CNN Software Library for ACE4K Chip
(Templates and Algorithms)

Version 1.0

Budapest

2000
# TABLE OF CONTENTS

1. TEMPLATES/INSTRUCTIONS ........................................................................................................ 1

## 1.1. BASIC IMAGE PROCESSING ................................................................................................. 1

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GradientIntensityEstimation</td>
<td>1</td>
</tr>
<tr>
<td>Estimation of the gradient intensity in a local neighborhood</td>
<td></td>
</tr>
<tr>
<td>DiagonalHoleDetection</td>
<td>4</td>
</tr>
<tr>
<td>Detects the number of diagonal holes from each diagonal line</td>
<td></td>
</tr>
<tr>
<td>CenterPointDetector</td>
<td>6</td>
</tr>
<tr>
<td>Center point detection</td>
<td></td>
</tr>
<tr>
<td>ContourExtraction</td>
<td>10</td>
</tr>
<tr>
<td>Grayscale contour detector</td>
<td></td>
</tr>
<tr>
<td>CornerDetection</td>
<td>13</td>
</tr>
<tr>
<td>Convex corner detector</td>
<td></td>
</tr>
<tr>
<td>VerticalLineRemover</td>
<td>15</td>
</tr>
<tr>
<td>Deletes vertical lines</td>
<td></td>
</tr>
<tr>
<td>DiagonalLineDetector</td>
<td>18</td>
</tr>
<tr>
<td>Diagonal-line-detector template</td>
<td></td>
</tr>
<tr>
<td>EdgeDetection</td>
<td>20</td>
</tr>
<tr>
<td>Binary edge detection template</td>
<td></td>
</tr>
<tr>
<td>OptimalEdgeDetector</td>
<td>22</td>
</tr>
<tr>
<td>Optimal edge detector template</td>
<td></td>
</tr>
<tr>
<td>PointExtraction</td>
<td>24</td>
</tr>
<tr>
<td>Extracts isolated black pixels</td>
<td></td>
</tr>
<tr>
<td>PointRemoval</td>
<td>26</td>
</tr>
<tr>
<td>Deletes isolated black pixels</td>
<td></td>
</tr>
<tr>
<td>SelectedObjectsExtraction</td>
<td>28</td>
</tr>
<tr>
<td>Extracts marked objects</td>
<td></td>
</tr>
<tr>
<td>3x3Halftoning</td>
<td>30</td>
</tr>
<tr>
<td>3x3 image halftoning</td>
<td></td>
</tr>
<tr>
<td>Hole-Filling</td>
<td>32</td>
</tr>
<tr>
<td>Hole-Filling</td>
<td></td>
</tr>
<tr>
<td>ObjectIncreasing</td>
<td>34</td>
</tr>
<tr>
<td>Increases the object by one pixel (DTCNN)</td>
<td></td>
</tr>
<tr>
<td>LocalSouthernElementDetector</td>
<td>36</td>
</tr>
<tr>
<td>Local southern element detector</td>
<td></td>
</tr>
<tr>
<td>RightEdgeDetection</td>
<td>38</td>
</tr>
<tr>
<td>Extracts right edges of objects</td>
<td></td>
</tr>
<tr>
<td>ShadowProjection</td>
<td>40</td>
</tr>
<tr>
<td>Projects onto the left the shadow of all objects illuminated from the right</td>
<td></td>
</tr>
<tr>
<td>VerticalShadow</td>
<td>42</td>
</tr>
<tr>
<td>Vertical shadow template</td>
<td></td>
</tr>
</tbody>
</table>
1.3. SPATIAL LOGIC

ConcaveLocationFiller
Fills the concave locations of objects

GrayscaleLineDetector
Grayscale line detector template

LogicANDOperation
Logic "AND" operation

LogicOROperation
Logic "OR" and Set Union \( \cup \) (Disjunction \( \vee \)) template

PatchMaker
Patch maker template

SmallObjectRemover
Deletes small objects

1.4. TEXTURE SEGMENTATION AND DETECTION

3x3TextureSegmentation
Segmentation of four textures by a 3*3 template

GameofLife1Step
Simulates one step of the game of life

2. SUBROUTINES

EDGE CONTROLLED DIFFUSION

REFERENCES

INDEX
1. Templates/Instructions
1.1. BASIC IMAGE PROCESSING

GradientIntensityEstimation: Estimation of the gradient intensity in a local neighborhood

Old names: AVERGRAD

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad \begin{bmatrix}
b & b & b \\
b & 0 & b \\
b & b & b \\
\end{bmatrix} \quad z = \begin{bmatrix} 0 \end{bmatrix}
\]

where \( b = \frac{|v_{ij} - v_{kl}|}{8}. \)

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: \( Y(t) \Rightarrow Y(\infty) = \) Grayscale image representing the estimated average gradient intensity in a local neighborhood in \( P \).

II. Examples

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

III. ACE4K implementation

Implementation method: optimization simplification.

