AN ACTIVE CONTOUR ALGORITHM FOR CONTINUOUS-TIME CELLULAR NEURAL NETWORKS

by

D. L. Vilarino, T. Kozek and L. O. Chua

Memorandum No. UCB/ERL M98/67

10 December 1998

COVER

AN ACTIVE CONTOUR ALGORITHM FOR CONTINUOUS-TIME CELLULAR NEURAL NETWORKS

by

D. L. Vilarino, T. Kozek and L. O. Chua

Memorandum No. UCB/ERL M98/67

10 December 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

An Active Contour Algorithm for Continuous-Time Cellular Neural Networks

D.L. Vilariño*, T. Kozek and L.O. Chua
Electronics Research Laboratory
Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA-94720, USA

Abstract

In this work a CNN-based algorithm for image segmentation by active contours is introduced. It is based on an iterative process of expansion of the contour and its subsequent thinning guided by external and internal energy. This strategy leads to a high level of control over the contour evolution which makes their topologic transformations easier. Therefore processing of multiple contours for segmenting several objects can be carried out simultaneously.

1 Introduction

Image segmentation techniques by means of deformable models, or more specifically, active contours (so-called *snakes*) represent an interesting approach among segmentation strategies. They are introduced by Kass et al.[4], and usually consist of elastic curves that, located over an image, evolve from their initial shapes and positions in order to adapt themselves to the notable characteristics of the scene. This evolution comes as a result of the combined action of external and internal forces. The external forces lead the snakes towards features of the image, whereas internal forces model the elasticity of the curves. In a parametric representation, a snake appears as a curve $u(s) = (x(s), y(s)), s \in [0, 1]$, with $u(0) = u(1)$. Its internal energy is often defined as

$$E_{int}(u(s)) = \alpha|u_s(s)|^2 + \beta|u_{ss}(s)|^2. \quad (1)$$

It is made up of two factors: the membrane energy $\alpha|u_s(s)|^2$, which weights its resistance to stretching, and the thin-plate energy $\beta|u_{ss}(s)|^2$, that weights its resistance to bending.

*On leave from Department of Electronics and Computer Science. University of Santiago de Compostela. E-15706. Spain.

$u_s(s)$ and $u_{ss}(s)$ represent the first and second derivatives respectively. The elasticity parameters α and β control the smoothness of the curve.
The external energy is generally defined as a potential field P,

$$E_{ext}(u) = \int_0^1 P(u(s))ds. \tag{2}$$

This external potencial is a combination of different terms based on the application and the characteristic of interest. Typical terms in the expression of the external energy are [1, 5]:

$$P(u(s)) = \gamma|\nabla(G_\sigma(u(s)) * I(u(s)))| + \zeta I(u(s)) + \eta e^{-d(u(s))^2} \tag{3}$$

The first term in (3) corresponds to the gradient of the image convoluted with a Gaussian in order to reduce noise; the second to the intensity and the third to the distance to the closest borders obtained for the zero crossing. Finally γ, ζ, η are positive constants that weight the contributions of the different terms in the external energy.
The total energy of the snake will be the sum of the external and internal energy terms along the curve $u(s)$:

$$E_{snake} = \int [E_{int}(u(s)) + E_{ext}(u(s))]ds \tag{4}$$

The solution to the problem of detecting the contour is found in the minimization of this energy function. In order to numerically compute a minimal energy solution it is necessary to discretize the expression of the energy. Different procedures to approach both the discretization and the minimization of (4) can be used. However all of them require, to a greater or lesser degree, a high computational effort, which renders them inappropiate for applications needing fast response. On the other hand, due to their parametric nature, they usually cannot split a contour or merge to of them into one. This limits their application to segmentation tasks where the number of interesting objects and their approximate locations are known *a priori*.

The development of strategies based on active contours by means of cellular neural networks (CNN) could become an alternative to classical active contour techniques. The main motivation in pursuing this is the possible implementation as specific integrated circuits or taking advantage of architectures as the CNN Universal Machine [6] which would allow the use of massively parallel processing to reduce processing time.
The reduction in computational cost would justify this kind of technique. However it is not the only reason for investigating such an approach to active-contour image segmentation. In fact, the CNN-based solution provides a higher control of the evolution dynamics of the snakes and thus allows us to approach complex tasks for classical techniques as simple topologic tranformations.

