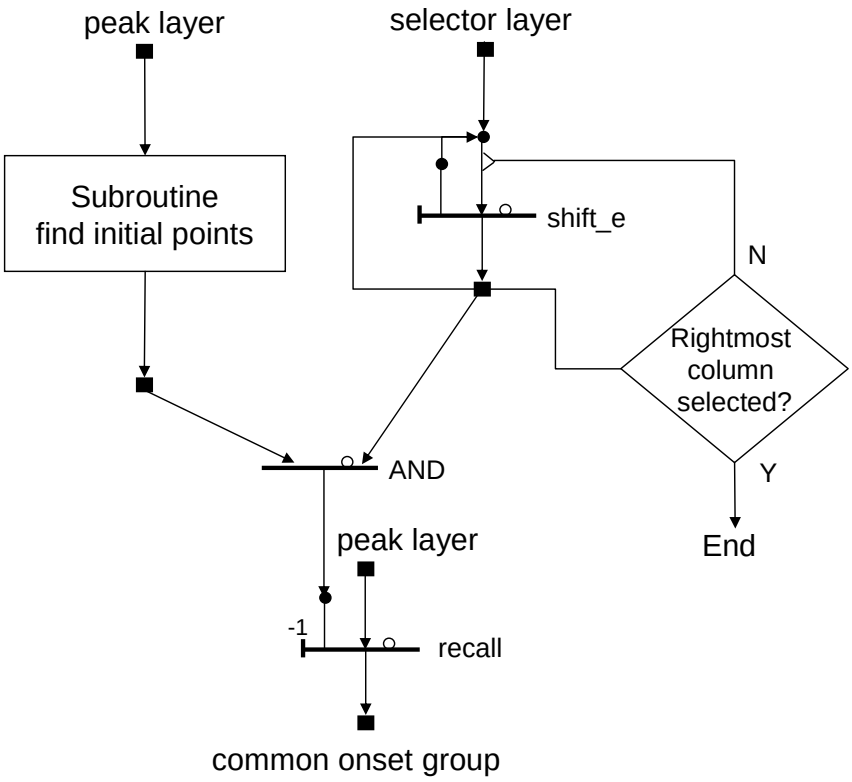
Output Parameters

common onset group	Object with synchronized onset

Remarks

Tolerance of time asynchrony is set by the width of vertical line shifted on the *selector layer*.

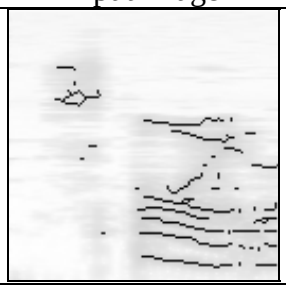
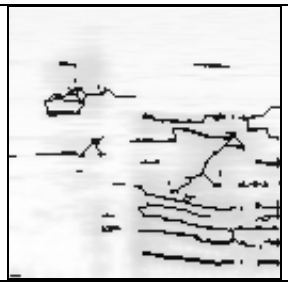
UMF diagram


peak layer
selector layer
Subroutine
find initial points
shift_e
N
Rightmost
column
selected?
Y
End
AND
peak layer
-1
recall
common onset group

CONTINUITY

Task description and algorithm

This algorithm connects object whose ending and initial is closer than a given threshold [72].

Input image	Connected curves
	

Input Parameters

peak layer	Input image

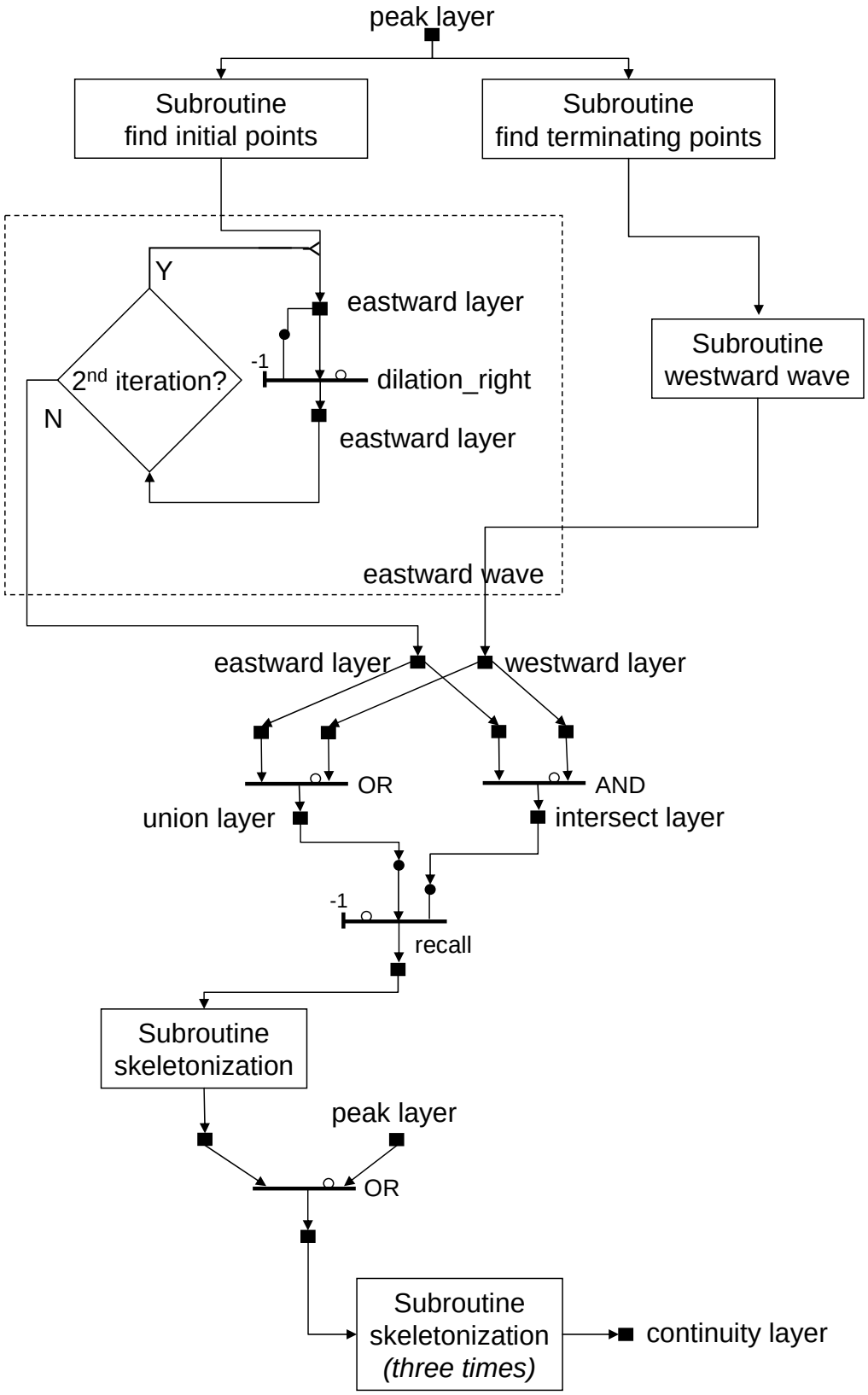
Output Parameters

continuity layer	Connected curves

Remarks

The spanned distance depends on the iteration of *dilation_right* template.

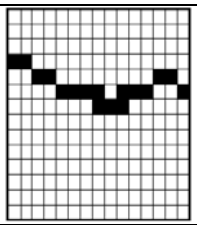
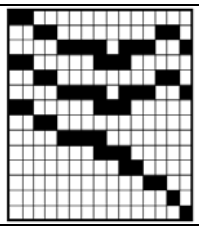
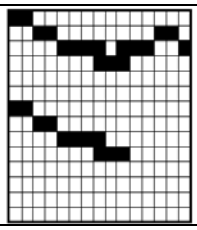
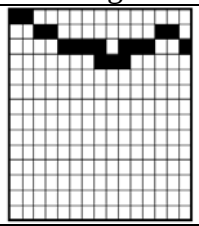
UMF diagram


peak layer
Subroutine
find initial points
Subroutine
find terminating points
Y
eastward layer
2nd iteration?
-1
dilation_right
N
eastward layer
Subroutine
westward wave
eastward wave
eastward layer
westward layer
OR
AND
union layer
intersect layer
-1
recall
Subroutine
skeletonization
peak layer
OR
Subroutine
skeletonization
(three times)
continuity layer

PARALLEL CURVE SEARCH

Task description and algorithm

This algorithm finds parallel curves to a selected one from an initial curve set [72].

Reference curve	Initial curve set	Output of the pixel-search step to a given distance	Output of the *keep almost complete curves* algorithm
			

Input Parameters

peak layer	Peak layer
selected object layer	Reference curve

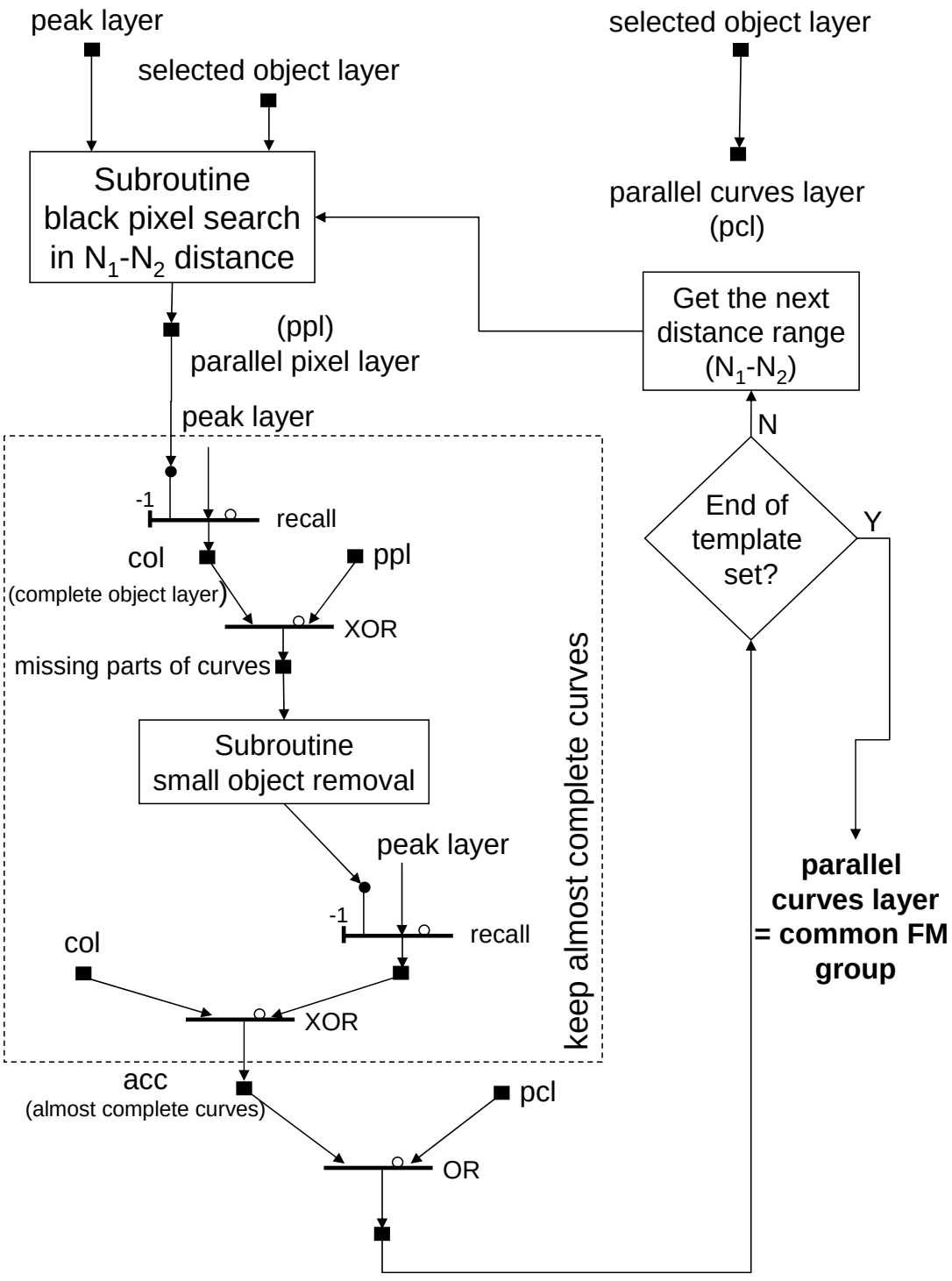
Output Parameters

parallel curves layer	Curves parallel to the reference

Remarks

The search region is controlled through the black pixel search template class.

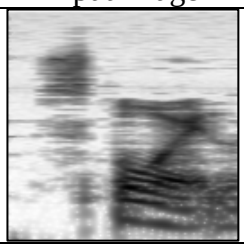
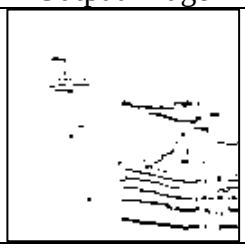
UMF diagram


peak layer
selected object layer
Subroutine
black pixel search
in N1-N2 distance
(ppl)
parallel pixel layer
peak layer
-1
recall
col
(complete object layer)
ppl
XOR
missing parts of curves
Subroutine
small object removal
peak layer
-1
recall
col
XOR
acc
(almost complete curves)
pcl
OR
keep almost complete curves
selected object layer
parallel curves layer
(pcl)
Get the next
distance range
(N1-N2)
N
End of
template
set?
Y
parallel
curves layer
= common FM
group

PEAK-AND-PLATEAU DETECTOR

Task description and algorithm

This algorithm detects horizontal the peaks and plateaus on a grayscale image [72].

Input image	Output image
	

Input Parameters

magnitude image	Input image

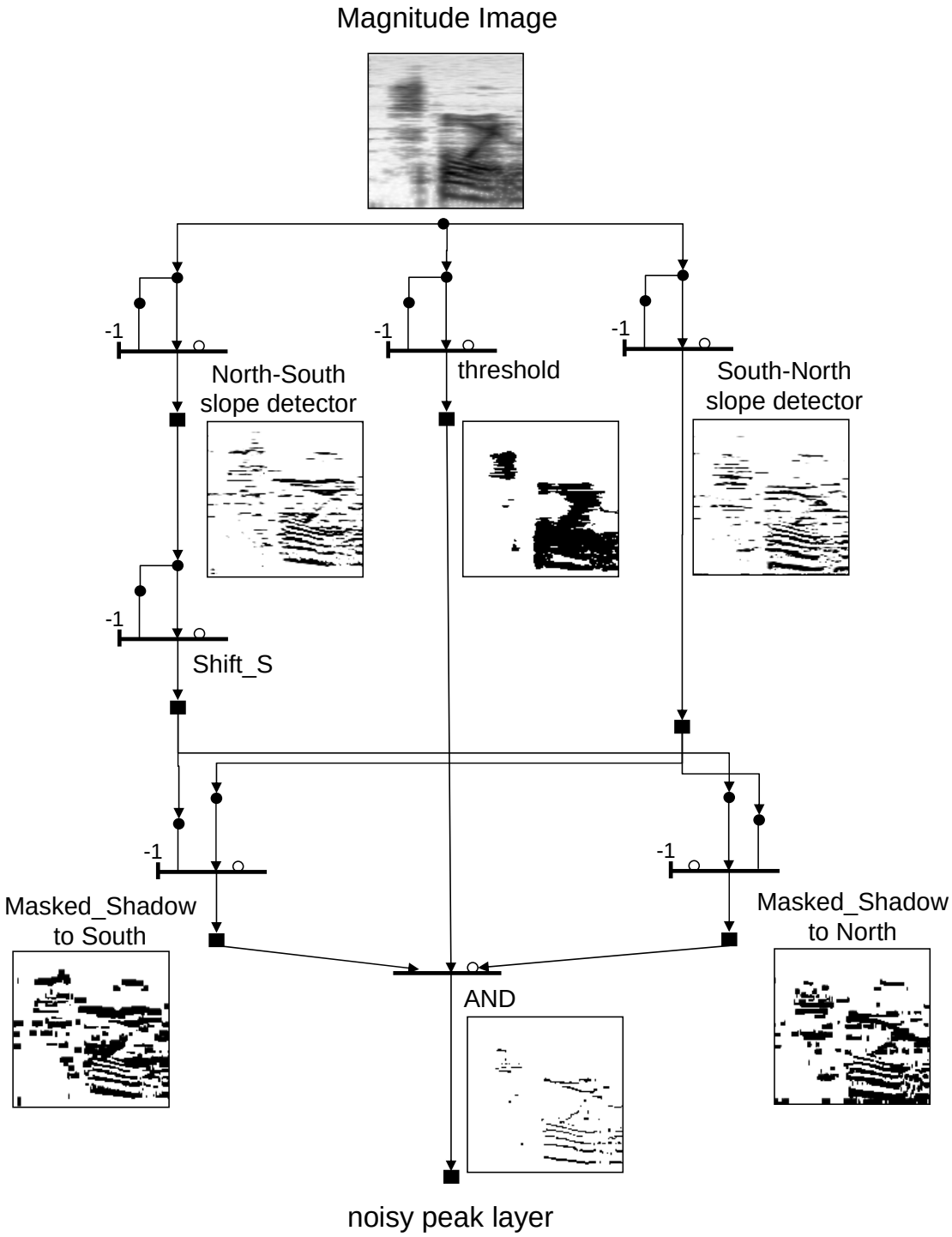
Output Parameters

noise peak layer	Peaks and plateaus

Remarks

There could be a few pixel wide gaps in the detected peaks and plateaus while vertical interconnecting pixels do not cause black pixels to emerge. This gaps can be filled by the *broken line connector* algorithm.