GradIntEstimation_{ACE4K}: (Full-range model, ACE4K)
Horidiffc.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 0 \quad z_2 = 2.45 \]

Verdiffc.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 0 \quad z_2 = 1.45 \]

NegLAMLLM.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 0 \quad z_2 = 1.1 \]

AbsVal.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.9 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 1.7 \quad z_2 = 0 \]

AverHor.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 0.8 \quad z_2 = 0 \]

AverVer.tem

\[ A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.75 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 1 \quad z_2 = 0 \]

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

- Masking works much better with DTCNN;
- The current values has to be set peculiarly for all the template operations;
- Central and non-central elements in the B templates behave different way…
- I used a simplified form of the original template: the diagonal directions are the sum of the horizontal and vertical gradients and so the diagonal template elements could be omitted;
- In this chip implementation averaging is omitted, because the algorithm has a bit superior performance, than the original nonlinear template (However, it can be included into the aververc.tem and averhorc.tem templates…).
- The 2nd order gradient can be computed much easier (for the above test images provide almost the same result): Antagonistic Center-surround template and an Absolute Value template.
DiagonalHoleDetection: Detects the number of diagonal holes from each diagonal line [6]

Old names: CCD_DIAG (Chua-Yang model)

\[
\begin{array}{ccc}
1 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & -1 \\
\end{array}
\quad \begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad z = 0
\]

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: \( Y(t) \Rightarrow Y(\infty) = \) 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

III. ACE4K implementation

Implementation method:

DiagonalHoleDetection_ACE4K: (Full-range model, ACE4K)

\[
\begin{array}{ccc}
2.3 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & -2.3 \\
\end{array}
\quad \begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad z_1 = 0 \quad z_2 = 0
\]

Example 1 (resolution: 64x64): image name: sc_09.bmp, template name: ccd_diag_se_ace4k.tem.
Remarks:
- Image should be loaded into a LAM;
- Repeat template operation a few times.
CenterPointDetector:  Center point detection [21]

Old names: CENTER

\[
\begin{align*}
A_1 &= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} & B_1 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & z_1 &= -1 \\
A_2 &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 0 \\ 1 & 0 & -1 \end{bmatrix} & B_2 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & z_2 &= -1 \\
A_3 &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 4 & 0 \\ 0 & -1 & 0 \end{bmatrix} & B_3 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & z_3 &= -1 \\
A_4 &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 6 & 1 \\ -1 & 0 & 1 \end{bmatrix} & B_4 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & z_4 &= -1 \\
& \ldots & & & \\
A_8 &= \begin{bmatrix} 1 & 0 & -1 \\ 1 & 6 & 0 \\ 1 & 1 & 1 \end{bmatrix} & B_8 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & z_8 &= -1 
\end{align*}
\]

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: \( Y(t) \Rightarrow Y(\infty) = \) 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:**

\[
A_1 = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B_1 = \begin{bmatrix}
1 & 0 & 0 \\
1 & 4 & -1 \\
1 & 0 & 0 \\
\end{bmatrix}
\quad z_1 = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

**CENTER2:**

\[
A_2 = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B_2 = \begin{bmatrix}
1 & 1 & 1 \\
1 & 6 & 0 \\
1 & 0 & -1 \\
\end{bmatrix}
\quad z_2 = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

**CENTER3:**

\[
A_3 = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B_3 = \begin{bmatrix}
1 & 1 & 1 \\
0 & 4 & 0 \\
0 & -1 & 0 \\
\end{bmatrix}
\quad z_3 = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

**CENTER4:**

\[
A_4 = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B_4 = \begin{bmatrix}
1 & 1 & 1 \\
0 & 6 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}
\quad z_4 = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

\[\ldots\]

**CENTER8:**

\[
A_8 = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B_8 = \begin{bmatrix}
1 & 0 & -1 \\
1 & 6 & 0 \\
1 & 1 & 1 \\
\end{bmatrix}
\quad z_8 = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

The robustness of templates CENTER1 and CENTER2 are \(\rho(\text{CENTER1}) = 0.22\) and \(\rho(\text{CENTER2}) = 0.15\), 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: chinese.bmp, image size: 16x16; template name: center.tem

![Input Image](image)

![Output](image)

**III. ACE4K implementation**

*Implementation method:* optimization.

*CenterPoint_ACE4K: (Full-range model, ACE4K)*

*Discrete time CNN implementation:*
\[
\begin{bmatrix}
A_1 &=& 01 & 0 & 0 \\
      &=& 0 & 1 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_1 &=& 1 & 0 & 0 \\
      &=& 1 & 3 & -1 \\
      &=& 1 & -1 & 0 \\
\end{bmatrix} \quad 
z_1 = -1.95
\]

\[
\begin{bmatrix}
A_2 &=& 0 & 0 & 0 \\
      &=& 0 & 2 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_2 &=& .66 & .66 & .66 \\
      &=& .66 & 2 & 0 \\
      &=& .66 & 0 & -1.66 \\
\end{bmatrix} \quad 
z_2 = -1
\]

\[
\begin{bmatrix}
A_3 &=& 0 & 0 & 0 \\
      &=& 0 & 0 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_3 &=& .75 & .75 & .75 \\
      &=& 0 & 3 & 0 \\
      &=& 0 & -1.75 & 0 \\
\end{bmatrix} \quad 
z_3 = -1.75
\]

\[
\begin{bmatrix}
A_4 &=& 0 & 0 & 0 \\
      &=& 0 & 3 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_4 &=& 1 & 1 & 1 \\
      &=& 0 & 3 & 1 \\
      &=& -1 & 0 & 1 \\
\end{bmatrix} \quad 
z_4 = -1.95
\]

\[
\begin{bmatrix}
A_8 &=& 0 & 0 & 0 \\
      &=& 0 & 3 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_8 &=& 1 & 0 & -1 \\
      &=& 1 & 3 & 0 \\
      &=& 1 & 1 & 1 \\
\end{bmatrix} \quad 
z_8 = -1.95
\]