2 CNN Architecture applied to Active Contours

In [7], a CNN-based strategy for the segmentation of an image on the basis of active contours has been proposed. The proposed method consists of an iterative process of

expansion of an initial contour, and its subsequent thinning, guided by external information which will indicate the direction of the displacement of the contour. Also, it obeys the influence of internal energy terms which try to maintain the connectivity of the contour (implicit to the thinning state) as well as to smooth the contour. The design has been realized using DTCNNs with cyclic time-variable cloning templates ([3] [2]) in such a way that,. differently from classical strategies, all of the contour points have an influence on the way the contour evolves. Therefore it could be considered as a continous treatment of the contour, given that its discretization is of the same order as the spatial variable in the images to be treated (pixel-level discretization). Therefore greater precision in the adaptation of the contour without additional computational cost is possible due to the greater freedom of movement of each point in the active contour.
However, because of the need for maintaining the connectivity of the contours, breaking of the active contours is not allowed. Therefore, as in classical active contour techniques, it is constrained to applications with a previous knowledge of the approximate location of the objects into the scene under processing.
Here we consider an CTCNN-based modification of that algorithm in [7] in order to allow the topologic transformation of the snakes. To this end, new processing steps are introduced to carry out the splitting and merging of the contours whenever they are needed to reach a valid final segmentation. In Figure 1, a diagram with the main processing steps is shown.

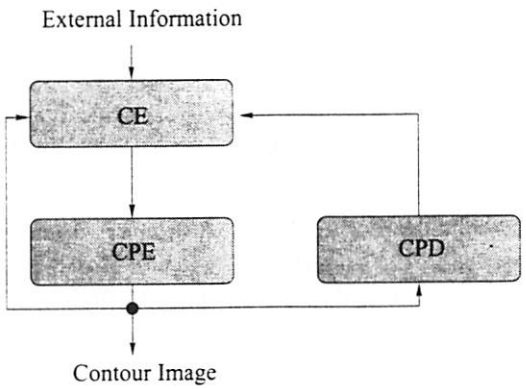


Figure 1: Block diagram of the complete algorithm.

The contour consists of a set of black pixels into a binary image, the so-called *contour image*. External information is represented by an array with real values called the *external information image* whose size coincides with that of the image under processing.

The algorithm consists of an iterative process in such a way that a complete cycle is carried out after the processing of the steps in Figure 1 along the four cardinal directions. Following a description of each elementary block is given.

2.1 *CE* module: Contour Evolution

The moving of the active contours is accomplished in the *CE* module of Figure 1. In Figure 2 a block diagram of the processing steps within the *CE* module are shown.

The block called *EXP* acts on the contour image (binary image) and the *EP* block on the

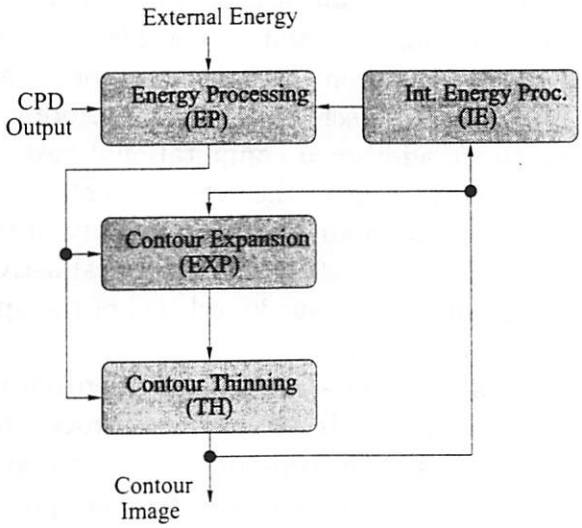


Figure 2: Block diagram for the processing in the *CE* module.

images of external and internal energy (The *CPD* output influence will be discussed later in this paper. For the moment, it will not be taken into account). Finally, the *TH* block combines both binary images from the output of *EXP* and *EP*, in order to carry out the correct evolution of the snake. So, unlike the classical proposals in deformable models, a contour reaches a final location after a dynamic process of activation and deactivation of pixels in a contour image. Following is a more detailed description of each block.

2.1.1 *EP* block: Information processing

In this block a weighted sum of gradients is calculated based on external (external output) and internal information (from the contour image) and subsequently, the result is thresholded. Therefore the *EP* output will be a binary image whose deactivated pixels indicate valid locations to move the contour in the *EXP* and *TH* stages. Figure 3 shows the block diagram associated with the operations in *EP*.