UMF diagram


Magnitude Image
-1
North-South
slope detector
-1
threshold
-1
South-North
slope detector
-1
Shift_S
-1
-1
Masked_Shadow
to South
Masked_Shadow
to North
AND
noisy peak layer

GLOBAL DISPLACEMENT DETECTOR

Task description and algorithm

This algorithm calculates the most probable global displacement vector of the input scene. The diagram shows the calculation steps for the vertical coordinate of the displacement vector. The horizontal coordinate can be calculated in an analogous way by exchanging vertical and horizontal directions.

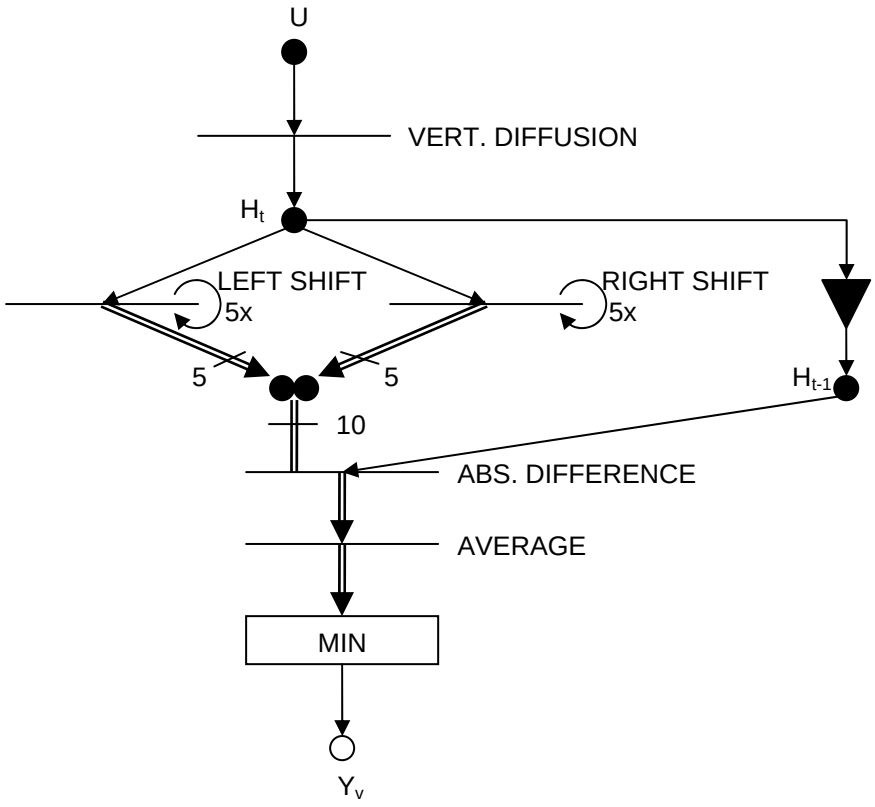
Input Parameters

U	Input image

Output Parameters

Y_h	Horizontal displacement
Y_v	Vertical displacement

UMF diagram


U
VERT. DIFFUSION
H_t
LEFT SHIFT
5x
RIGHT SHIFT
5x
5
5
H_{t-1}
10
ABS. DIFFERENCE
AVERAGE
MIN
Y_v

ADAPTIVE BACKGROUND AND FOREGROUND ESTIMATION

Task description and algorithm

This algorithm continuously estimates the background and foreground of a video flow.

Input image	Background	Foreground
		
		

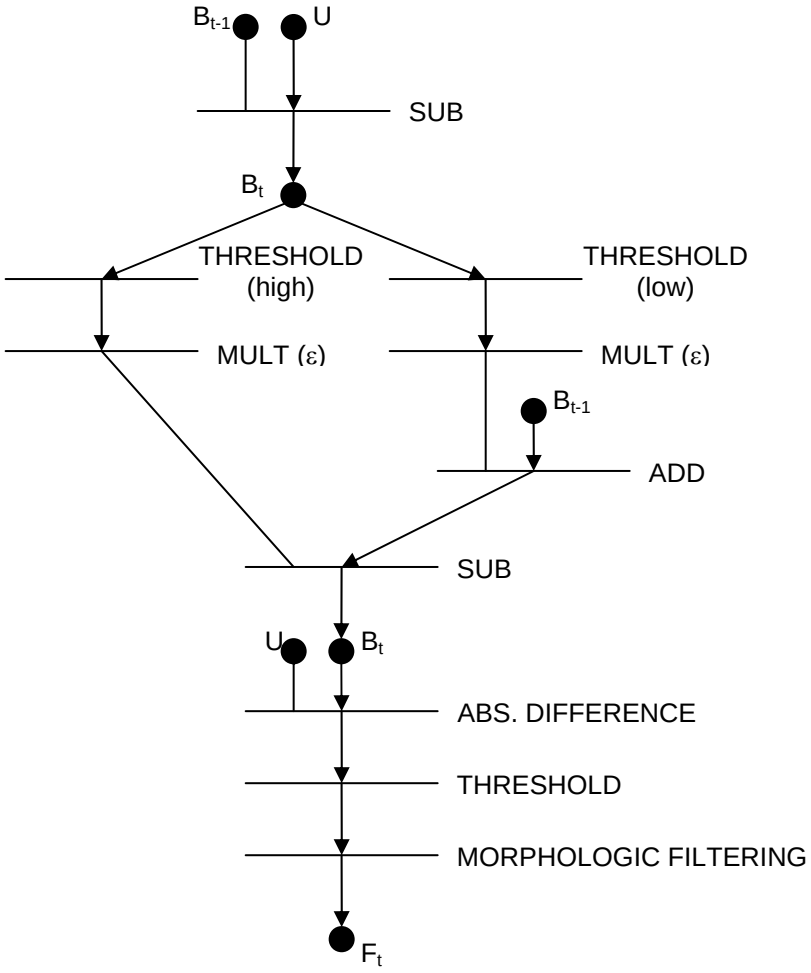
Input Parameters

U_t	Input frame at time t
B_{t-1}	Background estimation after previous frame

Output Parameters

B_t	Updated background estimation
F_t	Current foreground estimation

UMF diagram

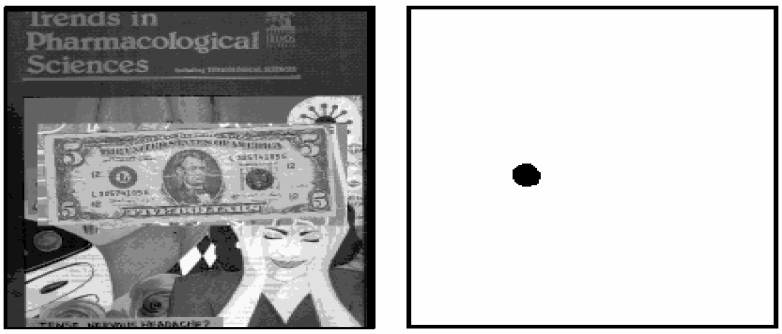

B_{t-1}
U
SUB
B_t
THRESHOLD (high)
THRESHOLD (low)
MULT (ε)
MULT (ε)
B_{t-1}
ADD
SUB
U
B_t
ABS. DIFFERENCE
THRESHOLD
MORPHOLOGIC FILTERING
F_t

Chapter 3. PROGRAMS

BANK-NOTE RECOGNITION

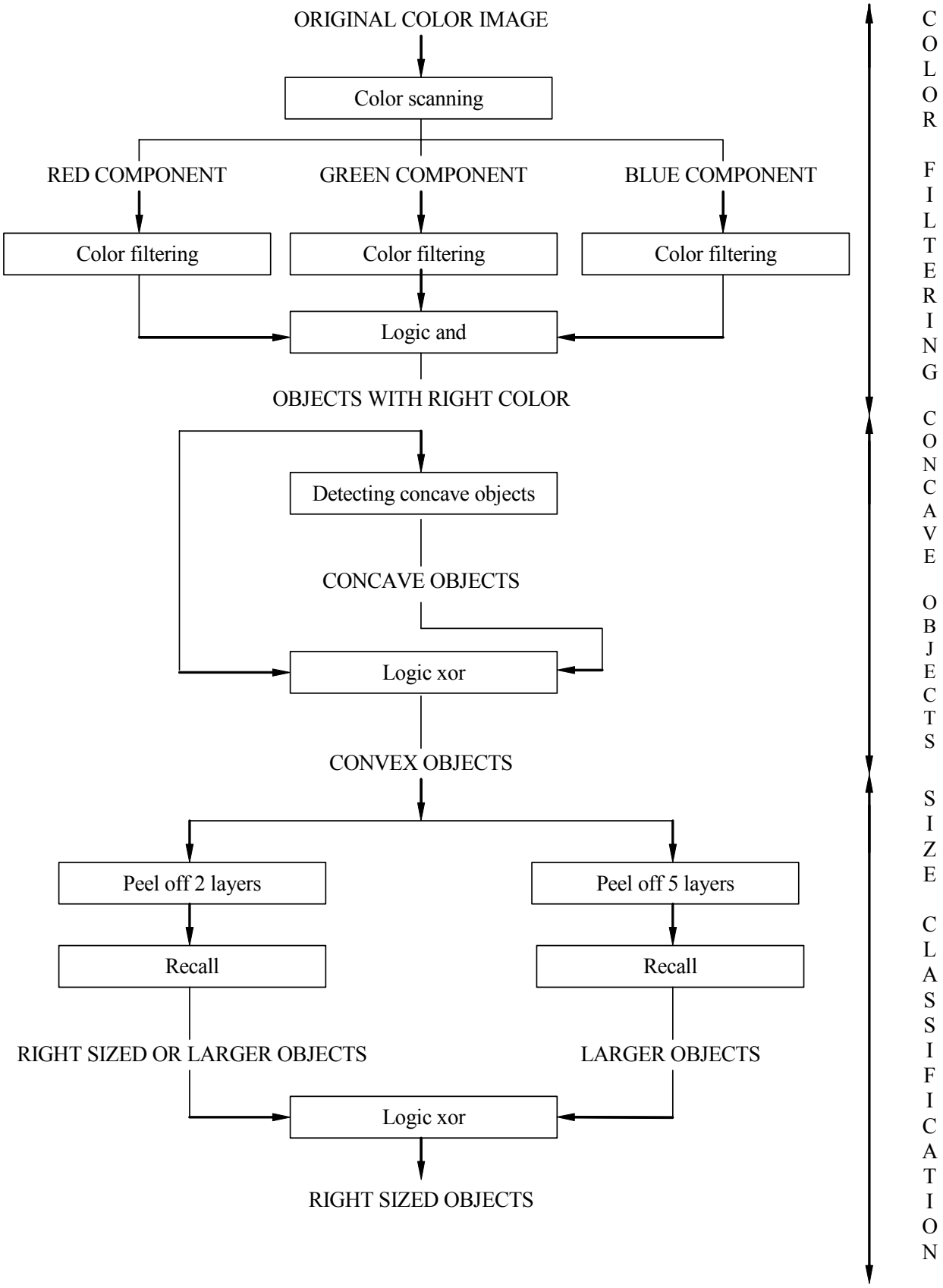
This algorithm identifies American bank-notes on color images. The bank-notes can be in the image with arbitrary offset and rotation. The positioned algorithm finds the green and black circles common to all US bank-notes. It analyses color, shape and size. The algorithm can be separated into three parts. These parts are indicated in the flow-chart. The detailed description can be found in [22]. The templates can be found in this template library.

*Example***:** A grayscale version of a color input image, and the extracted black circle.



The flow-chart of the algorithm:

This chart contains only half of the algorithm, and finds only one circle. The other circle can be found with a similar method, but with different parameters in the color filtering.


ORIGINAL COLOR IMAGE
Color scanning
RED COMPONENT
GREEN COMPONENT
BLUE COMPONENT
Color filtering
Color filtering
Color filtering
Logic and
OBJECTS WITH RIGHT COLOR
Detecting concave objects
CONCAVE OBJECTS
Logic xor
CONVEX OBJECTS
Peel off 2 layers
Peel off 5 layers
Recall
Recall
RIGHT SIZED OR LARGER OBJECTS
LARGER OBJECTS
Logic xor
RIGHT SIZED OBJECTS
COLOR FILTERING
CONCAVE OBJECTS
SIZE CLASSIFICATION

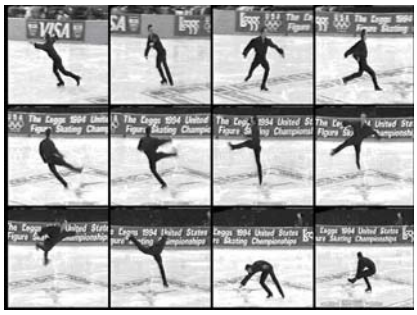
CALCULATION OF A CRYPTOGRAPHIC HASH FUNCTION [37]

Short Description

The CNN analogic program described here performs the basic operations needed to calculate a hash value, when a set of binary images and a key vector are given. The current version of the Alpha compiler cannot interpret sequences of key bits or image sequences, therefore the Alpha source code listed below contains only two images and two key bits.
The first image is loaded to the chip, and its columns (as binary vectors) are multiplied by the key vector. This multiplication is performed as a sequence of shift-add operations. Then, the next binary image is added to modulo 2, and the multiplication is performed once more.

Typical Example

Gray-scale images (or video sequences) must be quantized and cut into pieces according to the chip size. The following pictures show an input sequence and a typical hash result; the latter, of course, heavily depends on the key bits.

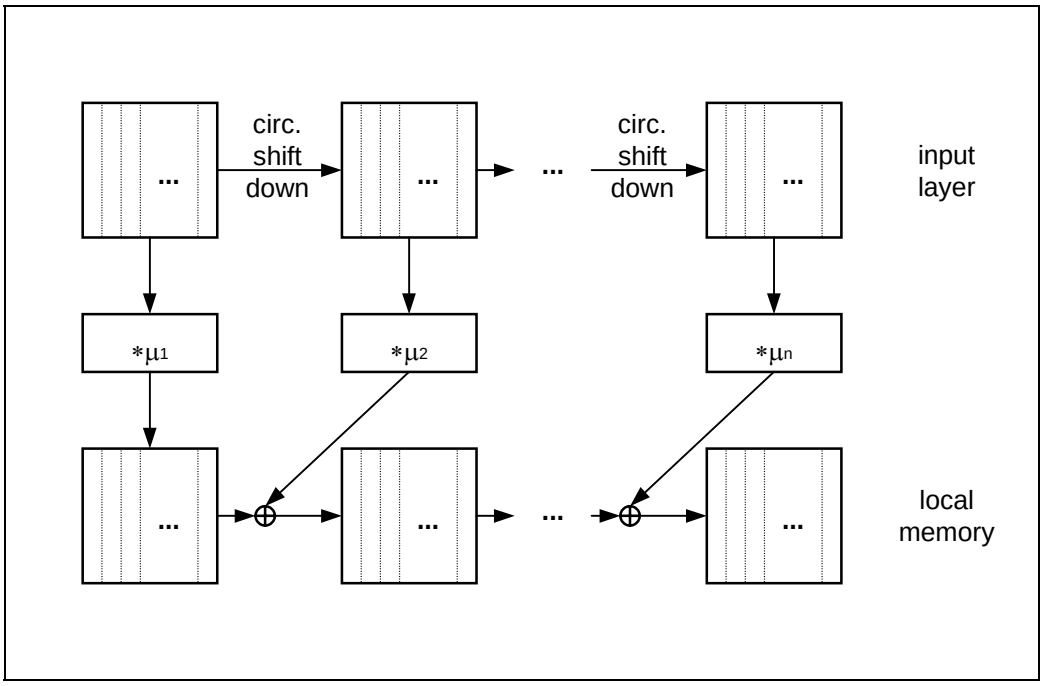


INPUT SEQUENCE (12x256x256x8 bits)

HASH RESULT (21x20 bits)

Flow-diagram of the algorithm

The following diagram shows the CNN implementation of the vector multiplication needed in the hashing process:



Templates used in the algorithm

The algorithm uses only a vertical shift operation and local logic. The shift operation is performed via the following template:

SHIFTSOU template:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 2 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = | 0 |

DETECTION OF MAIN CHARACTERS

Short Description

The algorithm has three major steps. In the first step, only a part of the noise is discarded, but the main features are coming out fine. In the second step a more aggressive filter were applied. After this step, only some parts of the largest objects remained on the image. In the last step, the main characters are reconstructed from the previous two results.

Typical Example

The speciality of this example is that the input image was stored in an extremely high-density optical memory [40]. It is corrupted with noise heavily.

In this example, the image size is 318x93. This image was automatically cut to about 300 20x22 image tiles and processed one after the other on the chip. The algorithm was executed on the 20x22 CNN chip [41]. Experimental results of the main feature extractor algorithm are as follows.

(a)

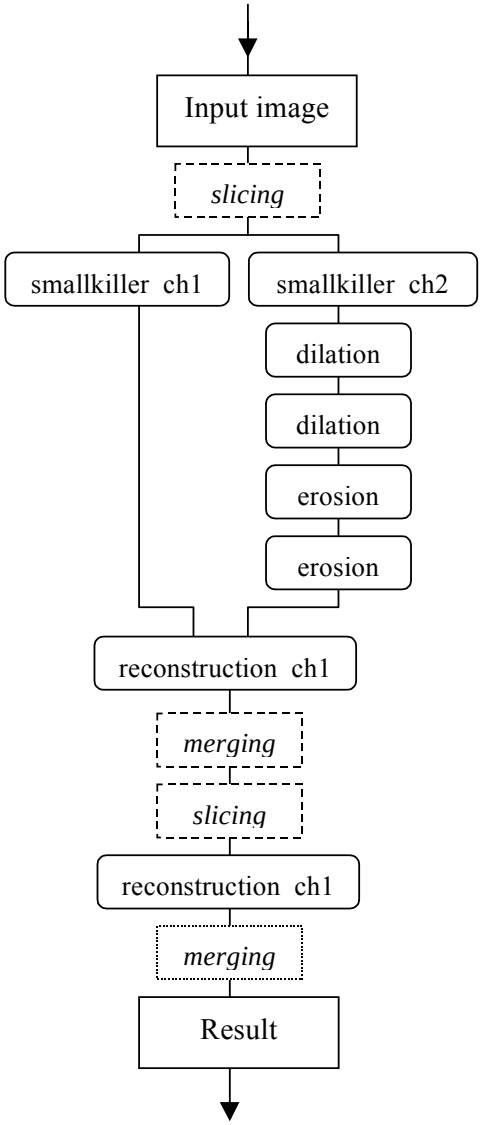
(b)

(c)

(d)

Block Diagram of the Algorithm

The flowchart of the main feature extractor algorithm is as follows.