Continuous time CNN implementation:

\[
\begin{bmatrix}
A_1 &=& 0 & 0 & 0 \\
      &=& 0 & -1 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_1 &=& 1 & 0 & 0 \\
      &=& 1 & 3 & -1 \\
      &=& 1 & -1 & 0 \\
\end{bmatrix} \quad 
z_1 = -2.5
\]

\[
\begin{bmatrix}
A_2 &=& 0 & 0 & 0 \\
      &=& 0 & -1 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_2 &=& 1 & 1 & 1 \\
      &=& 1 & 3 & 0 \\
      &=& 1 & -1 & 0 \\
\end{bmatrix} \quad 
z_2 = -6
\]

\[
\begin{bmatrix}
A_3 &=& 0 & 0 & 0 \\
      &=& 0 & -1.3 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_3 &=& 1 & 1 & 1 \\
      &=& 0 & 2 & 0 \\
      &=& 0 & -1 & 0 \\
\end{bmatrix} \quad 
z_3 = -6
\]

\[
\begin{bmatrix}
A_4 &=& 0 & 0 & 0 \\
      &=& 0 & -1 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_4 &=& 1 & 1 & 1 \\
      &=& 0 & 3 & 1 \\
      &=& -1 & 0 & 1 \\
\end{bmatrix} \quad 
z_4 = -6
\]

\[
\begin{bmatrix}
\begin{bmatrix}
A_8 &=& 0 & 0 & 0 \\
      &=& 0 & 3 & 0 \\
      &=& 0 & 0 & 0 \\
\end{bmatrix} \quad 
\begin{bmatrix}
B_8 &=& 1 & 0 & -1 \\
      &=& 1 & 3 & 0 \\
      &=& 1 & 1 & 1 \\
\end{bmatrix} \quad 
z_8 = -6
\]
\]

\[
\ldots
\]
$A_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ \hspace{1cm} $B_8 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 3 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ \hspace{1cm} $z_8 = -6$

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

Remarks:
- Before template running LLMs must be initialized with white;
- The logic values (zeros and ones) seems to be reversed in the chip…
- Both the discrete and continuous time CNN implementations are realizable on chip, but the later one is more sensitive for timing etc.
- The current values are extremely depends on the temperature of the chip and does not accord to the theoretical values…
- Continuous time version sometimes fail…
ContourExtraction: Grayscale contour detector [8]

Old names: ContourDetector, CONTOUR

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
a & a & a \\
a & 0 & a \\
a & a & a \\
\end{bmatrix}
\]

where \( a \) is defined by the following nonlinear function:

\[
\begin{array}{c}
| a \\
0.5 \\
0.18 \\
-0.18 \\
\end{array}
\]

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: \( Y(t) \Rightarrow Y(\infty) = \) 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.
III. ACE4K implementation

Implementation method: decomposition and optimization.

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

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

Horizontally, on the right:

Horizontally, on the right:

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & -3 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 3 & -3 \\
0 & 0 & 0 \\
\end{array}
\]

\[
z_1 = -0.1 \quad z_2 = 0
\]

With opposite sign:

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & -3 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & -3 & 3 \\
0 & 0 & 0 \\
\end{array}
\]

\[
z_1 = -0.1 \quad z_2 = 0
\]

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

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & -3 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
z_1 = 1 \quad z_2 = 0
\]
Example (resolution: 64x64): image name: madonna.bmp.

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

**Old names:** CornerDetector, CORNER

**CornerDetection (Chua-Yang model):**

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad B =
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 4 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\quad z = -5
\]

**I. Global Task**

Given: static binary image \( P \)

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

Initial State: \( X(0) = \text{Arbitrary} \) (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: \( Y(t) \Rightarrow Y(\infty) = \) Binary image where black pixels represent the convex corners of objects in \( P \).

Template robustness: \( \rho = 0.2 \).

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 and output images](image)

**III. ACE4K implementation**

Implementation method: optimization.

**CornerDetection_ACE4K: (Full-range model, ACE4K)**

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad B =
\begin{bmatrix}
-1.3 & -1.3 & -1.3 \\
-1.3 & 0 & -1.3 \\
-1.3 & -1.3 & -1.3
\end{bmatrix}
\quad z_1 = -5 \quad z_2 = 0
\]

**Example 1** (resolution: 64x64): image name: corner.bmp, template name: cornerdetection_ace4k.tem.
Remarks:

- We can use this template in the LAMs.
**VerticalLineRemover:** Deletes vertical lines [8]

*Old names:* DELVERT1

*Delvert1 (Chua-Yang model):*

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0 & -1 & 0 \\
0 & 1 & 0 \\
0 & -1 & 0
\end{bmatrix} \quad z = -2
\]

**I. Global Task**

*Given:* static binary image \( P \)

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

*Initial State:* \( X(0) = \) Arbitrary (in the examples we choose \( x_B(0) = 0 \))

*Boundary Conditions:* Fixed type, \( u_U = -1 \), for all virtual cells, denoted by \([U] = -1\)

*Output:* \( Y(t) \Rightarrow Y(\infty) = \) 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:* \( \rho = 0.58 \).

*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.
III. ACE4K implementation

Implementation method: optimization.

Delvert_ACE4K: (Full-range model, ACE4K)

\[ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1.2 & 0 \\ 0 & -1 & 0 \end{bmatrix} \]

\[ z_1 = -3 \quad z_2 = 0 \]

Delhor_ACE4K: (Full-range model, ACE4K)

\[ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1.2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ z_1 = -3 \quad z_2 = 0 \]

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

[Image of input and output]

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

[Image of input and output]
Remarks:
- Template operations should be executed twice successively.
DiagonalLineDetector: Diagonal-line-detector template

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

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B =
\begin{bmatrix}
-1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & -1 \\
\end{bmatrix}
\quad z = -4
\]

I. Global Task

Given: static binary image \( P \)

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

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

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

Output: \( Y(t) = Y(\infty) = \) 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 \( \pm 1 \) 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.