In the Figure, λ represents a constant (which may be changed over time) that weights the influence of each kind of energy. A default template for the directional derivative can be (for the processing in the Notherly direction):

$$A_{EP} = 0 \quad B_{EP} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad I_{EP} = 0 \tag{5}$$

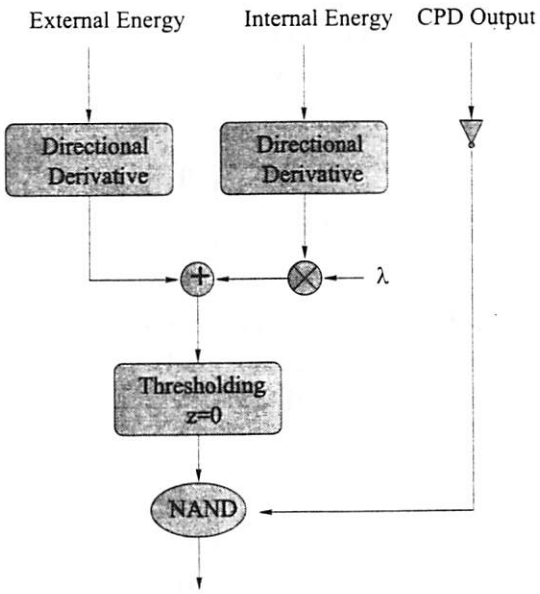


Figure 3: Block diagram of the *EP* processing block.

However, a more complex template can be used depending on the quality of the guide information.

2.1.2 *EXP* block: Contour expansion stage

This block makes a contour expansion process for the direction under consideration. The expansion will be allowed for one pixel only in order to make the subsequent thinning process easier. Also, only one duplication is allowed if the direction of processing coincides with the correct one for the movement of the contour based on the guide information. Figure 4 shows the blocks diagram associated to the *EXP* block processing.

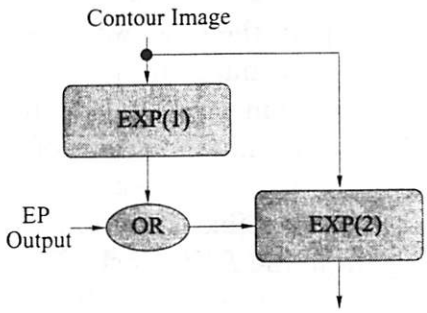


Figure 4: Block diagram of the *EXP* processing block.

In *EXP(1)* all activated pixels in the contour image are two-pixel shifted along the direction of processing. This is achieved by operating twice with the template (for the processing in North direction):

$$A_{EXP(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{EXP(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad I_{EXP(1)} = 0 \tag{6}$$

where the contour image is introduced as both initial state and external input. Following, a logical OR operation is carried out on the output from *EXP(1)* and *EP* blocks. The OR output is used as a fixed state map for the *EXP(2)* operation with template (for the processing in North direction):

$$A_{EXP(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{EXP(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad I_{EXP(2)} = 1 \tag{7}$$

where the contour image is introduced as both initial state and external input. Therefore in *EXP(2)* only a contour duplication along the correct direction (according with the guide information) is allowed.
In order to obtain the templates for the other directions $B_{EXP(1)}$ and $B_{EXP(2)}$ are suitably rotated.

2.1.3 *TH* block: Thinning stage

Through the *TH* block shown in Figure 2, a guided thinning process is carried out on the previously expanded contour in the *EXP* stage. The objective is to obtain a new thin contour slightly shifted and/or deformed based on the guide information from the *EP* block.
Due to the fact that thinning is made by parallel processing, a higher control is needed in order to avoid breaking of the contour. In Figure 5 a block diagram associated with the *TH* block is shown. This illustrates with a simple example of the thinning process on a previously expanded contour. To simplify the case, we have not considered any influence from the *EP* block (i.e all pixels of output image from *EP* are deactivated).

The operation of the thinning algorithm proceeds as follows: The output of the *EXP* block is introduced as external input and initial contour of the block *TH(1)* in Figure 5. Also the output from the *EP* block is used to mask those pixels that must not change their state based on the guide information (fixed state map). The *TH(1)* output will be almost a copy of the contour image from the *EXP* block. The difference is the deactivation of the contour borders in the direction of processing whose locations coincide with white pixels in the *EP* output. Therefore we will have a thinned contour image but with possible ruptures. Steps *TH(2)* and *TH(3)* act on areas with opposite slope in order to activate those pixels deactivated in *TH(1)* but needed to restore continuity of the contour. These two steps use the *EXP* as both external and initial state. Also, each block uses the output