Templates used in the algorithm

SMALLKILLER_CH1:

A = | 1 | 1 | 1 |
| 1 | 2 | 1 |
| 1 | 1 | 1 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = -1.7

SMALLKILLER_CH2:

A = | 1 | 1 | 1 |
| 1 | 2 | 1 |
| 1 | 1 | 1 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = -2

DILATION:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

z = 4.5

EROSION:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

z = -5.5

RECONSTRUCTION_CH:

A = | 0 | 1 | 0 |
| 1 | 3 | 1 |
| 0 | 1 | 0 |

B = | 0 | 0 | 0 |
| 0 | 3 | 0 |
| 0 | 0 | 0 |

z = -1.25

FAULT TOLERANT TEMPLATE DECOMPOSITION [49, 50]

Short Description

An idea of fault tolerant template decomposition in the case of local boolean operators (binary input/output templates) will be outlined here. Due to parameter deviations of current analog VLSI implementations, templates generated theoretically do not work properly. A solution to this problem is applying a sequence of so called fault tolerant templates, that "make no faults". Here two examples of such a decomposition will be presented: the Local Concave Place (LCP) detector template and the JUNCTION template from this template library will be decomposed into a sequence of two fault tolerant templates.

Typical Examples

Example 1: LocalConcavePlaceDetector (LCP) template decomposition

LCP:

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 2 | 2 | 2 |
| 1 | -2 | 1 |

$z =$ | -5 |
|---|

Minimized form of the function: $F(u) = u_6 u_5 u_4 \bar{u}_2 (u_1 + u_3)$

Sub-functions chosen: $F_1(u) = u_6 u_5 u_4 \bar{u}_2$ $F_2(u) = u_1 + u_3$

Fault tolerant template sequence:

LCP1:

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 1 | 1 | 1 |
| 0 | -1 | 0 |

$z =$ | -3 |
|---|

LCP2:

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

$z =$ | 1 |
|---|

Result: *LCP* $\Leftrightarrow$ *LCP1* ***AND*** *LCP2,*
where $\rho(\text{LCP}) = 0.24$, while $\rho(\text{LCP1}) = 0.5$ and $\rho(\text{LCP2}) = 0.71$.

Example 2: Decomposition of the JunctionExtractor template

JunctionExtractor:

$\mathbf{A} =$ | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$\mathbf{B} =$ | 1 | 1 | 1 |
|---|---|---|
| 1 | 6 | 1 |
| 1 | 1 | 1 |

$z =$ | -3 |
|---|

Minimized form of the function: $F(u) = u_5 (u_1 u_2 u_3 + u_1 u_2 u_4 + u_1 u_2 u_5 + ... + u_6 u_8 u_9 + u_7 u_8 u_9)$

Sub-functions chosen: $F_1(u) = u_5$

$$F_2(u) = u_1u_2u_3 + u_1u_2u_4 + u_1u_2u_5 + ... + u_6u_8u_9 + u_7u_8u_9$$

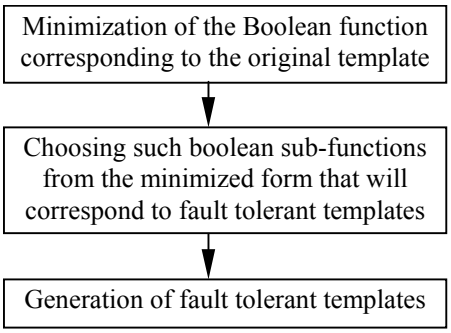
Fault tolerant template sequence:

JUNC1:

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

Result: *JunctionExtractor* $\Leftrightarrow$ INPUT ***AND*** *JUNC1*
where ρ(JunctionExtractor) = 0.15, and ρ(JUNC1) = 0.35 .

Block Diagram of the Algorithm



Templates used in the algorithm

Templates used in examples are shown in the “Typical Examples” session.

ALPHA source

Alpha source of Example1

```
PROGRAM LCP (input; output);
/* DECOMPOSITION OF THE LOCAL CONCAVE PLACE DETECTOR TEMPLATE */
/* BY USING TWO FAULT TOLERANT TEMPLATES */
CONSTANT
  White = -1.0;
  ONE = 1;
  TWO = 2;
  TIME = 5;
ENDCONST;
/* Chip set definition section */
CHIP_SET simulator.eng;
A_CHIP
SCALARS
IMAGES
 LLM1: BINARY;
 LLM2: BINARY;
 LLM3: BINARY;
ENDCHIP;
E_BOARD
SCALARS
```

```
IMAGES
 P: BINARY;
ENDBOARD;

OPERATIONS FROM LCP.tms;
PROCESS LCP_DECOMP;
  USE (lcp1, lcp2);
  HostLoadPic(input, P);
  HostDisplay(P, ONE);
  LLM1 := P;
  lcp1 (LLM1, LLM1, LLM2, TIME, White);
  lcp2 (LLM1, LLM1, LLM3, TIME, White);
  LLM1 := LLM3;
  LLM1 := LLM1 AND LLM2;
  P := LLM1;
  HostDisplay(P, TWO);
  HostSavePic(output, P);
ENDPROCESS;
ENDPROG;
```

Alpha source of Example2

```
PROGRAM JUNCTION (input; output);
/* DECOMPOSITION OF THE JUNCTION TEMPLATE */
/* BY USING ONE FAULT TOLERANT TEMPLATE AND A LOGIC OPERATION */
CONSTANT
  White = -1.0;
  ONE = 1;
  TWO = 2;
  TIME = 5;
ENDCONST;
/* Chip set definition section */
CHIP_SET simulator.eng;
A_CHIP
SCALARS
IMAGES
 LLM1: BINARY;
 LLM2: BINARY;
 LLM3: BINARY;
ENDCHIP;
E_BOARD
SCALARS
IMAGES
 P: BINARY;
ENDBOARD;
OPERATIONS FROM junction.tms;
PROCESS JUNCTION_DECOMP;
  USE (junc1);

  HostLoadPic(input, P);
```

```
  HostDisplay(P, ONE);
  LLM1 := P;
  junc1 (LLM1, LLM1, LLM2, TIME, White);
  LLM1 := LLM1 AND LLM2;
  P := LLM1;
  HostDisplay(P, TWO);
  HostSavePic(output, P);
ENDPROCESS;
ENDPROG;
```

Comments

Fault tolerant template generation results in a sequence of "reliable" templates.

GAME OF LIFE

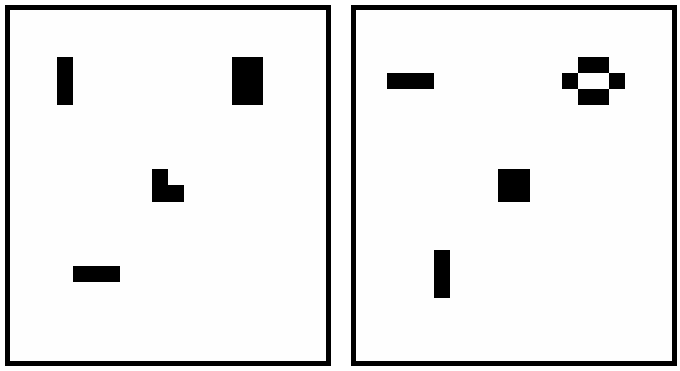
Short Description

The following simple algorithm simulates the Game of Life. Both input and output pictures are binary. Rules of the game:

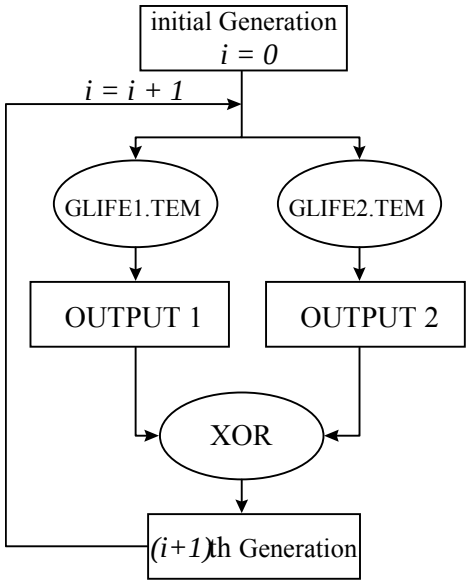
- a black pixel turns white if it has *more than three* or *less than two* black neighbors.
- a white pixel turns black if it has *exactly three* black neighbors.

Typical Example

The following example shows two consecutive generations of the Game of Life simulated by CNN.



Block Diagram of the Algorithm



Templates used in the algorithm

The corresponding templates can be found in this template library as LIFE_1.

ALPHA source

```
PROGRAM GAMEOFLIFE (inputFC);
/* THIS PROGRAM SIMULATES TEN CYCLES OF THE GAME OF LIFE  */

CONSTANT                                /* constant declaration */
```

```
   NULL = 0;
   NUM_GEN = 9;                          /* Number of generations */
   ONE = 1;
   WHITE = -1.0;
   TIME = 5;
   TIMESTEP = 0.5;
ENDCONST;
CHIP_SET simulator.eng;
A_CHIP
SCALARS
IMAGES
  L1: BINARY;          /* LLM1 */
  L2: BINARY;                 /* LLM2 */
  L3: BINARY;                 /* LLM3 */
  L4: BINARY;                 /* LLM4 */
ENDCHIP;
E_BOARD
SCALARS
 var: INTEGER;
IMAGES
 LargeInp: BINARY;
 LargeOut: BINARY;
ENDBOARD;
OPERATIONS FROM gameoflife.tms;
FUNCTION game_of_life;
 USE (glife1, glife2);
   L4 := NULL;                                 /* Zero state */
   glife1 (L3, L4, L1, TIME, WHITE);           /* Template1 execution */
   glife2 (L3, L4, L2, TIME, WHITE);           /* Template2 execution */
   L4 := L1 XOR L2;                      /* XOR operation */
ENDFUNCT;
PROCESS freichen;                        /* here starts the main routine */
 USE ();
  SwSetTimeStep (TIMESTEP);
  HostLoadPic(inputFC, LargeInp);
  L4 := LargeInp;
  REPEAT var := 1 TO NUM_GEN BY 1;
   L3 := L4;                             /* Reload i-th genaration to input */
   game_of_life;                         /* Simulating one generation of the Game of Life */
   LargeOut := L4;                       /* copying the image from chip to board */
   HostDisplay(LargeOut, ONE);
  ENDREPEAT;
 ENDPROCESS;
ENDPROG;
```

Comments

The templates can be found by using the TemMaster [38] template design software package.

HAMMING DISTANCE COMPUTATION

In the theory of information processes it is a common problem that, given a code received in a noisy channel and the set of legal code words, we have to determine the code word nearest in some metric to the received one. In the case of binary codes the Hamming distance is the most common choice to measure the distance. Here a 4-step method is given presented, which selects the legal code closest to the input.

The first step compares the input to all legal code words. In order for this to happen, the *m* legal code words should be put in a single image, each code being a separate row, while the input should be written in another image *m* times. This step can be performed by the logic **XOR** template. In the second step the number of differences is calculated, after feeding the output of the previous step back to the input and setting the initial state to 0. In the third step the minimum distance is determined, and finally, the best matching code word is selected. For this to be realized, the output of the previous step should be used as initial state, and that of step 2 as input [20].

Templates:

Differences: *Min. distance:* *Best matching*

$\mathbf{A}_2 =$

0	0	0
0	0	1
0	0	0

$\mathbf{A}_3 =$

0	***b***	0
0	1	0
0	***b***	0

$\mathbf{A}_4 =$

0	0	0
0	2	0
0	0	0

$\mathbf{B}_2 =$

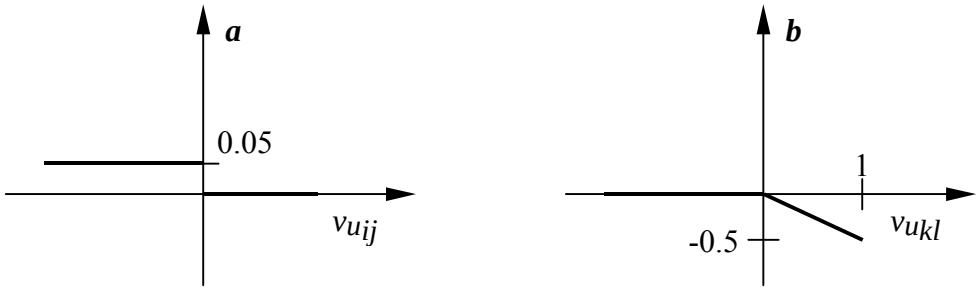
0	0	0
0	***a***	0
0	0	0

$\mathbf{B}_4 =$

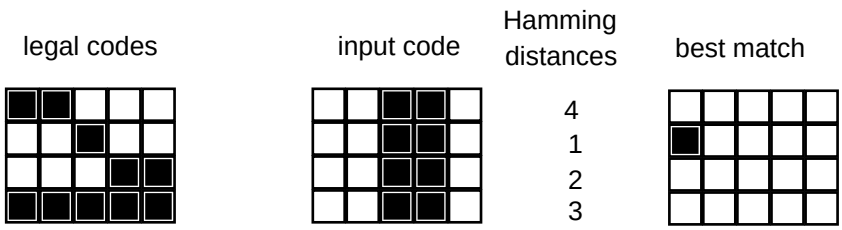
0	0	0
0	-1	0
0	0	0

$z_4 =$ 0.02

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



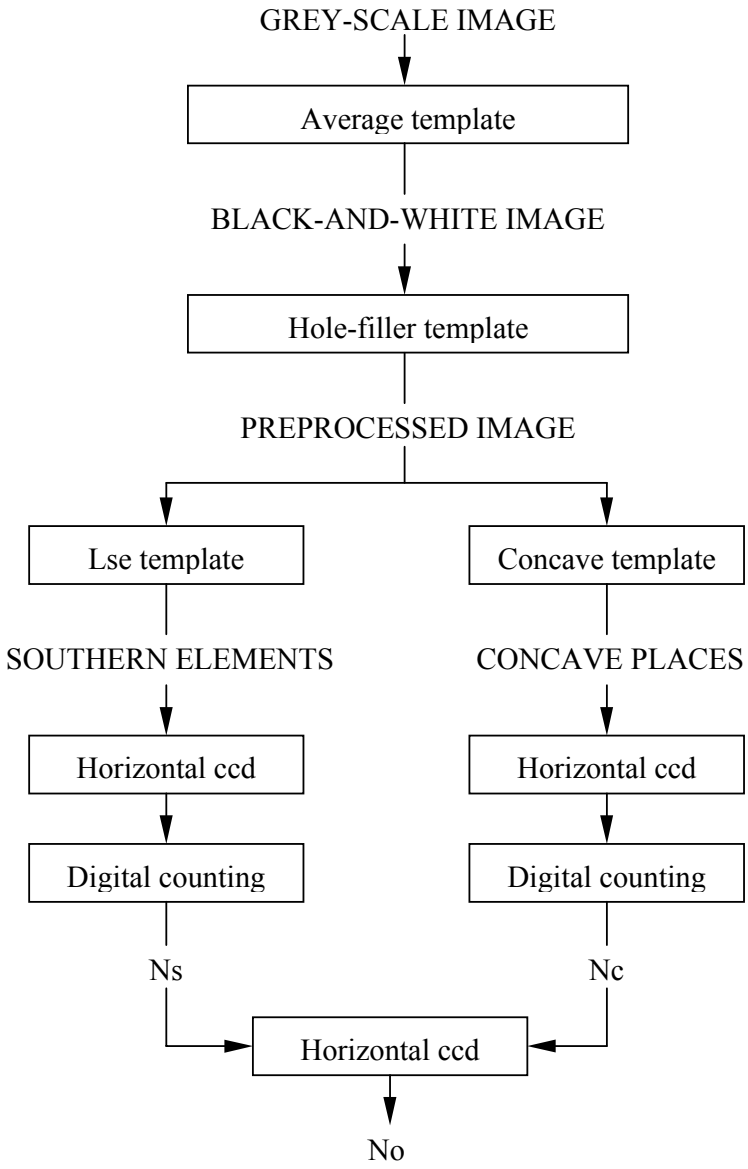
Example:



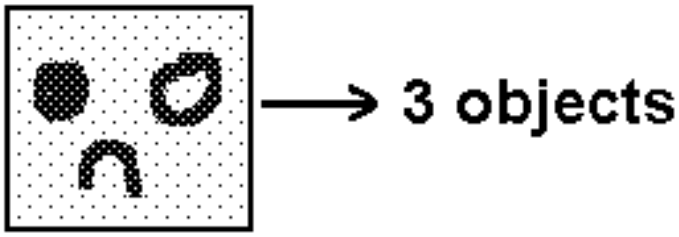
OBJECT COUNTING

This algorithm counts the connected objects on a grayscale image. The algorithm is detailed in [11]. The cited templates can be found in this template library.