III. ACE4K implementation

Implementation method: optimization

\( \text{DIAG1LIU\_ACE4K: (Full-range model, ACE4K)} \)

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B =
\begin{bmatrix}
-1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & -1 \\
\end{bmatrix}
\quad z_1 = -5 \quad z_2 = -5
\]
Example 1 (resolution: 64x64): image name: detdiag.bmp, template name: diag1liu_ace4k.tem.

![Input Image](image1.png) ![Output Image](image2.png)

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

![Input Image](image3.png) ![Output Image](image4.png)

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

**Old names:** EdgeDetector, EDGE

**EdgeDetection (Chua-Yang model):**

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\quad z = \begin{bmatrix}
-1 \\
\end{bmatrix}
\]

**I. Global Task**

Given: static binary image \( P \)

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

Initial State: \( X(0) = \text{Arbitrary} \) (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: \( Y(t) \Rightarrow Y(\infty) = \text{Binary image showing all edges of } P \text{ in black} \)

Template robustness: \( \rho = 0.12 \).

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

**II. Example**

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

![input output](image)

**III. ACE4K implementation**

Implementation method: template decomposition and optimization.

Result: \( \text{EdgeDetection} \iff \text{Edge1 AND INPUT} \)

**Edge1 (Chua-Yang model):**

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\quad B = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 0 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\quad z = \begin{bmatrix}
7 \\
\end{bmatrix}
\]
Edge1_ACE4K: (Full-range model, ACE4K)

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 0 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix} \quad z_1 = 6 \quad z_2 = 4.8
\]

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

![input](image1)

![output](image2)

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

![input](image3)

![output](image4)

Remarks:
- Template operations should be executed twice in a row.
**OptimalEdgeDetector:**  
*Optimal edge detector [43]*

**OptimalEdgeDetector (Chua-Yang model):**

\[
A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -0.11 & 0 & 0.11 \\ -0.28 & 0 & 0.28 \\ -0.11 & 0 & 0.11 \end{bmatrix} \quad z = \begin{bmatrix} 0 \end{bmatrix}
\]

**I. Global Task**

Given: static grayscale image \( P \)

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

Initial State: \( X(0) = \) Arbitrary (in the examples we choose \( x_0(0)=0 \))

Boundary Conditions: Zero-flux boundary condition (duplicate)

Output: \( Y(t) \Rightarrow Y(\infty) = \) 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: optiedge.tem.

**III. ACE4K implementation**

Implementation method: optimization.

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

\[
A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -0.33 & 0 & 0.33 \\ -0.84 & 0 & 0.84 \\ -0.33 & 0 & 0.33 \end{bmatrix} \quad z_1 = \begin{bmatrix} 0.7 \end{bmatrix} \quad z_2 = \begin{bmatrix} 0 \end{bmatrix}
\]
Example 1 (resolution: 64x64): image name: bird64_i.bmp, template name: optimedge_ace4k.tem.

![input](image)

![output](image)

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

![inputs](image)

![outputs](image)

Remarks:
- Due to memory problems the original 256x256 image was cut into 256x128 sub-images, and processed in two phases.
**PointExtraction:** Extracts isolated black pixels

*Old names:* FigureRemover, FIGDEL

*PointExtraction (Chua-Yang model):*

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & -1
\end{bmatrix} \quad z = -8
\]

**I. Global Task**

*Given:* static binary image \( P \)

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

*Initial State:* \( X(0) = \text{Arbitrary} \)

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

*Output:* \( Y(t) \Rightarrow Y(\infty) = \text{Binary image representing all isolated black pixels in } P. \)

*Template robustness:* \( \rho = 0.33 \).

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

**III. ACE4K implementation**

*Implementation method:* optimization.

*PointExtraction_ACE4K: (Full-range model, ACE4K)*

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1.5 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 0.5 & -1 \\
-1 & -1 & -1
\end{bmatrix} \quad z_1 = -6 \quad z_2 = 0
\]
Example 1 (resolution: 64x64): image name: points.bmp, template name: PointExtraction_ace4k.tem.

Remarks:
- This template can be used in LAMs.
**PointRemoval:** Deletes isolated black pixels

*Old names:* FigureExtractor, FIGEXTR

**PointRemoval (Chua-Yang model):**

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
1 & 1 & 1 \\
1 & 8 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} \quad z = -1
\]

**I. Global Task**

- **Given:** static binary image \( P \)
- **Input:** \( U(t) = P \)
- **Initial State:** \( X(0) = \) Arbitrary
- **Boundary Conditions:** Fixed type, \( u_{ij} = 0 \) for all virtual cells, denoted by \([U] = 0\)
- **Output:** \( Y(t) \Rightarrow Y(\infty) = \) Binary image showing all connected components in \( P \).
  