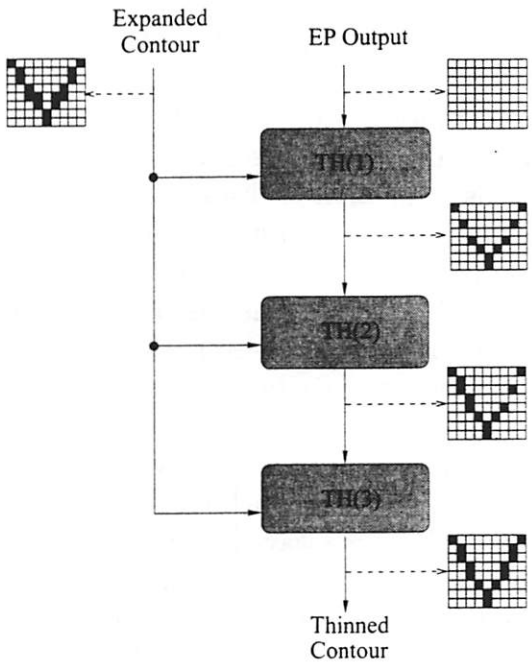


Figure 5: Block diagram for the *TH* processing block.

of the previous one as a fixed state map in such a way that only white pixels in the previous step can change their states. Templates for these operations for processing to the North are as follow:

$$A_{TH}(1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{TH}(1) = \begin{bmatrix} 0 & -2.5 & 0 \\ 0 & 5.0 & 0 \\ 0 & 2.5 & 0 \end{bmatrix} \quad I_{TH}(1) = -2.5 \tag{8}$$

$$A_{TH}(2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{TH}(2) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad I_{TH}(2) = -2 \tag{9}$$

$$A_{TH}(3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{TH}(3) = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I_{TH}(3) = -2 \tag{10}$$

The outlined algorithm operates as an eight-connectivity thinning process when a white image is used as a mask for *TH(1)*. However, in some situations eight-connectivity might not be reached at certain locations because of the influence of the guide information from the *EP* block.

2.1.4 *IE* block: Internal energy processing

Sometimes the external energy information is enough to guide the contour [8]. However, in those cases where the external information is insufficient or is corrupted by noise,

anchored pixels may appear in the contour, preventing it to reach its correct final location. The function of internal energy will be to smooth the contour shape in order to avoid rough deformation, so that a better convergence to the final contour is achieved.
In classical approaches, the internal energy depends on the tension of the contour (Equation (1)) and it may be measured as a function of distances among adjacent points according to the used discretization. This approach can not be directly applied to our case because the contour is not defined as a predetermined number of discretization points but the evolution of a set of black pixels of a binary image is based on the activation and deactivation of pixels belonging to this contour image. However, the desired smoothing effect can be obtained by assigning a higher amount of energy to those pixels in the contour image that are situated in concavities, with respect to those situated outside. Thus, it is possible to locally alter the influence of external energy, and so, invert the state of interesting pixels in the output image of the *EP* block. This approach recalls the thin-plate energy method in classical deformable-model strategies. One way to carry this out is by means of a diffusion operation in the *IE* block that has a binary image input (contour image from *TH*) and continuous output. Therefore the *IE* output will be proportional to the number of active pixels in the neighbourhood considered [7]. A template for this operation would be:

$$A_{IE} = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 1 & -5 & 1 \\ 0.5 & 1 & 0.5 \end{bmatrix} \quad B_{IE} = 0 \quad I_{IE} = 0 \tag{11}$$

The internal energy, appropiately weigthed in *EP* by the parameter λ, leads to smoothing of the contour shape. This influence will be higher at locations where the contour has a rough shape and will decrease in smooth areas.

In order to show the operation of the *CE* module, in Figure 6 an example of evolution for one complete cycle of the *CE* module is shown. This Figure shows the initial contour and the external information images, as well as the output of blocks *EXP*, *EP* and *TH* for each direction and iteration. Each row includes the results of all the processing steps for a given direction. The columns represent the output from all blocks for a particular processing step. The time sequence goes from left to right and from top to bottom. The guide information is constantly decreased in North-East direction and the internal energy influence is considered negligible ($\lambda \simeq 0$). Therefore the contour is moved in the direction of decreasing external energy. Highlighted images (i.e. output of *TH(3)* for the West direction of processing) are the external output after the complete cycle.