The flow-chart of the algorithm:



Example: The algorithm is demonstrated on a grayscale image containing 3 objects. Image name: objcount.bmp; image size: 91x83.



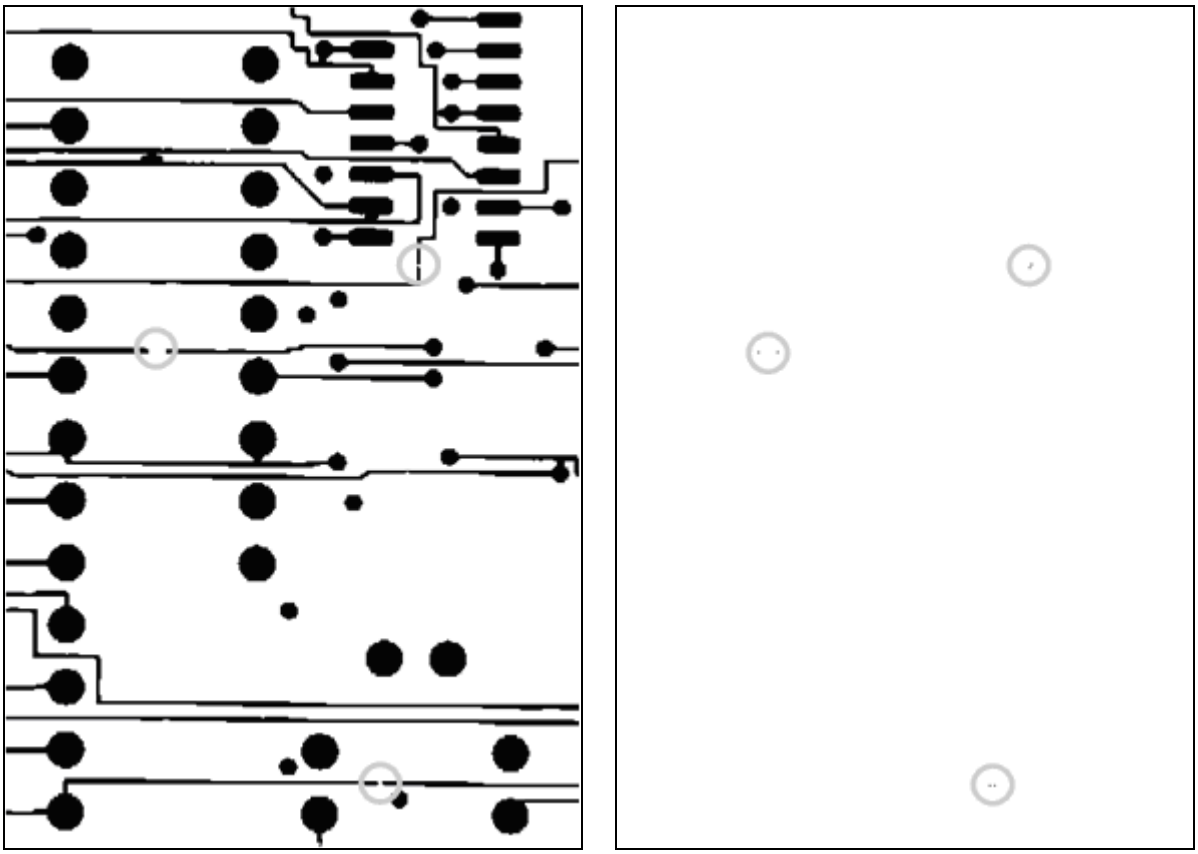
OPTICAL DETECTION OF BREAKS ON THE LAYOUTS OF PRINTED CIRCUIT BOARDS [39]

Short Description

The input images of a printed circuit board or an artwork film are scanned in by using a scanner. For detecting breaks, the basic idea of the algorithm is that a wire has to be terminated in a pad, in a via hole or in another wire.

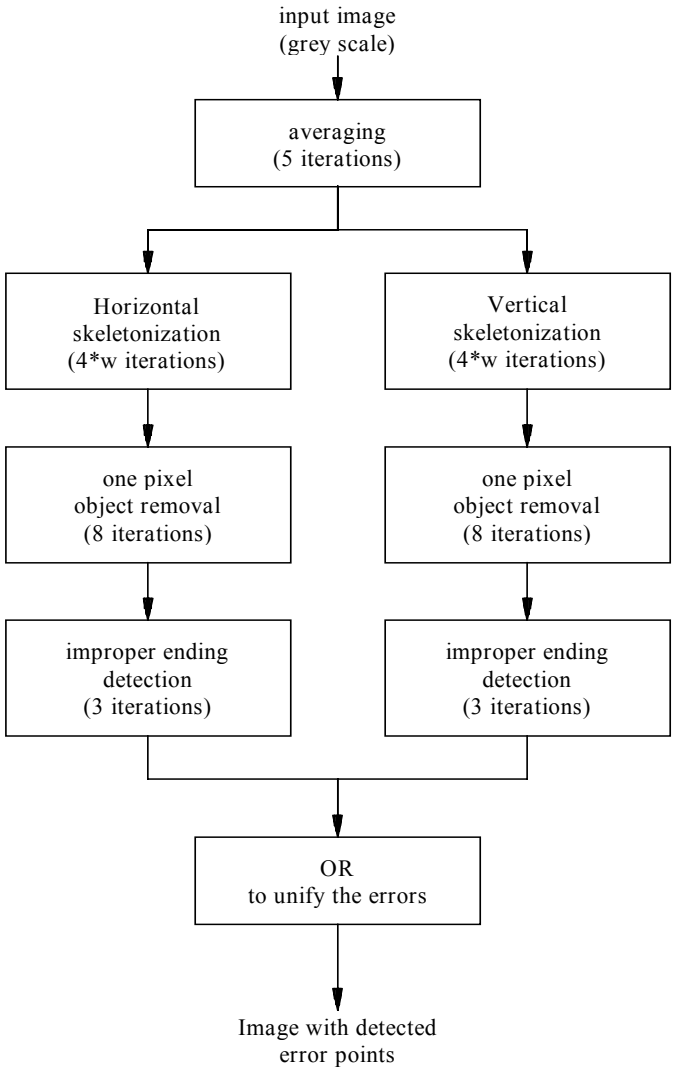
Typical Example

This example was run on the CNN Hardware Accelerator Board (CNN-HAB).



The input and result of the wire break detection analogic CNN algorithm

Flow-diagram of the algorithm



Templates used in the algorithm

The one-pixel-object removal and the OR templates can be found in this template library as *SmallObjectRemover* and *LogicOR*, respectively.

HorSkelL: horizontal skeleton from left

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.5 & 0 & 0.125 \\ 0.5 & 0.5 & -0.5 \\ 0.5 & 0 & 0.125 \end{bmatrix} \quad z = \boxed{-1}$$

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, peeling the black pixels from left of a wire

Remark:

The template *HorSkelR* (horizontal skeleton from right) can be obtained by rotating *HorSkelL* by 180°). The *VerSkelT* and *VerSkelB* templates (rotating the *HorSkelR* and *HorSkelL* templates by 90°) are used for horizontal line skeletonization.

DeadEndV: ***finds the endings of vertical wires***

A = | 0 | 0 | 0 |
|---|---|---|
| 0 | 3 | 0 |
| 0 | 0 | 0 |

B = | -0.25 | -0.25 | -0.25 |
|---|---|---|
| -0.25 | 0.5 | -0.25 |
| -0.25 | -0.25 | -0.25 |

z = | -5.8 |
|---|

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 of the endings of the vertical wires

Remark:

The *DeadEndH* templates (rotating the *DeadEndV* template by 90°) are used to detect the endings of horizontal wires.

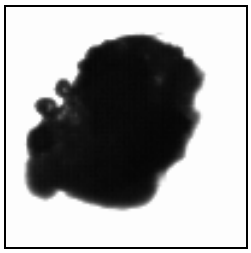
ROUGHNESS MEASUREMENT VIA FINDING CONCAVITIES [18]

Short Description

This simple CNN analogic program detects concavities of objects. This can be used for surface roughness measurement.
The basic idea here is to find the concave parts of objects. First, the gray-scale image is converted into a binary image via a thresholding operation. Next, pixels being located at concave places are driven into black by using the "hollow" template. This template turns black all those white pixels which have at least four black direct neighbors. Next, we extract concavities of objects by using the logical XOR operation between the thresholded image and filled image.

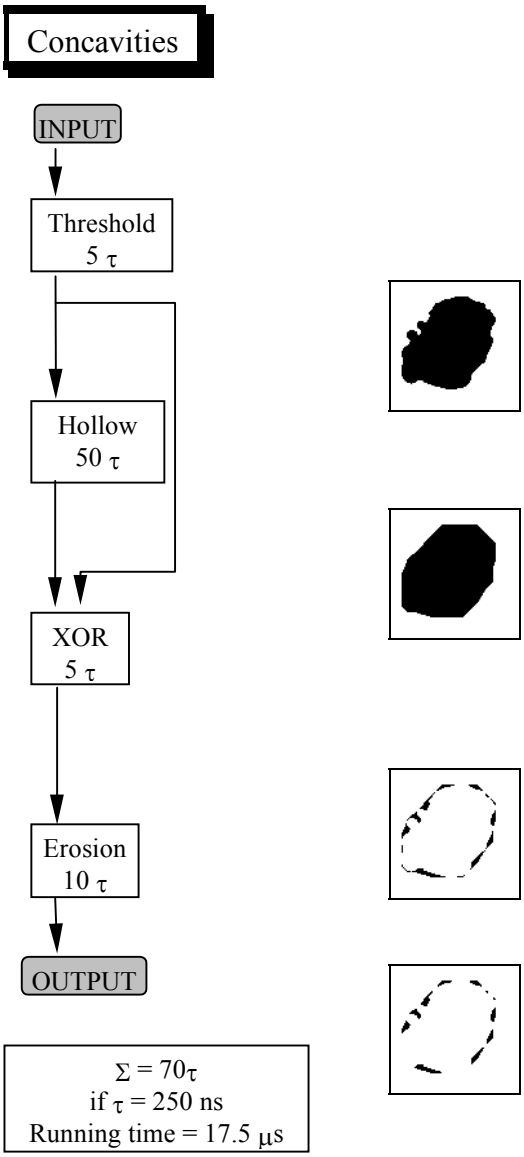
Typical Example

The following example shows the detected concave parts of an object.


INPUT


OUTPUT

Flow-diagram of the algorithm



Templates used in the algorithm

Templates can be found in this template library as *Threshold*, *ConcaveLocationFiller* (HOLLOW), and *Erosion*, respectively. The exact template values are presented below.

Threshold: converting gray-scale image to binary image

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

ConcaveLocationFiller: fills the concave locations of objects

This template turns black all those white pixels which have at least four black direct neighbors. We call concave those white pixels which are surrounded by black pixels from at least four of the eight possible directions. The network transient must be stopped after a given amount of time, depending on the size of the largest holes to be filled in.

A =	0.5	0.5	0.5	**B** =	0	0	0	z =	3.5
	0.5	2	0.5		0	2	0		
	0.5	0.5	0.5		0	0	0		

Erosion: eroding picture with a given structuring element

Erosion represents the probing of an image to see where some primitive shapes fit inside the image. The primitive is called structuring element placed in the **B** term.

A =	0	0	0	**B** =	1	1	1	z =	-8.5
	0	2	0		1	1	1		
	0	0	0		1	1	1		

ALPHA source

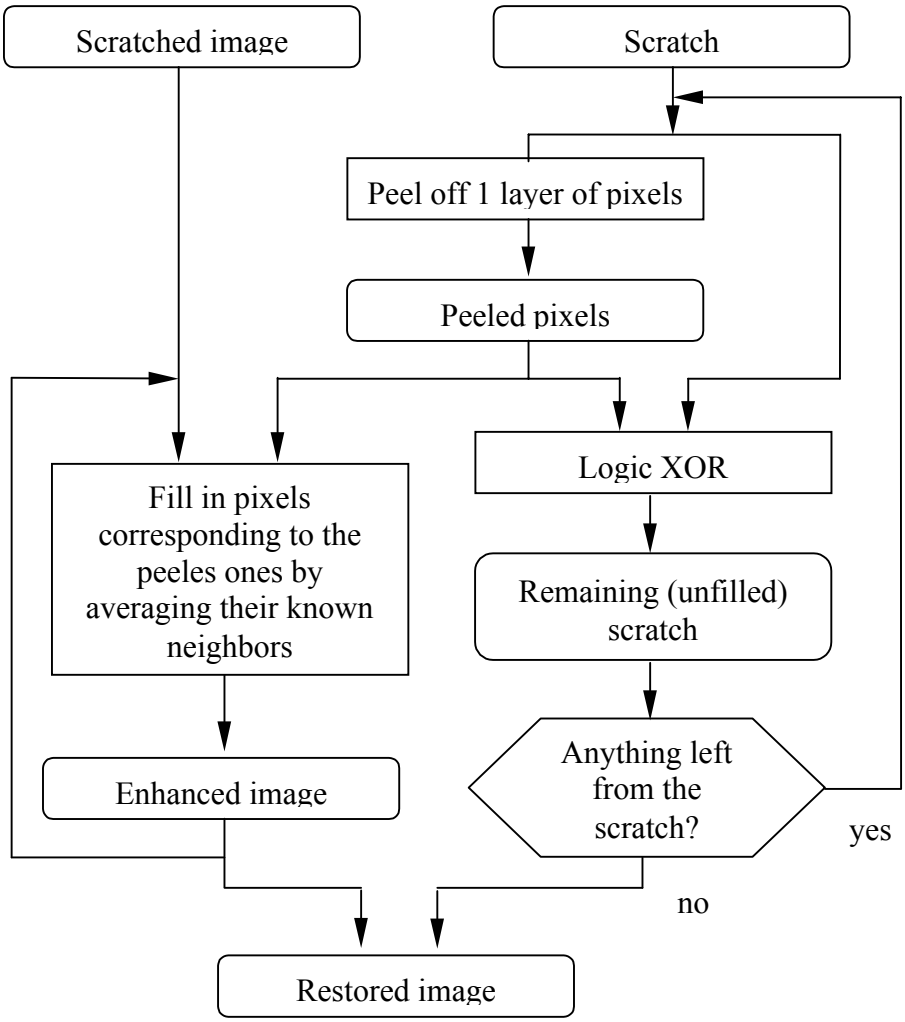
```
/* THE PROGRAM DETECTS CONCAVITIES OF OBJECTS */
PROGRAM concave (in; out);
CONSTANT
ONE = 1;
TWO = 2;
WHITE = -1.0;
TIME1 = 50;
TIME2 = 10;
ENDCONST;
/* Chip set definition section */
CHIP_SET simulator.eng;
A_CHIP
SCALARS
IMAGES
 c1: BINARY;
 c2: BINARY;
 c3: BINARY;
 c4: BINARY;
ENDCHIP;
E_BOARD
SCALARS
IMAGES
 bi1: BINARY;
 bi2: BINARY;
ENDBOARD;
OPERATIONS FROM concave.tms;

PROCESS concave;
USE (thres, hollow, erosion);
  HostLoadPic(in, bi1);
  HostDisplay(bi1, ONE);
  c1 := bi1;
  thres (c1, c1, c1, TIME1, WHITE);
  hollow (c1, c1, c2, TIME1, WHITE);
```

```
  c3 := c1 XOR c2;
  erosion(c3, c3, c1, TIME2, WHITE);
  bi2 := c3;
  HostDisplay(bi2, TWO);
ENDPROCESS;
ENDPROG;
```

SCRATCH REMOVAL

On photocopier machines, the glass panel often gets scratched, which scratch is then copied together with the material, resulting in a visually annoying copy. The following algorithm is capable of removing such scratches assuming that the location of the scratch is known in advance. This is a valid assumption, since the scratches can automatically be detected e.g. by copying a blank sheet of paper. The algorithm removes the scratches gradually, peeling off pixels circularly [19].

The flow-chart of the algorithm:



Smoothing:

Selection templates: *Fill templates:*

$\mathbf{A}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{B}_1 = \begin{bmatrix} -0.5 & 0 & 0 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 \end{bmatrix}$ $z_1 = \boxed{-1.5}$ $\mathbf{B}_1 = \begin{bmatrix} 0.33 & 0 & 0 \\ 0.34 & 0 & 0 \\ 0.33 & 0 & 0 \end{bmatrix}$

$\mathbf{A}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{B}_2 = \begin{bmatrix} -0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $z_2 = \boxed{-1.5}$ $\mathbf{B}_2 = \begin{bmatrix} 0.34 & 0.33 & 0 \\ 0.33 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

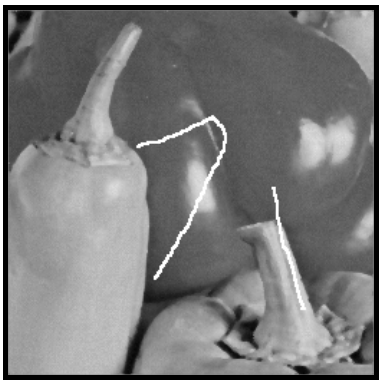
$\mathbf{A}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{B}_3 = \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $z_3 = \boxed{-1.5}$ $\mathbf{B}_3 = \begin{bmatrix} 0.33 & 0.34 & 0.33 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

. . . . . .

$\mathbf{A}_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{B}_8 = \begin{bmatrix} 0 & 0 & 0 \\ -0.5 & 0.5 & 0 \\ -0.5 & -0.5 & 0 \end{bmatrix}$ $z_8 = \boxed{-1.5}$ $\mathbf{B}_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0.33 & 0 & 0 \\ 0.34 & 0.33 & 0 \end{bmatrix}$

Example: image names: peppers.bmp, scratch.bmp; image size: 256x256.