**Remark:** Black pixels having no black neighbors are deleted. This template is the opposite of PointExtraction.

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

- **input**
  - ![Input Image](image1)
- **output**
  - ![Output Image](image2)

**III. ACE4K implementation**

- **Implementation method:** optimization.
- **PointRemoval_ACE4K: (Full-range model, ACE4K)**

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & -2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0.5 & 0.5 & 0.5 \\
0.5 & 3 & 0.5 \\
0.5 & 0.5 & 0.5 \\
\end{bmatrix} \quad z_1 = \begin{bmatrix} 0.5 \end{bmatrix} \quad z_2 = \begin{bmatrix} 0 \end{bmatrix}
\]
Example 1 (resolution: 64x64): image name: points.bmp, template name: PointRemoval_ace4k.tem.

input

output

Remarks:
1) We can use this template in the LLMs.
**SelectedObjectsExtraction:** Extracts marked objects

**Old names:** FigureReconstructor, FIGREC, RECALL (Chua-Yang model)

**SelectedObjectsExtraction (Chua-Yang model):**

\[
\begin{bmatrix}
0.5 & 0.5 & 0.5 \\
0.5 & 4 & 0.5 \\
0.5 & 0.5 & 0.5
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 4 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

\[z = 3\]

**I. Global Task**

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

**Input:** \(U(t) = P_1\)

**Initial State:** \(X(0) = P_2\)

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

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

**Template robustness:** \(\rho = 0.12\).

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

[Input, Initial State, Output images]

**III. ACE4K implementation**

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

**SelectedObjectsExtraction\_ACE4K: (Full-range model, ACE4K)**

\[
\begin{bmatrix}
0.41 & 0.59 & 0.41 \\
0.59 & 1.80 & 0.59 \\
0.41 & 0.59 & 0.41
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

\[z_1 = 4, \quad z_2 = 0\]
Remarks:
- Initial state should be a LLM;
- Fixed state should be the LLM4 and white pixels denotes for positions where cell transient can take place.
- Input image should be a LAM filled with zero.

Example 1 (resolution: 64x64): image name: recall1.bmp, template name: recall_ace4k.tem.

Example 2 (resolution: 64x64): image name: recall2.bmp, template name: recall_ace4k.tem.
3x3 Halftoning: 3x3 image halftoning

Old names: HLF3, HLF33

3x3 Halftoning (Chua-Yang model):

\[
\begin{align*}
A &= \begin{bmatrix}
-0.07 & -0.1 & -0.07 \\
-0.1 & 1+\varepsilon & -0.1 \\
-0.07 & -0.1 & -0.07 \\
\end{bmatrix},
B &= \begin{bmatrix}
0.07 & 0.1 & 0.07 \\
0.1 & 0.32 & 0.1 \\
0.07 & 0.1 & 0.07 \\
\end{bmatrix},
\end{align*}
\]

\[
z = 0
\]

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: $Y(t) \Rightarrow Y(\infty) = \text{Binary image preserving the main features of } P$

Remark: The speed of convergence is controlled by $\varepsilon \approx [0.1...1]$. The greater the $\varepsilon$ is, the faster the process and the rougher the result will be. The inverse of the template is $3x3 \text{InverseHalftoning}$. 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

III. ACE4K implementation

Implementation method: optimization.

3x3 Halftoning_ACE4K: (Full-range model, ACE4K)
\[ \mathbf{A} = \begin{bmatrix} -0.07 & -0.15 & -0.07 \\ -0.15 & 1.15 & -0.15 \\ -0.07 & -0.15 & -0.07 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.07 & 0.15 & 0.07 \\ 0.15 & 0.15 & 0.15 \\ 0.07 & 0.15 & 0.07 \end{bmatrix} \quad z_1 = 1.25 \quad z_2 = 0 \]

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

Remarks:
- There is some bias on the chip surface (top right corner is whiter, than it should be…).
**Hole-Filling:** *Fills the interior of all closed contours [6]*

**Old names:** HoleFiller, HOLE

**Hole-Filling (Chua-Yang model):**

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 3 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 4 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[z = -1\]

**I. Global Task**

Given: static binary image \(P\)

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

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

Boundary Conditions: Fixed type, \(y_i = 0\) for all virtual cells, denoted by \([Y]=0\)

Output: \(Y(t)\Rightarrow Y(\infty) = \) Binary image representing \(P\) with holes filled.

Remark:
1. this is a propagating template, the computing time is proportional to the length of the image
2. 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.

**III. ACE4K implementation**

Implementation method: optimization.

**Hole-Filling\_ACE4K:** (Full-range model, ACE4K)

\[
\begin{bmatrix}
0 & 0.68 & 0 \\
0.68 & 1.71 & 0.68 \\
0 & 0.68 & 0 \\
\end{bmatrix}
\]

\[z_1 = -0.95 \quad z_2 = -6\]

**Hole-FillingDT\_ACE4K:** (DT-CNN mode, ACE4K)

\[
\begin{bmatrix}
0 & 3 & 0 \\
3 & 1 & 3 \\
0 & 3 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[z_1 = 4 \quad z_2 = 0\]
Example 1 (resolution: 64x64): image name: hole_i.bmp, template name: hole_ace4k.tem, holeDT_ace4k.tem.

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

Remarks:
- Special settings (continuous mode): hw.set.ref 0  60 -90 -81 16 -51 51 113 84
- Special settings (DTCNN mode): hw.set.mode 1 2
**ObjectIncreasing:** Increases the object by one pixel (DTCNN) [16]

**Old names:** INCREASE

\[ \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = 4 \]

**I. Global Task**

Given: static binary image \( P \)

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

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

Boundary Conditions: Zero-flux boundary condition (duplicate)

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

**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.

**III. ACE4K implementation**

Implementation method: optimization.