Original image



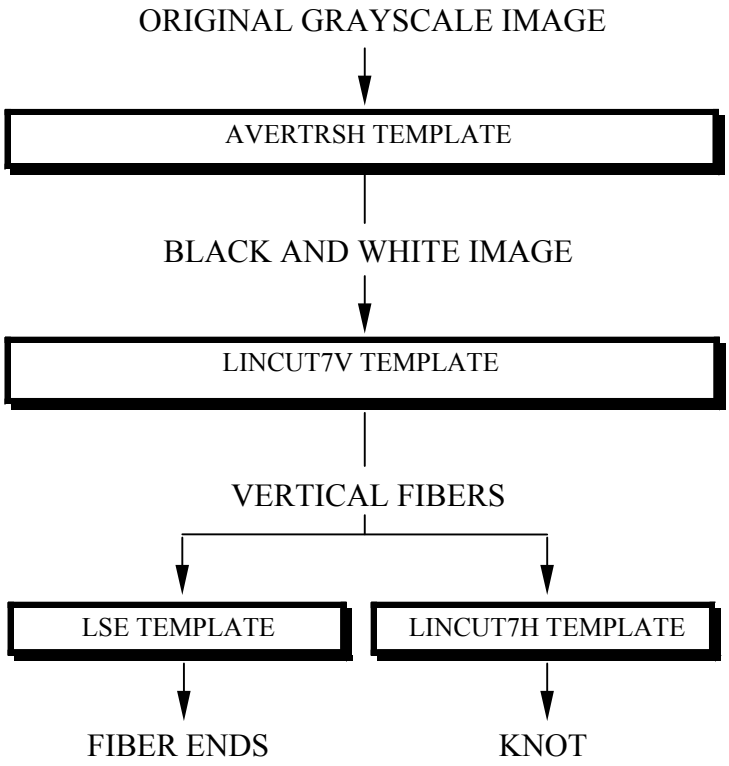
Scratched image

Restored image

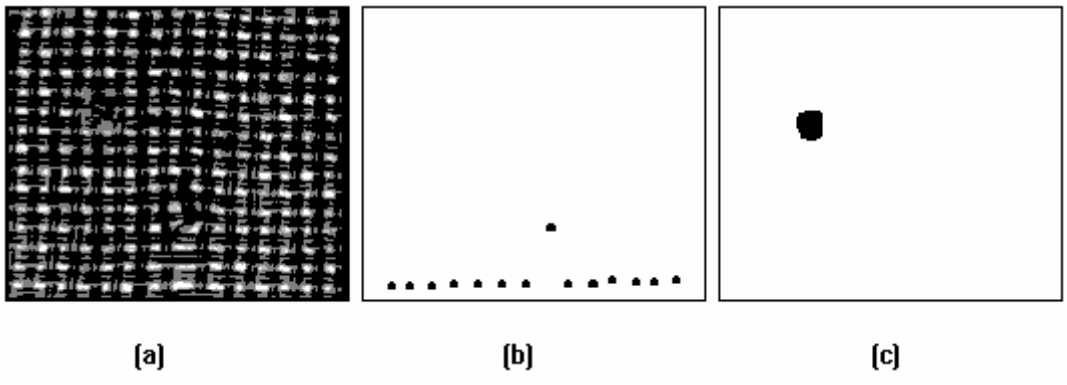
TEXTILE PATTERN ERROR DETECTION

This algorithm finds the knots and fiber breakings in a loose-waved textile. The algorithm is detailed in [10]. The cited templates can be found in this template library. (The lincut7h is the rotated version of lincut7v (*LE7pixelVerticalLineRemover*))

The flow-chart of the algorithm:



Example: The algorithm is demonstrated on a piece of table cloth. (a): the original image, (b) the southern ends of the fibers (the inside one indicates the fault), (c): the knot. Image name: textpatt.bmp; image size: 170x145.

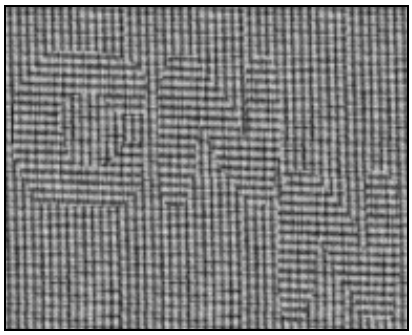


TEXTURE SEGMENTATION I [17]

Short Description

This simple CNN analogic program is able to separate two predefined types of binary textures mixed up in images.
The first step is a template execution, which separates the textures by local ratio of the black-and-white pixels (ROB). In the next step, a small killer template increases the separation. Then an erosion and small killer templates are used for improving the segmentation.

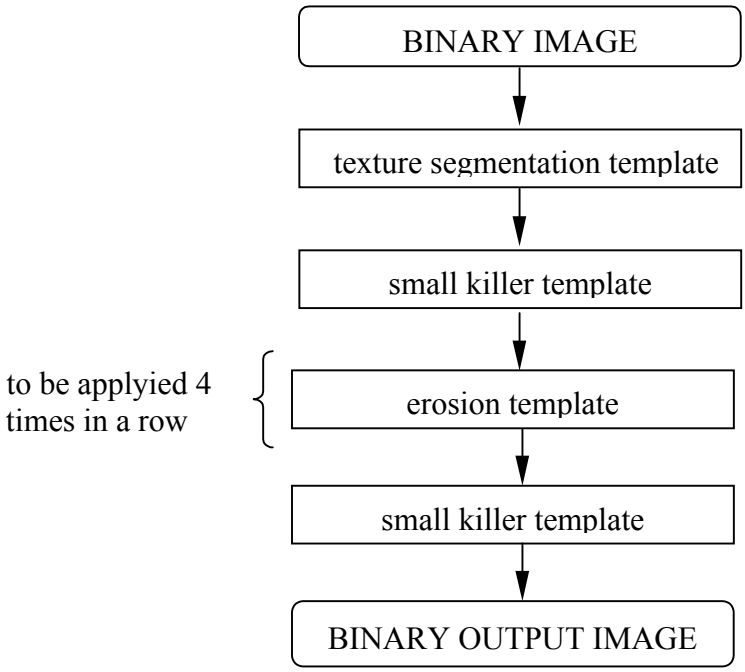
Typical Example



INPUT

OUTPUT

Flow-diagram of the algorithm



Templates used in the algorithm

The small killer template can be found in this template library as *SmallObjectRemover*.
The rest of templates used in this algorithm are described as follows:

Texture segmentation template (tx_hclc1.tem):

A =

-3.44	0.86	-1.64	-0.16	-1.02
-1.09	0.16	-2.19	-3.2	3.51
2.50	1.56	3.91	2.66	2.42
0.55	2.89	-0.62	0.47	3.67
-1.80	-0.55	2.50	-0.23	2.34

B =

-2.19	-0.23	0.16	-0.63	-0.78
1.64	2.27	-3.2	1.09	2.03
0.08	0.55	0.86	3.52	0.08
0.39	-3.83	-3.12	-2.34	-2.11
0.78	-2.66	-1.17	-1.41	1.02

z = 4.8

Erosion template (erosion1.tem):

A =

0	0	0
0	1	0
0	0	0

B =

0.5	1	0.5
1	1	1
0.5	1	0.5

z = -6

ALPHA source

```
/* TEXTURE_1.ALF */
/* Separates two predefined  types of binary textures */
PROGRAM texture_1(in; out);
CONSTANT
ONE = 1;
TWO = 2;
THREE = 3;
FOUR = 4;
FIVE = 5;
SIX = 6;
BLACK = 1;
TimeStep005 = 0.05;
TimeStep01 = 0.1;
TimeStep03 = 0.3;
TimeStep1 = 1;
TimeStep2 = 2;
TIME1 = 1;
TIME2 = 2;
TIME4 = 4;
TIME03 = 0.3;
ENDCONST;
/* Chip set definition section */
CHIP_SET simulator.eng;
A_CHIP
SCALARS
IMAGES
 im1: BINARY;
 im2: BINARY;
 im3: BINARY;
 im4: BINARY;
ENDCHIP;

E_BOARD
```

```
SCALARS
Loop: INTEGER;
IMAGES
 input:  BINARY;
 output: BINARY;
ENDBOARD;
/* Definition of analog operation symbol table */
OPERATIONS FROM texture_1.tms;
PROCESS texture_1;
 USE (tx_hclc1, smkiller, erosion1);
 HostLoadPic(in, input);
 HostDisplay(input, ONE);
 im1:= input;
    SwSetTimeStep (TimeStep005);
     tx_hclc1(im1, im1, im2, TIME1, ZEROFLUX);
     output:=im2;
     HostDisplay(output, TWO);
    SwSetTimeStep (TimeStep01);
    smkiller (im2, im2, im3, TIME03, BLACK);
    output:=im3;
    HostDisplay(output, THREE);
    SwSetTimeStep (TimeStep01);
    REPEAT Loop:= 1 to 4 BY 1;
      erosion1(im3, im3, im3, TIME2,  BLACK);
    ENDREPEAT;
    output:=im3;
    HostDisplay(output, FOUR);
   SwSetTimeStep (TimeStep01);
    smkiller(im3, im3, im4, TIME4, ZEROFLUX);
    output:=im4;
    HostDisplay(output, FIVE);
ENDPROCESS;
ENDPROG;
```

Comments

Using the genetic template learning algorithm, a wide range of texture types can be segmented with high accuracy. In order to achieve flat segments propagating-type filter template(s) could be used.

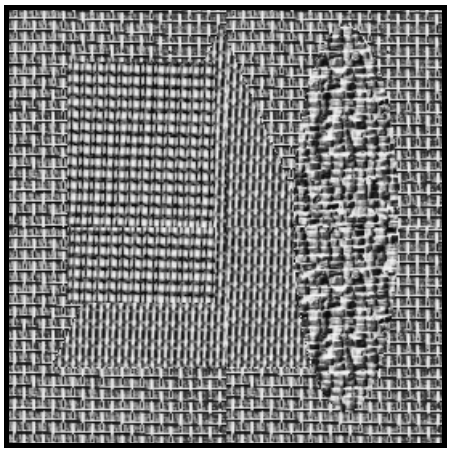
TEXTURE SEGMENTATION II [17]

Short Description

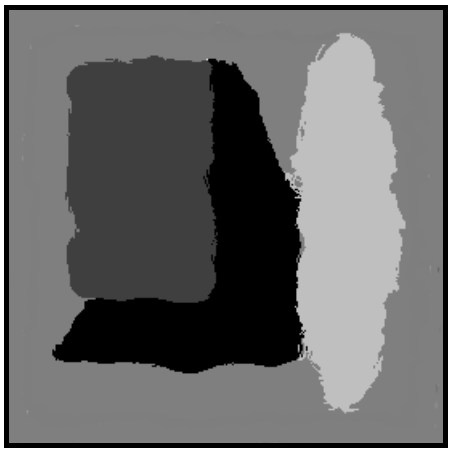
This simple CNN analogic program is able to separate 4 (maybe more) types of textures mixed up in images.
We run 4 or 5 templates consecutively. The first step in an iteration is a template execution which separates the textures by local ratio of the black-and-white pixels (ROB). The chip may physically scan the textured surface or the texture-segments of the whole image are executed separately by reading in from the A-RAM.

Typical Example

The following example can be evaluated by the CNN Engine Board using the next-generation CNN chip.

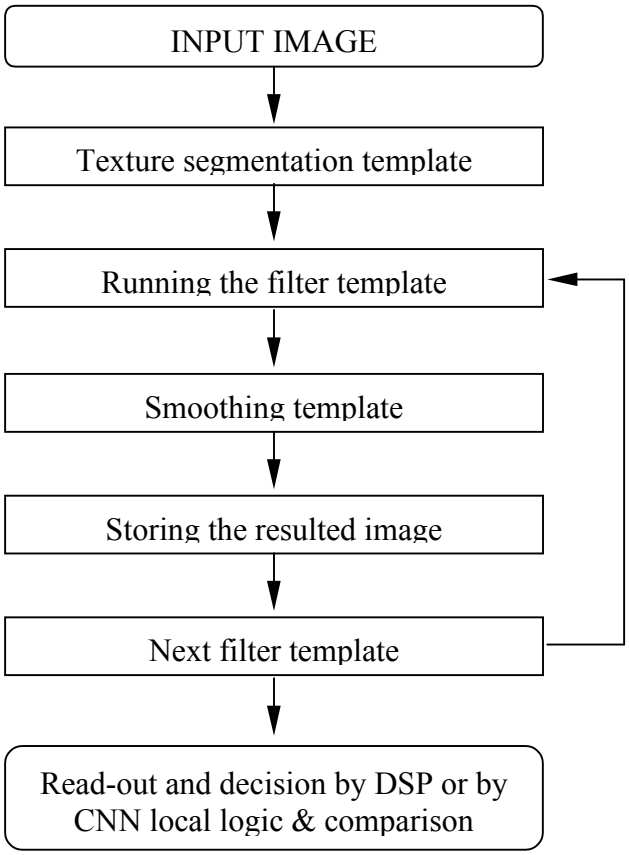


INPUT



OUTPUT

Flow-diagram of the algorithm



Templates used in the algorithm

Texture segmentation templates can be found in this library as *3x3TextureSegmentation*, *5x5TextureSegmentation#*, *Smoothing*.

The Alpha language description of the core of the texture segmentation algorithm is as follows:

ALPHA source

```
/* TEXTURE_2.ALF */
/* Classifies 4 different textured images */
PROGRAM texture_2(in);
CONSTANT
  ONE = 1;
  TWO = 2;
  THREE = 3;
  FOUR = 4;
  WHITE = -1.0;
  RUNTIME_3 = 3;
  TIMESTEP = 0.2;
ENDCONST;
/* Chip description file */
CHIP_SET simulator.eng;
/* Chip variables */
A_CHIP
SCALARS
IMAGES
  ci1: ANALOG;
```

```
  ci2: BINARY;
ENDCHIP;
/* Board variables */
E_BOARD
SCALARS
  GLOB_COUNT: REAL;
IMAGES
  input: BYTE;
  display: BINARY;
ENDBOARD;
/* Template list */
OPERATIONS FROM texture_2.tms;
PROCESS texture_2;
USE (tx_hclc1);
  HostLoadPic(in, input);
  SwSetTimeStep (TIMESTEP);
  ci1:=input;
  tx_hclc1 (ci1, ci1, ci2, RUNTIME_3, ZEROFLUX);
  display:=ci2;
  HostDisplay(display, ONE);
ENDPROCESS;
ENDPROG;
```

VERTICAL WING ENDINGS DETECTION OF AIRPLANE-LIKE OBJECTS [51]

Short Description

One of the characteristic features of objects, which the human recognition is based on, is the local curvature. This algorithm detects property, namely, the locations of a binary image where the local edges are convex from north. By this method, for example, the wing endings of an airplane can be detected.

In the first part of the algorithm local shadows are created with appropriate templates in the image into the 35°, 65°, 125°, 155°, -155°, -115°, -65° and -25° directions. The generation of shadows depends on the local curvature of edges. As a result we get eight images. Then four by four, groups of images get selected from the eight images and the logic **AND** operation of four images (within each group) is performed. With this step we can enhance the direction selectivity of the fill operation. In the next step we take the logic difference of each of these images and the original image. Then the undesired arc locations (the orientations of these locations are orthogonal to the preferred one) are subtracted from the resulting images. This way we get two images containing patches which denote the possible wing endings. In the last phase shadows are created, starting from these patches into appropriate directions according to the direction represented by the patch. The logic **AND** of the two shadow images and the arc location images one by one yields two images whose union gives the final result.

Some incorrectly detected points can be seen on the resulting image. More sophisticated subtracting and shadowing methods are able to remove these points. The algorithm is invariant to small rate rotation and distortion.