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

\[ \begin{bmatrix} 0 & 1.6 & 0 \\ 1.6 & -3 & 1.6 \\ 0 & 1.6 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z_1 = 0 \quad z_2 = 0 \]

Example 1 (resolution: 64x64): image name: aletter.bmp, template name: cornerdetection_ace4k.tem.
**ObjectIncreasing_ACE4K: (Full-range model, ACE4K)**

Example 1

\[
\mathbf{A} = \begin{bmatrix}
0 & 0 & 0 \\
0 & -2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad \mathbf{B} = \begin{bmatrix}
0.5 & 0.5 & 0.5 \\
0.5 & 2 & 0.5 \\
0.5 & 0.5 & 0.5 \\
\end{bmatrix} \\
z_1 = \begin{bmatrix} 0 \end{bmatrix} \quad z_2 = \begin{bmatrix} 0 \end{bmatrix}
\]

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

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

**Remarks:**
- The Examples 1 and 2 were run in the LAMs. The Example 3 was run in the LLMs.
LocalSouthernElementDetector: Local southern element detector [11]

Old names: LSE

LocalSouthernElementDetector (Chua-Yang model):

\[
A = \begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad B = \begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
-1 & -1 & -1 \\
\end{array}
\quad z = -3
\]

I. Global Task

Given: static binary image \( P \)

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

Initial State: \( X(0) = \text{Arbitrary} \)

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

Output: \( Y(t) \Rightarrow Y(\infty) = \text{Binary image representing local southern elements of objects in } P. \)

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.

III. ACE4K implementation

Implementation method: optimization.

LocalSouthernElementDetector_ACE4K: (Full-range model, ACE4K)

\[
A = \begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad B = \begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad z_1 = -3.5 \quad z_2 = 0
\]

Example 1 (resolution: 64x64): image name: local.bmp, template name: localsouthernelementdetector_ace4k.tem.
Remarks:

- This template can be used in the LAMs.
RightEdgeDetection: Extracts right edges of objects

Old names: RightContourDetector, RIGHTCON

RightEdgeDetection (Chua-Yang model):

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
1 & 1 & -1 \\
0 & 0 & 0
\end{bmatrix} \quad z = -2
\]

I. Global Task

Given: static binary image P

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

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

Boundary Conditions: Fixed type, \( u_i = 0 \) for all virtual cells, denoted by \([U]=0\)

Output: \( Y(t) \Rightarrow Y(\infty) = \) Binary image representing the right edges of objects in P.

Template robustness: \( \rho = 0.58 \).

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

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

III. ACE4K implementation

Implementation method: optimization.

RightEdgeDetection_ACE4K: (Full-range model, ACE4K)

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
2 & 3 & -2 \\
0 & 0 & 0
\end{bmatrix} \quad z_1 = -6 \quad z_2 = 0
\]
Example 1 (resolution: 64x64): image name: corner.bmp, template name: rightedgedetection_ace4k.tem.

Remarks:
- This template can be used in the LAMs.
**ShadowProjection:** Projects onto the left the shadow of all objects illuminated from the right [6]

Old names: LeftShadow, SHADOW

ShadowProjection (Chua-Yang model):

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 2 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad z = \begin{bmatrix}
0 \\
\end{bmatrix}
\]

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

Template robustness: \( = 0.12 \).

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

II. Example

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

**III. ACE4K implementation**

Implementation method: optimization.

ShadowProjection _ACE4K: (Full-range model, ACE4K)

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 2 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad z_1 = \begin{bmatrix}
1 \\
\end{bmatrix} \quad z_2 = \begin{bmatrix}
0 \\
\end{bmatrix}
\]

Example 1 (resolution: 64x64): image name: shadow.bmp, template name: shadow_ace4k.tem.
Remarks:
- The execution of the template could not be solved using LLM-s. The loading of logical TRUE in the initial state was also faulty. There was no problem by using LAM-s. LAM with value 1 was used for the initial state.
- The template worked only in a loop, after many executions. The range of the shadow effect increased continuously in the repetitions.
VerticalShadow: Vertical shadow template

Old names: SHADSIM, SUPSHAD

VerticalShadow (Chua-Yang model):

\[
\begin{align*}
A &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{pmatrix} & B &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & z &= 2
\end{align*}
\]

I. Global Task

Given: static binary image \( P \)

Input: \( U(t) = \text{Arbitrary} \)

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

Boundary Conditions: Zero-flux boundary condition (duplicate)

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

Template robustness: \( = 0.12 \).

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

II. Example

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

III. ACE4K implementation

Implementation method: optimization.

VerticalShadow _ACE4K: (Full-range model, ACE4K)

\[
\begin{align*}
A &= \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} & B &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & z_1 &= 3.5 & z_2 &= 0
\end{align*}
\]

Example 1 (resolution: 64x64): image name: skelbwi64.bmp, template name: shadsim_ace4k.tem.
Remarks:
- The execution of the template could not be solved using LLM-s.
- The template worked only in a loop, after many executions. The range of the shadow effect increased continuously in the repetitions.
1.3. SPATIAL LOGIC

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

*Old names:* HOLLOW

*ConcaveLocationFiller (Chua-Yang model):*

\[
\begin{bmatrix}
A & = & 0.5 & 0.5 & 0.5 \\
 B & = & 0 & 0 & 0 \\
 z & = & 3.25
\end{bmatrix}
\]

**I. Global Task**

*Given:* static binary image \( P \)

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

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

*Boundary Conditions:* Fixed type, \( y[i] = 0 \) for all virtual cells, denoted by \( [Y]=0 \)

*Output:* \( Y(t) \Rightarrow Y(\infty) = \) 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.
III. ACE4K implementation

Implementation method:

ConcaveLocationFiller_ACE4K: (Full-range model, ACE4K)

\[
A = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & -3 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad z_1 = 1, \quad z_2 = 1.0
\]

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

Remarks:
- Initial State: white.bmp
GrayscaleLineDetector: Grayscale line detector template

Old names: LINE3060

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1.5 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad B = \begin{bmatrix}
b & a & a \\
b & 0 & a \\
a & b & b
\end{bmatrix}
\quad z = -4.5
\]

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: \(Y(t) \Rightarrow Y(\infty) = \) 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.
III. ACE4K implementation

Implementation method: optimization.

GrayscaleLineDetector _ACE4K: (Full-range model, ACE4K)

‘Center –surround’ template by continuous time cnn.

\[
A = \begin{pmatrix}
-0.2 & -0.20 & 0 \\
-0.20 & 1.2 & -0.2 \\
0 & -0.2 & -0.2 \\
\end{pmatrix}
\]

\[
B = \begin{pmatrix}
-0.2 & -0.2 & 0 \\
-0.2 & 1.2 & -0.2 \\
0 & -0.2 & -0.2 \\
\end{pmatrix}
\]

\[
z_1 = \begin{pmatrix}
-0.5 \\
0 \\
\end{pmatrix}
\]

Line detection template

\[
A = \begin{pmatrix}
0 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
\]

\[
B = \begin{pmatrix}
-0.75 & 0.25 & 0.25 \\
-0.75 & 1 & -0.25 \\
0.25 & -0.75 & -0.75 \\
\end{pmatrix}
\]

\[
z_1 = \begin{pmatrix}
0 \\
0.2 \\
\end{pmatrix}
\]

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

Remarks:

- Before template running LLMs must be initialized with white.
**LogicANDOperation:** Logic AND and Set Intersection $\cap$ (Conjunction $\land$) template

*Old names:* LogicAND, LOGAND, AND

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>-1</td>
</tr>
<tr>
<td>0 2 0</td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

**I. Global Task**

*Given:* two static binary images $P_1$ and $P_2$

*Input:* $U(t) = P_1$

*Initial State:* $X(0) = P_2$

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

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

**III. ACE4K implementation**

*Implementation method:* optimization.

LogAND_ACE4K: (Full-range model, ACE4K)

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$z_1$</th>
<th>$z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 2 0</td>
<td>0 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Example1* (resolution: 64x64): image name: striphor.bmp and stripver.bmp, template name: logand.tem.
Remarks:

- Because of the experienced interference between binary pictures put in a common template operation robust operation could achieved only by the use of fixed map which works very reliably.
- The test AMC code can be found here:
**LogicOROperation:**  Logic OR and Set Union \( \cup \) (Disjunction \( \lor \)) template

**Old names:** LogicOR, LOGOR, OR

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad z = \begin{bmatrix}
1 \\
\end{bmatrix}
\]

I. **Global Task**

Given: two static binary images \( P_1 \) and \( P_2 \)

*Input:* \( U(t) = P_1 \)

*Initial State:* \( X(0) = P_2 \)

*Output:* \( Y(t) \Rightarrow Y(\infty) = \) binary output of the logic operation OR between \( P_1 \) and \( P_2 \). In logic notation, \( Y(\infty) = P_1 \lor P_2 \), where \( \lor \) denotes the “disjunction” operator. In set-theoretic notation, \( Y(\infty) = P_1 \cup P_2 \) where \( \cup \) denotes the “set union” operator.

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

![input](image1)

![initial state](image2)

![output](image3)

III. **ACE4K implementation**

*Implementation method:* optimization.

*LogOR_ACE4K: (Full-range model, ACE4K)*

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad z_1 = \begin{bmatrix}
0 \\
\end{bmatrix} \quad z_2 = \begin{bmatrix}
0 \\
\end{bmatrix}
\]
Example 1 (resolution: 64x64): image name: striphor.bmp and stripver.bmp, template name: logor.tem.

Remarks:

- Because of the experienced interference between binary pictures put in a common template operation robust operation could achieved only by the use of fixed map which works very reliably.
PatchMaker: Patch maker template [22]

Old names: PATCHMAK (Chua-Yang model)

\[
A = \begin{bmatrix}
0 & 1 & 0 \\
1 & 2 & 1 \\
0 & 1 & 0 \\
\end{bmatrix} \quad \quad \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad \quad \quad z = 4.5
\]

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: \( Y(t) \Rightarrow Y(T) = \) 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 \( T \to \infty \) all pixels will be driven to black.

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

III. ACE4K implementation

Implementation method:
patchmaker_ace4k: (Full-range model, ACE4K)

\[
A = \begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0 \\
\end{bmatrix} \quad \quad \quad B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \quad \quad \quad z = 4.5
\]

Remarks:
- Images should be fed into LLMs;

Example 1 (resolution: 64x64); template: patchmaker_ace4k.tem.
Example 2 (resolution: 200x300); template: `patchmaker_ace4k.tem`.

Running time: 630 µs.
**SmallObjectRemover:** Deletes small objects [22]

**Old names:** SMKILLER

**SmallObjectRemover (Chua-Yang model):**

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad
z = 0
\]

**I. Global Task**

Given: static binary image \( P \)

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

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

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

Output: \( Y(t) \Rightarrow Y(\infty) = \) 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.

**III. ACE4K implementation**

Implementation method: optimization.

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

\[
\begin{array}{ccc}
0.8 & 0.9 & 0.9 \\
0.8 & 1 & 0.9 \\
0.8 & 0.9 & 0.9 \\
\end{array}
\quad
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\quad
z_1 = -1.4 \quad z_2 = 0
\]
Example 1 (resolution: 64x64): image name: smkiller64_i.bmp, template name: smkiller_ace4k.tem.