Templates used in the algorithm

FILL35:

A = | 1 | 0 | 1 |
|---|---|---|
| 0 | 2 | 0 |
| 1 | 1 | 0 |

B = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

z = 2

FILL65:

A = | 1 | 0 | 0 |
|---|---|---|
| 1 | 2 | 0 |
| 0 | 0 | 2 |

B = | 0 | 2 | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

z = 3

FILL125:

A = | 1 | 0 | 0 |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 0 | 1 |

B = | 0 | 0 | 0 |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

z = 2

FILL155:

A = | 0 | 0 | 2 |
|---|---|---|
| 0 | 2 | 0 |
| 1 | 1 | 0 |

B = | 0 | 0 | 0 |
|---|---|---|
| 2 | 0 | 0 |
| 0 | 0 | 0 |

z = 3

FILL-155:

A = | 0 | 1 | 1 |
| 0 | 2 | 0 |
| 1 | 0 | 1 |

B = | 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

z = 2

FILL-115:

A = | 2 | 0 | 0 |
| 0 | 2 | 1 |
| 0 | 0 | 1 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 2 | 0 |

z = 3

FILL-65:

A = | 1 | 0 | 1 |
| 1 | 2 | 0 |
| 0 | 0 | 1 |

B = | 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

z = 2

FILL-25:

A = | 0 | 1 | 1 |
| 0 | 2 | 0 |
| 2 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 2 | 0 |

z = 3

SHADOW90:

A = | 0 | -1 | 0 |
| 0.3 | 2 | 0.3 |
| 0.4 | 1 | 0.4 |

B = | 0 | 0 | 0 |
| 0 | 1.4 | 0 |
| 0 | 0 | 0 |

z = 2.5

SHADOW270:

A = | 0.4 | 1 | 0.4 |
| 0.3 | 2 | 0.3 |
| 0 | -1 | 0 |

B = | 0 | 0 | 0 |
| 0 | 1.4 | 0 |
| 0 | 0 | 0 |

z = 2.5

LOGANDN:

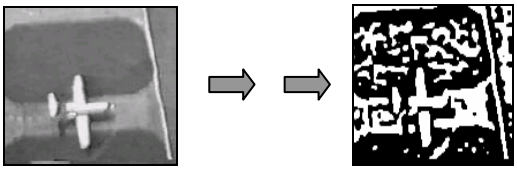
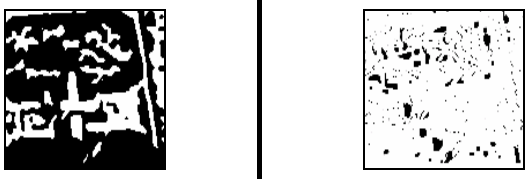
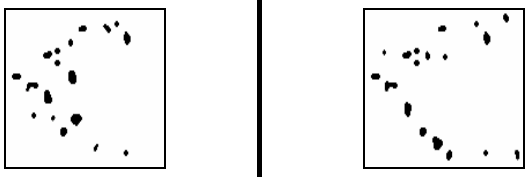
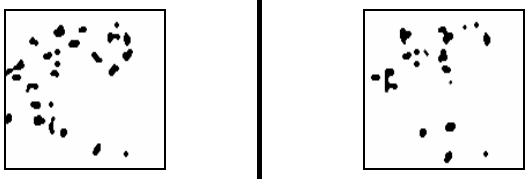
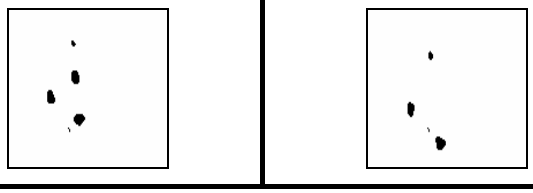
A = | 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | -1 | 0 |
| 0 | 0 | 0 |

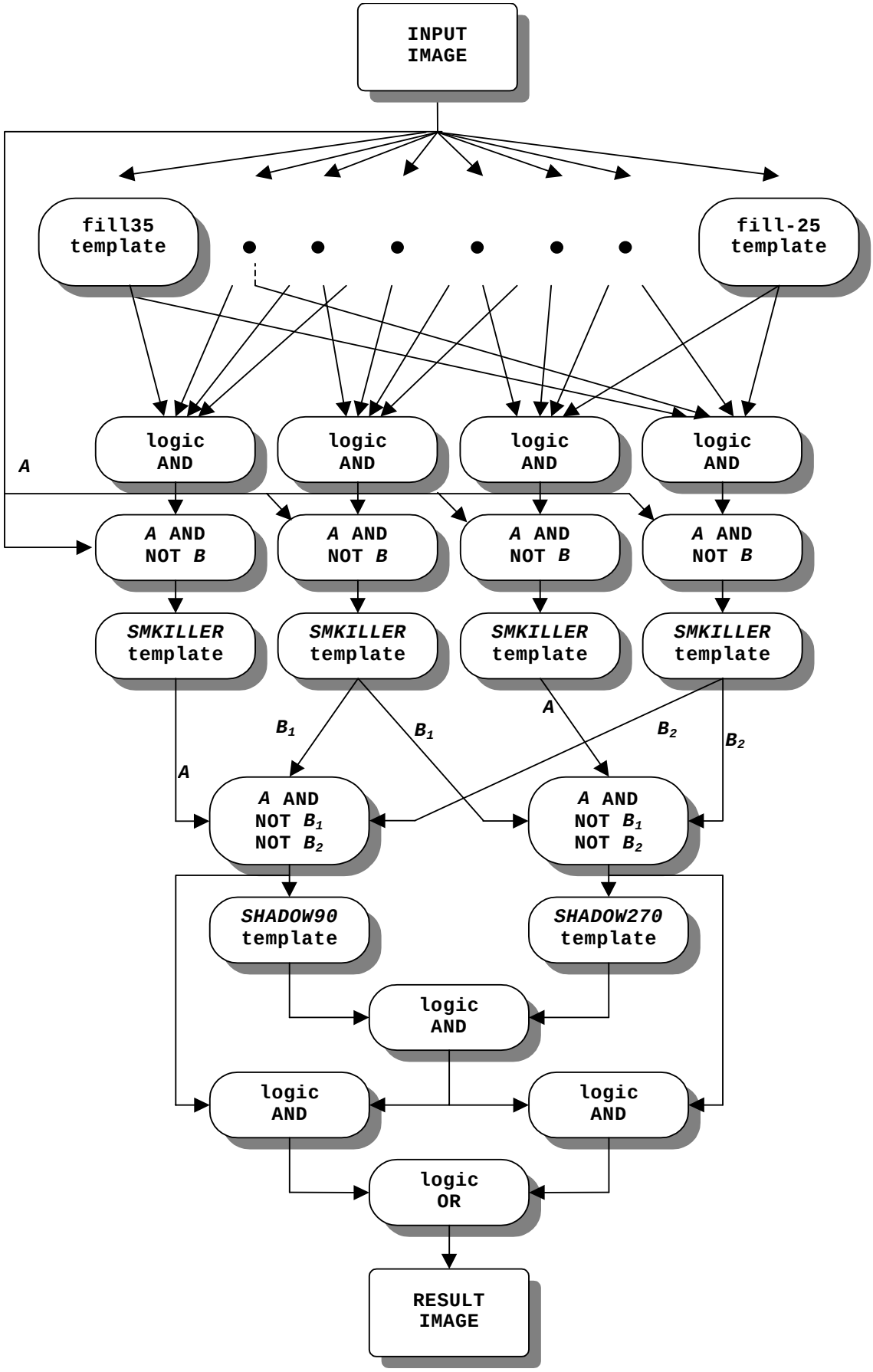
z = -1

The other templates used in the algorithm are available in this library.

Typical Example

Airplane wing endings detection	
Input image	
Concavities into all eight directions (only the first 4 are displayed)	
*Make logic **AND** of 4 subsequent images (only the first 4 are displayed). Logic difference to the original image*	
*The result of **SMKILLER** template containing the desired concave locations*	
*The result of **SMKILLER** template containing the locations which are orthogonal to the desired concave locations*	
*After subtracting (logic **AND**) the orthogonal directions*	
The result (masked with the original)	

Flow-diagram of the algorithm



ALPHA source

```
/* airplane.ALF                                    */
/* Performs airplane wing ending detection */

PROGRAM airplane(in; out);
CONSTANT
ONE = 1;
TWO = 2;
THREE = 3;
FOUR = 4;
FIVE = 5;
SIX = 6;
SEVEN = 7;
EIGHT = 8;
NINE = 9;
SIXTY = 60;
WHITE = -1.0;
TIME = 25;
TEN =10;
TS =0.9;
ENDCONST;

/* Chip definiton section                          */
CHIP_SET simulator.eng;
A_CHIP
SCALARS

IMAGES
input: BINARY;          /* input */
arc35: BINARY;          /* arc */
arc65: BINARY;          /* arc */
arc125: BINARY;         /* arc */
arc155: BINARY;         /* arc */
arc_35: BINARY;         /* arc */
arc_65: BINARY;         /* arc */
arc_125: BINARY;        /* arc */
arc_155: BINARY;        /* arc */
wing_up: BINARY;
wing_left: BINARY;
wing_down: BINARY;
wing_right: BINARY;
shadow_up: BINARY;
shadow_down: BINARY;
shadow_intrsct: BINARY;
output: BINARY;
ENDCHIP;

/* Chip set definition section                     */
E_BOARD
SCALARS
IMAGES
```

```
ENDBOARD;

/* Definition of analog operation symbol table                    */
OPERATIONS FROM airplane.tms;

PROCESS airplane;
USE (fill35, fill65, fill125, fill155, fill_115, fill_65, fill_25, fill_155, logdif, smkiller, shadow0,
shadow90, shadow180, shadow270);
SwSetTimeStep(TS);
HostLoadPic(in, input);
HostDisplay(input, ONE);

/* Filling into different directions */
fill35 ( input , input , arc35 , TIME , WHITE );
fill65 (input,input,arc65, TIME, WHITE);
fill125 (input,input,arc125, TIME, WHITE);
fill155 (input,input,arc155, TIME, WHITE);
fill_25 (input,input,arc_35, TIME, WHITE);
fill_65 (input,input,arc_65, TIME, WHITE);
fill_115 (input,input,arc_125, TIME, WHITE);
fill_155 (input,input,arc_155, TIME, WHITE);

/* Enhance direction selectivity to degree 90*/
wing_up:=     arc35 AND arc65;
wing_up:= wing_up AND arc125;
wing_up:= wing_up AND arc155;
logdif (input,wing_up,wing_up, TWO, WHITE);
smkiller (wing_up,wing_up,wing_up, TEN, WHITE);
HostDisplay(wing_up, THREE);

/* Enhance direction selectivity to degree 180 */
wing_left:=    arc125 AND arc155;
wing_left:= wing_left AND arc_125;
wing_left:= wing_left AND arc_155;
logdif (input,wing_left,wing_left, TWO, WHITE);
smkiller (wing_left,wing_left,wing_left, TEN, WHITE);
HostDisplay(wing_left, FOUR);

/* Enhance direction selectivity to degree 270*/
wing_down:= arc_35 AND arc_65 ;
wing_down:= wing_down AND arc_125;
wing_down:= wing_down AND arc_155;
logdif (input,wing_down,wing_down, TWO, WHITE);
smkiller (wing_down,wing_down,wing_down,TEN, WHITE);
HostDisplay(wing_down, FIVE);

/* Enhance direction selectivity to degree 0*/
wing_right:=  arc_35 AND arc_65;
wing_right:= wing_right AND arc35;
wing_right:= wing_right AND arc65;
logdif (input,wing_right,wing_right, TWO, WHITE);
```

```
smkiller (wing_right,wing_right,wing_right, TEN, WHITE);
HostDisplay(wing_right, TWO);

/* Removing of horizontally directed arcs*/
wing_up:= wing_up AND1NOT2 wing_right;
wing_up:= wing_up AND1NOT2 wing_left;
smkiller (wing_up,wing_up,wing_up, TEN, WHITE);
/* Shadow generation to degree 90*/
shadow90(wing_up, wing_up, shadow_up, SIXTY, WHITE );
HostDisplay(shadow_up, SIX);

/* Removing of horizontally directed arcs*/
wing_down:= wing_down AND1NOT2 wing_right;
wing_down:= wing_down AND1NOT2 wing_left;
smkiller (wing_down,wing_down,wing_down, TEN, WHITE);

/* Shadow generation to degree 270*/
shadow270(wing_down,wing_down,shadow_down, SIXTY, WHITE);
HostDisplay(shadow_down, SEVEN);

/* Distance classification*/
shadow_intrsct:=shadow_up AND shadow_down;
HostDisplay(shadow_intrsct, EIGHT);

/* Resulting wing endings (up and down)*/
output:=wing_up OR wing_down;
output:=output AND shadow_intrsct;
HostDisplay(output, NINE);

ENDPROCESS;
ENDPROG;
```

Chapter 4. PDE SOLVERS

LaplacePDESolver: Solves the Laplace PDE: $\nabla^2 x = 0$

Old names: LAPLACE

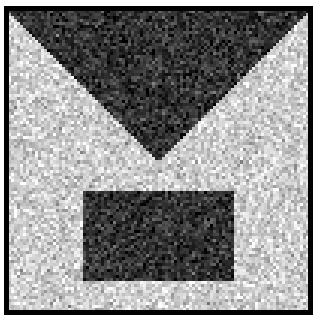
$$\mathbf{A} = \begin{bmatrix} 0 & \nu/h^2 & 0 \\ \nu/h^2 & -4*\nu/h^2+1/R & \nu/h^2 \\ 0 & \nu/h^2 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

where ν is the diffusion constant,
h is the spatial step
R is the value of the resistor of the CNN cell.

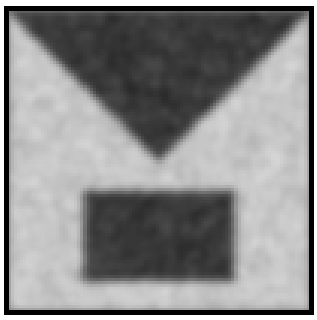
If $\nu = h = R = 1$, our template is as follows:

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

Example : image name: laplace.bmp, image size: 100x100; template name: laplace.tem .



initial state



output (t=T)

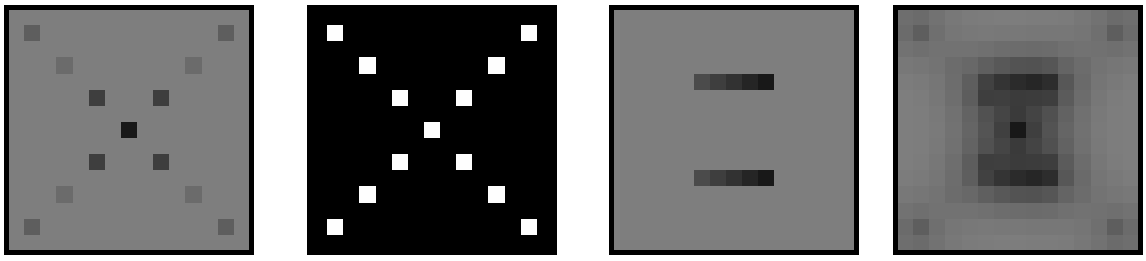
PoissonPDESolver: Solves the Poisson PDE: $\nabla^2 x = f(x)$

Old names: POISSON

$$\mathbf{A} = \begin{bmatrix} 0 & \nu/h^2 & 0 \\ \nu/h^2 & -4*\nu/h^2+1/R & \nu/h^2 \\ 0 & \nu/h^2 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

where -f(x) is filled into the Bias map.

Example: $\nu = h = R = 1$. Image names: poisson1.bmp, poisson2.bmp, poisson3.bmp; image size: 15x15; template name: poisson.tem . Torus frame style is used.



initial state fixed state bias map output

Chapter 5. NEUROMORPHIC ILLUSIONS AND SPIKE GENERATORS

***HerringGridIllusion: Herring-grid illusion* [13]**

Old names: HERRING

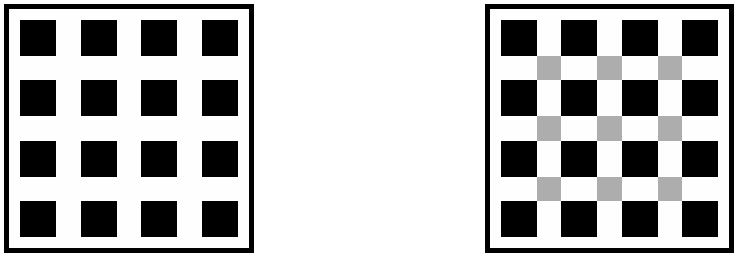
$$\mathbf{B} = \begin{bmatrix} -0.16 & -0.16 & -0.16 & -0.16 & -0.16 \\ -0.16 & -0.40 & -0.40 & -0.40 & -0.16 \\ -0.16 & -0.40 & 4 & -0.40 & -0.16 \\ -0.16 & -0.40 & -0.40 & -0.40 & -0.16 \\ -0.16 & -0.16 & -0.16 & -0.16 & -0.16 \end{bmatrix} \qquad z = \boxed{0}$$

$$\mathbf{A}^{\tau} = \begin{bmatrix} -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \\ -0.1 & -0.3 & -0.3 & -0.3 & -0.1 \\ -0.1 & -0.3 & 0 & -0.3 & -0.1 \\ -0.1 & -0.3 & -0.3 & -0.3 & -0.1 \\ -0.1 & -0.1 & -0.1 & -0.1 & -0.1 \end{bmatrix} \qquad \tau = 3\ \tau_{CNN}$$

I. Global Task

Given:	static binary image **P** with a grid of black squares
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(∞)** = Grayscale image representing **P** with gray patches at the intersections of the grid of black squares.

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



input output

MüllerLyerIllusion: ***Simulates the Müller-Lyer illusion*** **[13]**

Old names: MULLER

A =

0	0	0	0	0
0	0	0	0	0
0	0	1.3	0	0
0	0	0	0	0
0	0	0	0	0

B =

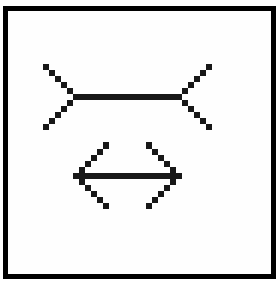
-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	1.3	-0.1	-0.1
-0.1	-0.1	-0.1	-0.1	-0.1
-0.1	-0.1	-0.1	-0.1	-0.1

z = -2.8

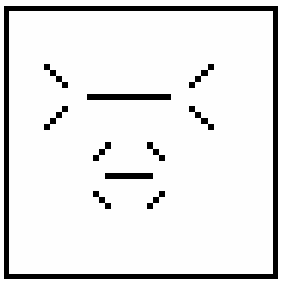
I. Global Task

Given:	static binary image **P** representing two horizontal lines between arrows. The arrows are dark-gray, the background is white
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 showing that the horizontal line on the top in **P** seems to be longer than the other one.

II. Example: image name: muller.bmp, image size: 44x44; template name: muller.tem .



input

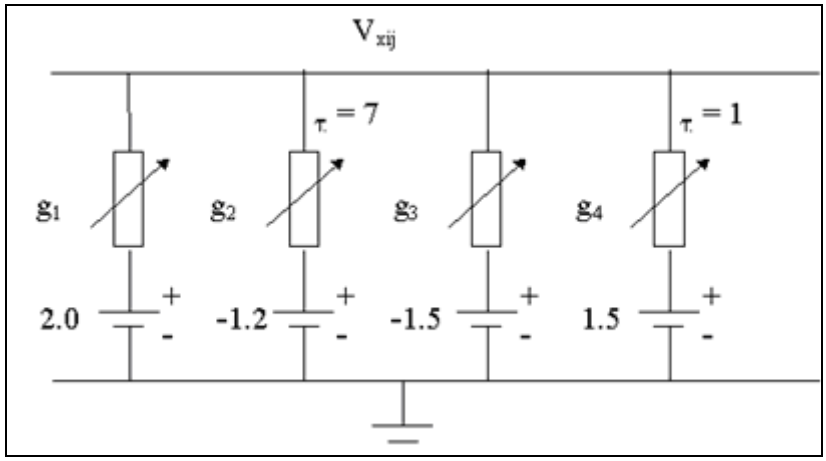


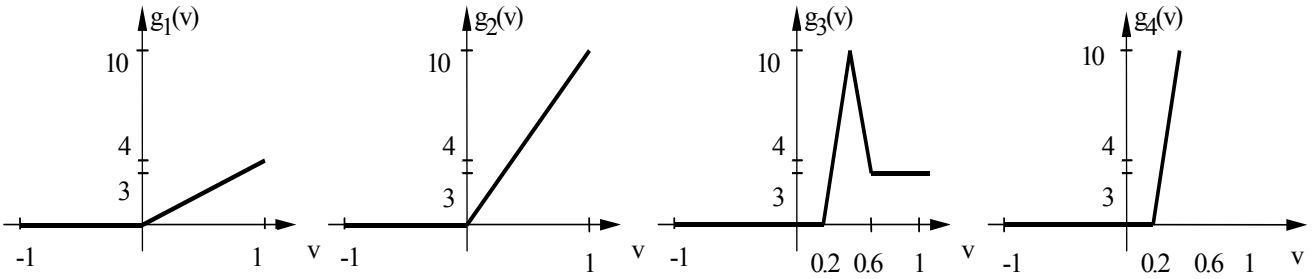
output

SpikeGeneration1: ***Rhythmic burst-like spike generation using 4 ion channels, 2 of them are delayed***

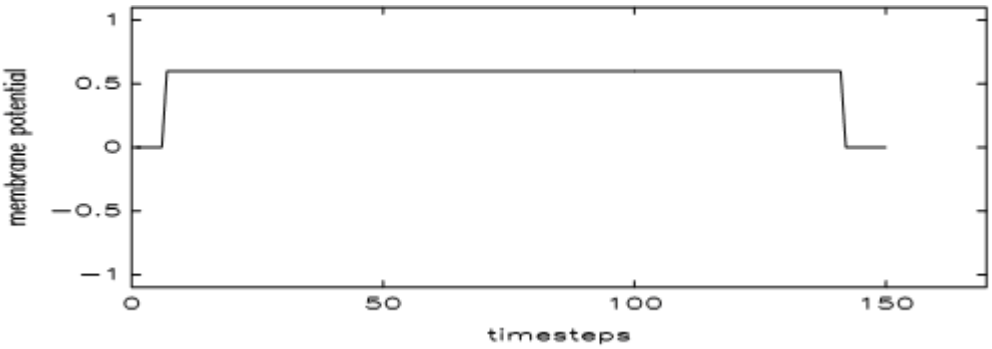
Old names: SPIKE_BU

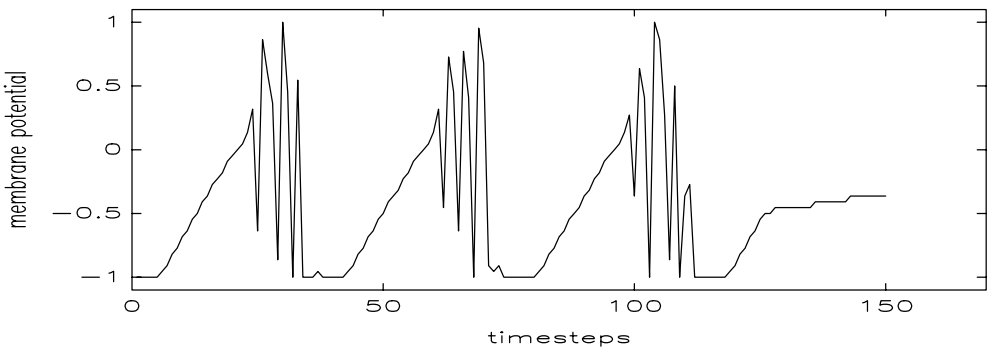
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \begin{bmatrix} 0 \end{bmatrix}$$





Example: Input and output waveforms





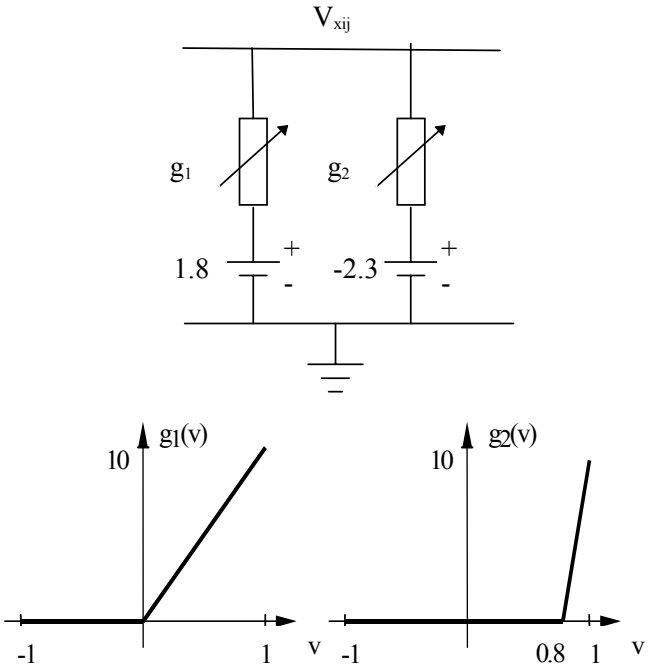
SpikeGeneration2: ***Action potential generation in a neuromorphic way*** **without delay** ***using 2 ion channels***

Old names: SPIKE_N

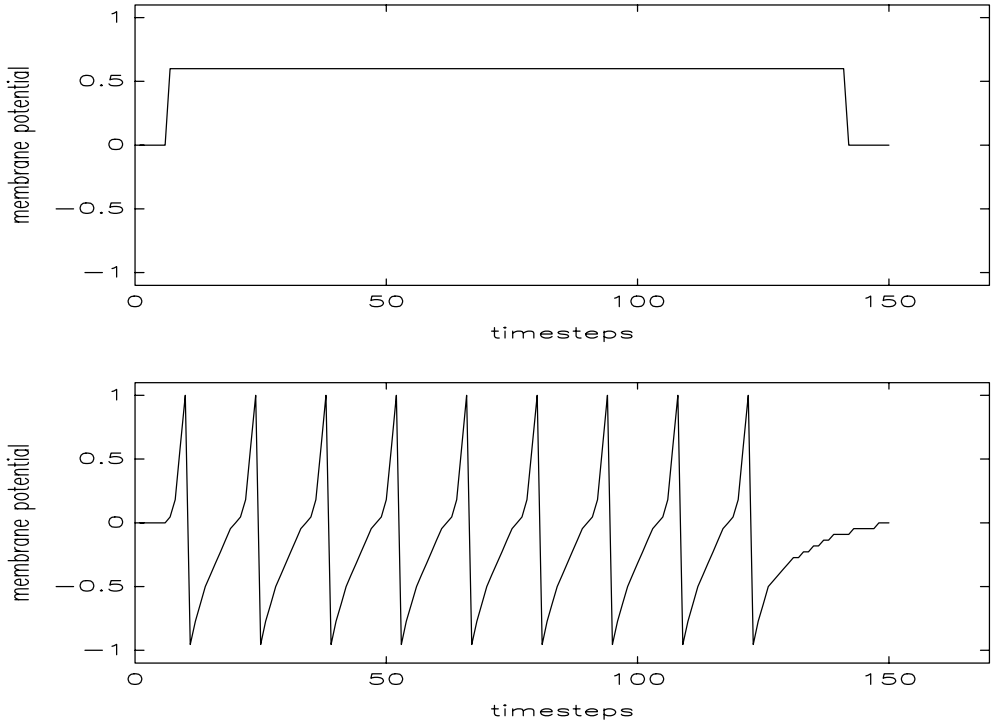
A = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| --- | --- | --- |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

z = | 0 |
| --- |



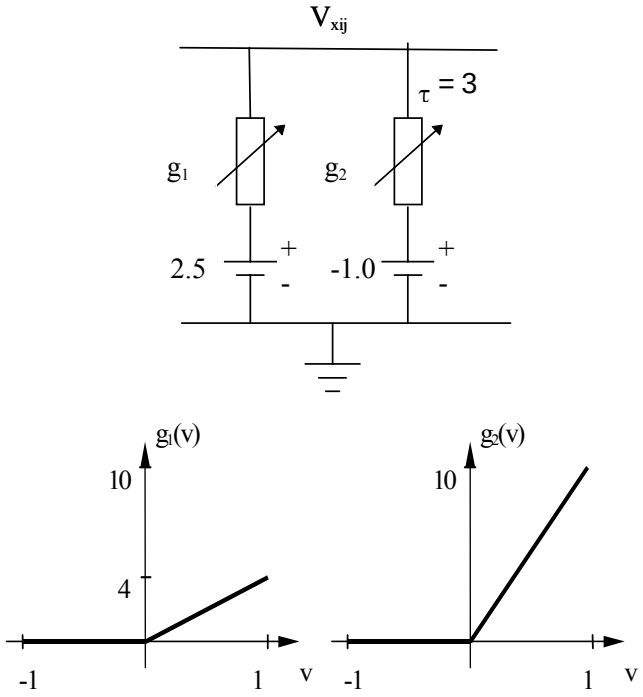
Example: Input and output waveforms



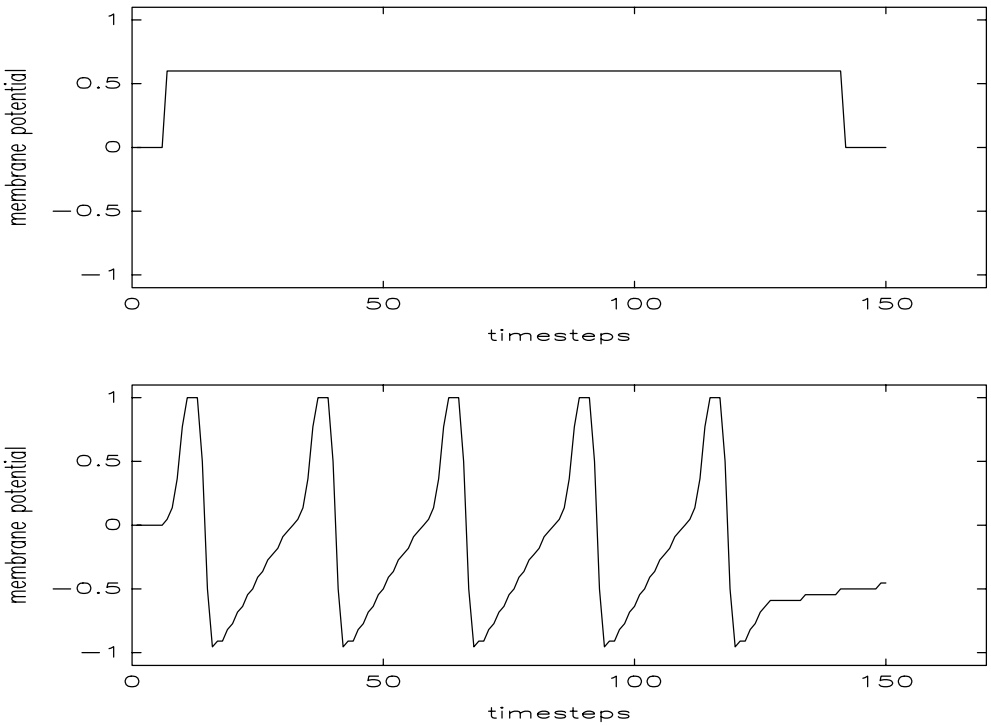
SpikeGeneration3: ***Action potential generation in a* neuromorphic *way, using 2 ion channels where one is* delayed. *Ion channels are modeled with voltage-controlled conductance (VCC) templates***

Old names: SPIKE_ND

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{0}$$



Example: Input and output waveform at each cell

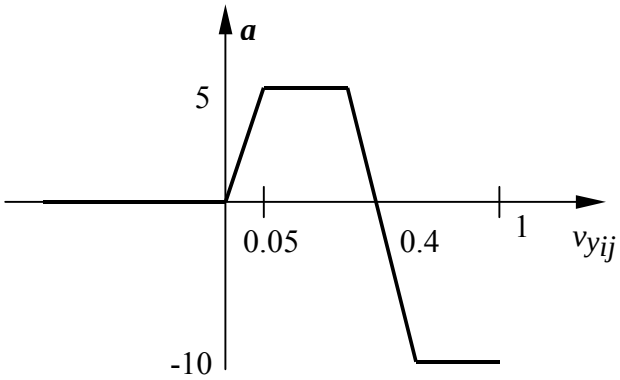


SpikeGeneration4: ***Action potential (spike) generation in a phenomenological way (with a nonlinear multivibrator-like template)***

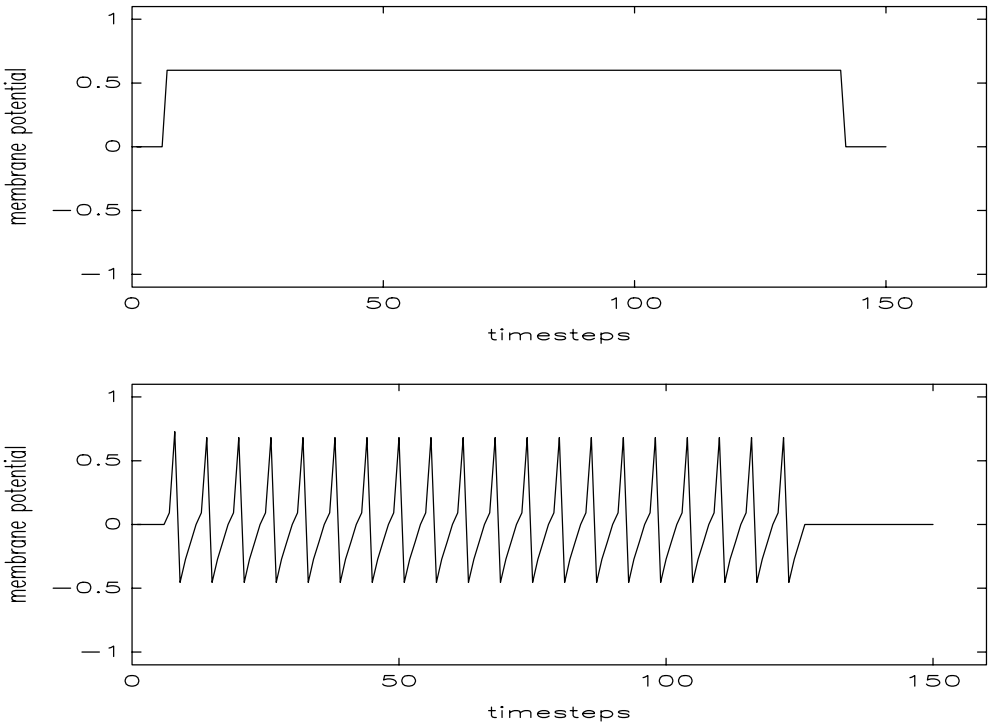
Old names: SPIKE_PH

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \boldsymbol{a} & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = \boxed{0}$$

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



Example: Input and output waveform at each cell



Chapter 6. OTHERS

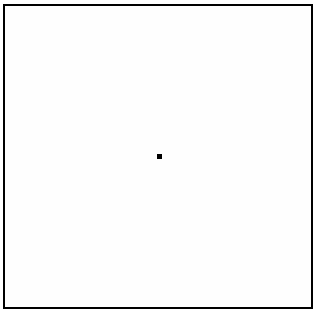
CELLULAR AUTOMATA **[44]**

Short Description

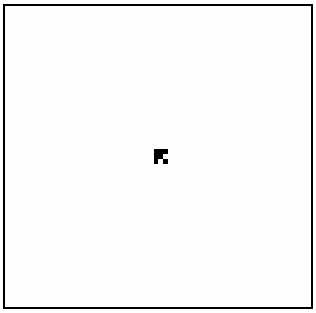
The following simple algorithm simulates the functioning of a cellular automaton. Input and output pictures are binary. The input of the n^{th} iteration is replaced by the output of the $(n-1)^{th}$ iteration.

Typical Example

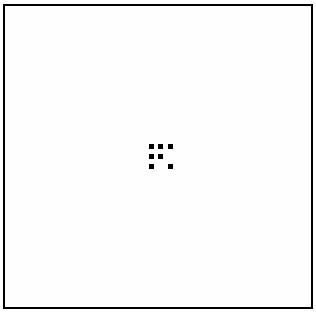
The following example shows a few consecutive states of the simulated cellular automata.