Remarks:
- LAM must be used for the template execution.
1.4. TEXTURE SEGMENTATION AND DETECTION

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

Old names: TX_RACC3

\[
A = \begin{bmatrix}
0.86 & 0.94 & 3.75 \\
2.11 & -2.81 & 3.75 \\
-1.33 & -2.58 & -1.02
\end{bmatrix}, \quad
B = \begin{bmatrix}
0.16 & -1.56 & 1.25 \\
-2.89 & 1.09 & -3.2 \\
4.06 & 4.69 & 3.75
\end{bmatrix}, \quad
z = 1.8
\]

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: \( Y(t) \Rightarrow Y(T) = \) 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.

III. ACE4K implementation

Implementation method: recalculation of the template elements

Texture_ACE4K: (Full-range model, ACE4K)
Example: (resolution: 64x64): image name: text3x3.bmp, code name: text3x3.amc.

text3x3.tem: segmentation template

\[
\begin{bmatrix}
0.55 & 0.6 & 2.39 \\
-1.65 & -2.43 & -1.65 \\
2.39 & 2.39 & -0.65
\end{bmatrix}
\quad \begin{bmatrix}
0.1 & -0.99 & 0.79 \\
-1.84 & 0.69 & -2.04 \\
2.59 & 3 & 2.39
\end{bmatrix}
\]
\[ z_1 = \begin{bmatrix} 0 \end{bmatrix} \quad z_2 = \begin{bmatrix} 1.51 \end{bmatrix} \]

Using LAMs for the operation

Using LLMs for the operation.

Remarks:

- The original picture read back after the operation seems quite agreement to the original picture so no significant distortion of the gray scale image were assumed.
- The original template's values were down-sized to the range of the chip (-3,3).
- The classification work reliably on binary images only. The gray scale variant turns to black very soon in case of loop running.
Game of Life 1-Step: Simulates one step of the game of life [11]

Old names: LIFE_1

\[
\begin{align*}
A_{11} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & B_{11} &= \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & -1 \\ -1 & -1 & -1 \end{bmatrix}, & z = & -1 \\
A_{22} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & B_{21} &= \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, & z = & -4
\end{align*}
\]

I. Global Task

Given: static binary image \( P \)

Inputs:

\( U_1(t) = P, \ U_2(t) = P \)

Initial States:

\( X_1(0) = X_2(0) = -1 \)

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

Outputs:

\( Y_1(t), \ Y_2(t) \Rightarrow Y_1(\infty), Y_2(\infty) = \) Binary images representing partial results. The desired output is \( Y_1(\infty) \text{ XOR } Y_2(\infty) \). 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

III. ACE4K implementation

Implementation method: optimization
**LIFE_1_ACE4K: (Full-range model, ACE4K)**

\[ A_{11} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_{11} = \begin{bmatrix} -2 & -2 & -2 \\ -2 & 0 & -2 \\ -2 & -2 & -2 \end{bmatrix}, \quad z = 1.4 \]

\[ A_{22} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_{21} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad z = -3 \]

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

![Game of life](input) ![Game of life](output)

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

![Game of life](input) ![Game of life](output)

**Remarks:**

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

**EDGE CONTROLLED DIFFUSION**

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

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

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

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 (\( \Delta \)) is done by the HeatDiffusion (diffusion) template with different diffusion coefficients. The ThresholdedGradient (gradient) template can also be found in this library. This equation can be used for noise filtering without decreasing the sharpness of edges.

*The flow-chart of the algorithm:*
Example of the original algorithm: image name: laplace.bmp; image size: 100x100.

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

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

Diffusion:

\[
A = \begin{bmatrix}
0.35 & 0.35 & 0.35 \\
0.35 & -2.8 & 0.35 \\
0.35 & 0.35 & 0.35
\end{bmatrix}
\quad B = \begin{bmatrix}
0.2 & 0.2 & 0.2 \\
0.2 & 0.1 & 0.2 \\
0.2 & 0.2 & 0.2
\end{bmatrix}
\quad z = 1.2
\]

Edge1:

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad B = \begin{bmatrix}
0.7 & 0.7 & 0 \\
0.7 & 0 & -0.7 \\
0 & -0.7 & -0.7
\end{bmatrix}
\quad z = -1.5
\]

Edge2:

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\quad B = \begin{bmatrix}
-0.7 & -0.7 & 0 \\
-0.7 & 0 & 0.7 \\
0 & 0.7 & 0.7
\end{bmatrix}
\quad z = -1.5
\]

Execution time:

Diffusion: 1
Edge: 50

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


INDEX

3
3x3Halftoning, 30
3x3TextureSegmentation, 56

C
CenterPointDetector, 6
ConcaveLocationFiller, 44
ContourExtraction, 10
CornerDetection, 13

D
DiagonalHoleDetection, 4
DiagonalLineDetector, 18

E
EDGE CONTROLLED DIFFUSION, 60
EdgeDetection, 20

G
GameofLife1Step, 58
GradientIntensityEstimation, 1
GrayscaleLineDetector, 46

H
Hole-Filling, 32

L
LocalSouthernElementDetector, 36
LogicANDOperation, 48
LogicOROperation, 50

O
ObjectIncreasing, 34
OptimalEdgeDetector, 22

P
PatchMaker, 52
PointExtraction, 24
PointRemoval, 26

R
RightEdgeDetection, 38

S
SelectedObjectsExtraction, 28
ShadowProjection, 40
SmallObjectRemover, 54

V
VerticalLineRemover, 15
VerticalShadow, 42