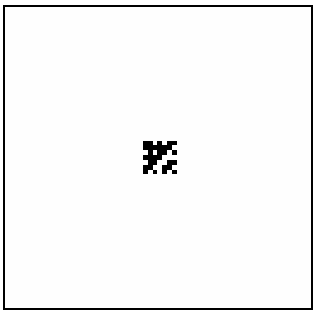
INPUT



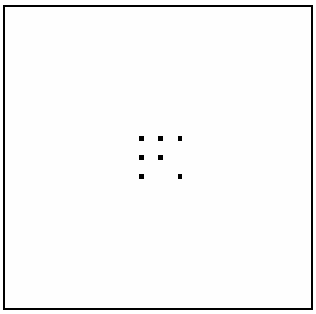
Output of the 1. iteration



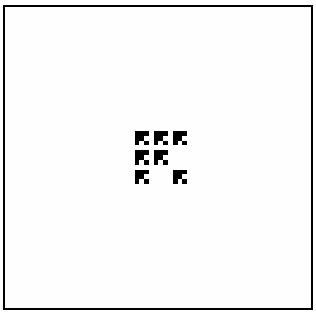
Output of the 2. iteration



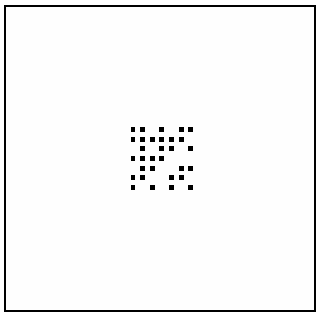
Output of the 3. iteration



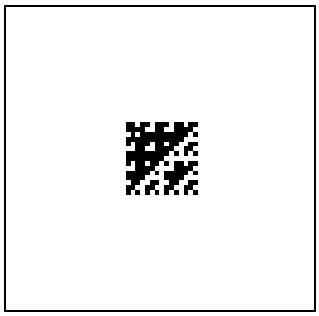
Output of the 4. iteration



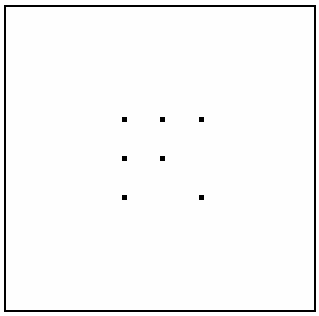
Output of the 5. iteration



Output of the 6. iteration

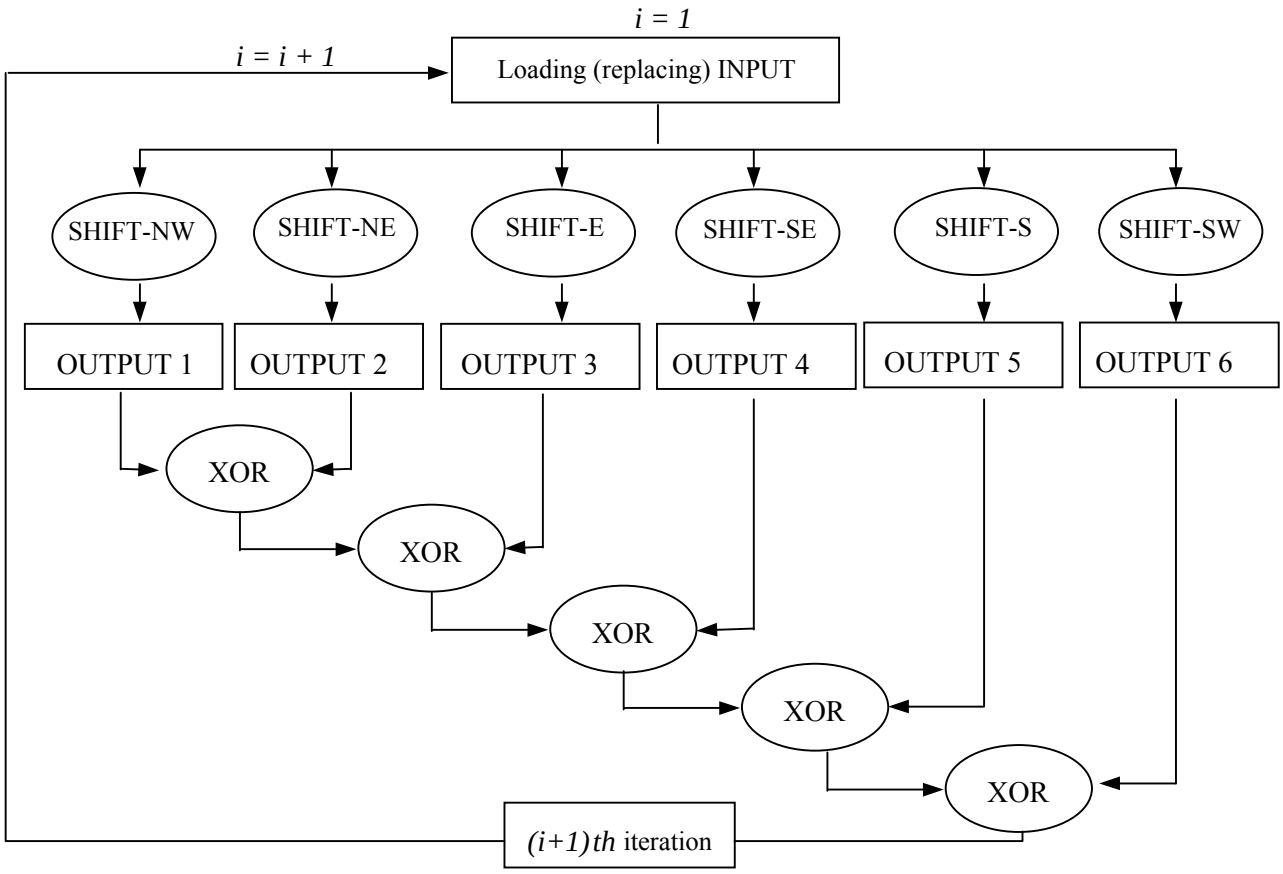


Output of the 7. iteration



Output of the 8. iteration

Block Diagram of the Algorithm



Templates used in the algorithm

Templates used in the algorithm are direction-dependent SHIFT templates, which are as follows:

SHIFT_NW:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

SHIFT_NE:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

SHIFT_E:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad z = \boxed{0}$$

SHIFT_SE:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

z = | 0 |

SHIFT_S:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |

z = | 0 |

SHIFT_SW:

A = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B = | 0 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 0 | 0 |

z = | 0 |

ALPHA source

```
/* CELLAUT.ALF */
/* Performs a function of the cellular automata; */

PROGRAM cellaut(in);
CONSTANT
ONE = 1;
TIME = 5;
TIMESTEP = 1.0;
ENDCONST;

CHIP_SET simulator.eng;
/* Chip definition section */
A_CHIP
SCALARS

IMAGES
 im1: BINARY;
 im2: BINARY;
 im3: BINARY;
ENDCHIP;

/* Chip set definition section */
E_BOARD
SCALARS
cycle: INTEGER;
IMAGES
 input:  BINARY;
ENDBOARD;
```

```
/* Definition of analog operation symbol table */
OPERATIONS FROM cellaut.tms;

PROCESS cellaut;
 USE (shift_nw, shift_ne, shift_e, shift_se, shift_s, shift_sw);

 SwSetTimeStep (TIMESTEP);
 HostLoadPic(in, input);
 im1:= input;
 cycle:=1;

 REPEAT UNTIL (cycle > 30) ;
 input:=im1;
 HostDisplay(input, ONE);
    shift_nw (im1, im1, im2, TIME, PERIODIC);
    im2 := im1 XOR im2;

   shift_ne(im1, im1, im3, TIME, PERIODIC);
    im2 := im3 XOR im2;

   shift_e(im1, im1, im3, TIME, PERIODIC);
    im2 := im3 XOR im2;

   shift_se(im1, im1, im3, TIME, PERIODIC);
    im2 := im3 XOR im2;

   shift_s(im1, im1, im3, TIME, PERIODIC);
    im2 := im3 XOR im2;

   shift_sw(im1, im1, im3, TIME, PERIODIC);
    im2 := im3 XOR im2;

  im1:=im2;
 cycle := cycle + 1;
 ENDREPEAT;
ENDPROCESS;
ENDPROG;
```

GENERALIZED CELLULAR AUTOMATA [44]

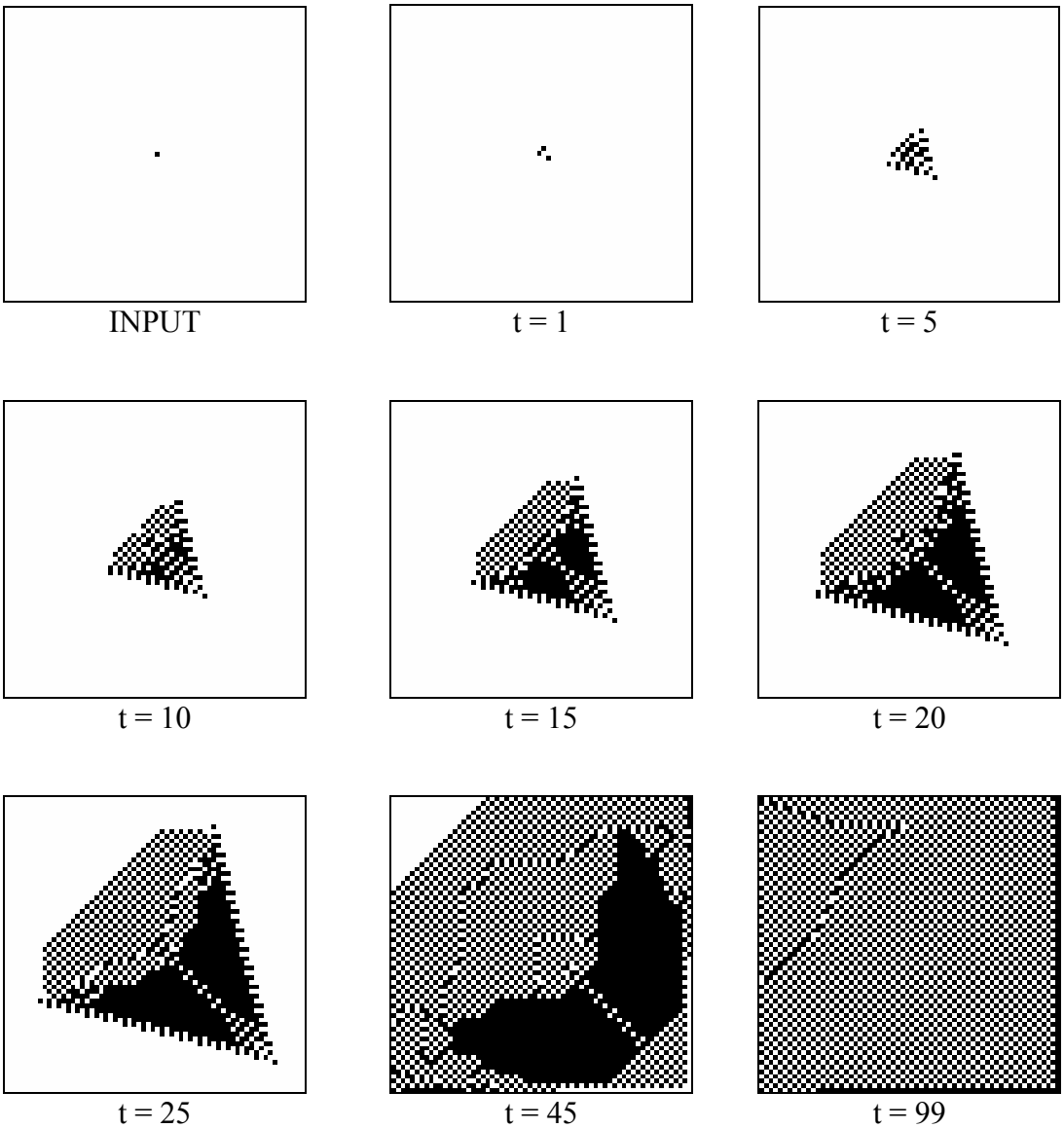
Definition: A Generalized Cellular Automaton (GCA) is a CNN with a single binary input/output template. The output is fed back to the input (the initial state is the same and is prescribed). If B = 0 then the output is fed back to the initial state.

Short Description

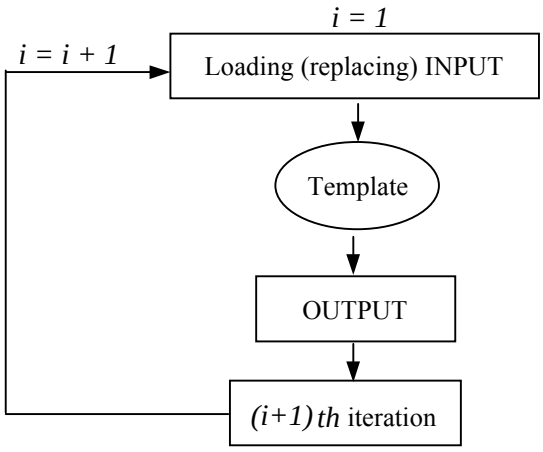
The following simple algorithm simulates the functioning of a general cellular automaton. Input and output pictures are binary. In each consecutive step the initial state is equal to zero. The input of the n^{th} iteration is replaced by the output of the $(n-1)^{th}$ iteration.

Typical Example

The following example shows a few consecutive steps of the simulated general cellular automaton.



Block Diagram of the Algorithm



Templates used in the algorithm

A =

0	0.5	0
0.5	2	-1
0	-1	0

B =

0	0.5	0
0.5	-0.5	0.5
0	0.5	0

z = 0.5

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*, Vol. 11, No. 6, pp.1385-1393, November 2000.

[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 (CNNA'2000)*, pp. 3-8, Catania, Italy, 23-25 May, 2000.

[53] B. E. Shi, "Gabor-type filtering in space and time with cellular neural networks," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 45, pp. 121-132, 1998.

[54] G. Tímár, K. Karacs, and Cs. Rekeczky: Analogic Preprocessing and Segmentation Algorithms for Off-line Handwriting Recognition., *IEEE Journal on Circuits, Systems and Computers*, Vol. 12(6), pp. 783-804, Dec. 2003.

[55] L. Orzó, T. Roska, "A CNN image-compression algorithm for improved utilization of on-chip resources", *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'2004)*, pp. 297-302, Budapest, Hungary, 22-24 July, 2004.

[56] I. Szatmari, "Synchronization Mechanism in Oscillatory Cellular Neural Networks", *Research report of the Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI),* DNS-1-2006, Budapest 2006.

[57] I. Petrás, T. Roska, and L. O. Chua "New Spatial-Temporal Patterns and The First Programmable On-Chip Bifurcation Test-Bed", *IEEE Trans. on Circuits and Systems I, (TCAS I.)*, Vol. 50(5), pp. 619-633, May 2003.

[58] A. Gacsádi and P. Szolgay, "Image Inpainting Methods by Using Cellular Neural Networks", *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'2005)*, ISBN:0780391853, pp. 198-201, Hsinchu, Taiwan, 2005

[59] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variational based noise removal algorithms", *Physica D*, Vol. 60, pp. 259–268, 1992.

[60] A. Gacsádi, P. Szolgay, "A variational method for image denoising by using cellular neural networks", *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'2004)*, ISBN 963-311-357-1, pp. 213-218, Budapest, Hungary, 2004.

[61] R. Matei, "New Image Processing Tasks On Binary Images Using Standard CNNs", *Proceedings of the International Symposium on Signals, Circuits and Systems, SCS'2001,* pp.305-308, July 10-11, 2001, Iaşi, Romania

[62] R. Matei, "Design Method for Orientation-Selective CNN Filters", *Proceedings of the IEEE International Symposium on Circuits and Systems ISCAS'2004*, May 23-26, 2004, Vancouver, Canada

[63] CNN Young Researcher Contest, Analogic CNN Algorithm Design, *7th IEEE International Workshop on Cellular Neural Networks and their Applications*, Frankfurt-Germany, July 2002.

[64] D. Bálya: "CNN Universal Machine as Classification Platform: an ART-like Clustering Algorithm", *Int. Journal of Neural Systems*, 2003, Vol. 13(6), pp. 415-425.

[65] B. Roska, F. Werblin, "Rapid global shifts in natural scenes block spiking in specific ganglion cell types", *Nature Neuroscience*, 2003

[66] D. Bálya "Sudden Global Spatial-Temporal Change Detection and its Applications", *Journal of Circuits,* Systems*, and* Computers *(JCSC)*, Vol. 12(5), Aug-Dec 2003.

[67] Gy. Cserey, Cs. Rekeczky and P. Földesy "PDE Based Histogram Modification With Embedded Morphological Processing of the Level-Sets", *Journal of Circuits, System and Computers* (JCSC 2002).

[68] K. Karacs, G. Prószéky and T. Roska, "Intimate Integration of Shape Codes and Linguistic Framework in Handwriting Recognition via Wave Computers", *European Conference on Circuit Theory and Design*, Kraków, Poland, Sept. 2003.

[69] Z. Szlávik, T. Szirányi, "Face Identification with CNN-UM", *European Conference on Circuit Theory and Design*, Kraków, Poland, Sept. 2003.

[70] Cs. Rekeczky, G. Tímár, and Gy. Cserey "Multi-Target Tracking With Stored Program Adaptive CNN Universal Machines" in *Proc. 7th IEEE International Workshop on Cellular Neural Networks and their Applications*, Frankfurt am Main, Germany, July 22-24, 2002., pp. 299-306.

[71] L. Török, Á. Zarándy "CNN Optimal Multiscale Bayesian Optical Flow Calculation", *European Conference on Circuit Theory and Design*, Kraków, Poland, Sept. 2003.

[72] Z. Fodróczi, A. Radványi "Computational Auditory Scene Analysis in Cellular Wave Computing Framework" *International Journal of Circuit Theory and Applications,* Vol: 34(4) pp: 489-515, ISSN:0098-9886 (July 2006)

[73] L. Kék and Á. Zarándy, "Implementation of Large-Neighborhood Nonlinear Templates on the CNN Universal Machine", *International Journal of Circuit Theory and Applications,* Vol. 26, No. 6, pp. 551-566, 1998.

[74] G. Constantini, D. Casali, M. Carota, and R. Perfetti, Translation and Rotation of Grey-Scale Images by means of Analogic Cellular Neural Network, *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'2004)*, ISBN 963-311-357-1, pp. 213-218, Budapest, Hungary, 2004.

INDEX

INDEX (OLD NAMES)