Sometimes, the number of active contours that are being processed do not coincide with the number of objects in the scene. These situations lead to collision between different contours (or different parts of the same contour). We will deal with such situations by separating or joining different contours. The procedure consists of two actions:

1. The possible collision between contours must be detected and avoided. This operation will be carried out by the *CPD* module of Figure 1. The locations of the contour image where collision can occur are called *collision points* (CP).

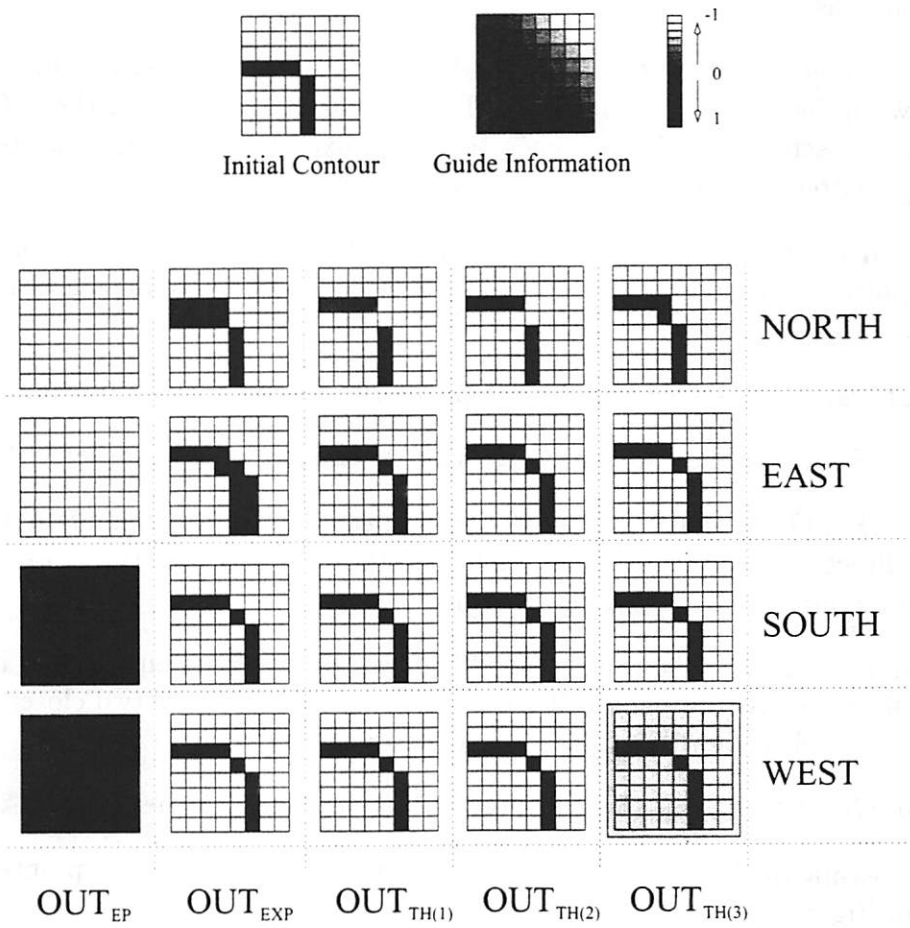


Figure 6: Example of the evolution of the contour in the North-East direction through one complete cycle of the *CE* module.

2. Based on the information about CP, topological transformations must be used to allow the splitting and merging of contours. This operation is carried out in the *CPE* module.

2.2 *CPD* module: Collision Point Detection

In this module a search for collision points is carried out. We can classify the behaviour of collision points as follows:

1. *Passive collision points (PCP)*: Pixels whose activation can lead to connections between two different contour pieces in the next processing step in the *CE* module. They are detected in *CPD* to generate a one-pixel wide *wall* that keeps the two colliding contour sections apart. These CP do not generate breakpoints.

2. *Active collision points (ACP)*: They are those CP that resolve one collision by means of the spliting and merging of the collided contours. They are detected in the *CPE* module.

Also the CP can be classified based on their distribution in the contour image:

1. *Isolated CP (ICP)*: They are those CP without connection with other CP.

2. *Vertical CP (VCP)*: They are those CP connected to at least another CP in the vertical direction. Also we will call *VCP-A* to those VCP whose two closer neighbors in the horizontal direction belong to some contour.

3. *Horizontal CP (HCP)*: They are those CP connected to a least another CP in the horizontal direction. Also we will call *HCP-A* to those HCP whose two closer neighbors in the vertical direction belong to some contour.

4. *Diagonal CP (DCP)*: They are all those CP not included in the previous cases.

Taking processing in the North direction as example, possible collision points would be those shown in Figure 7.

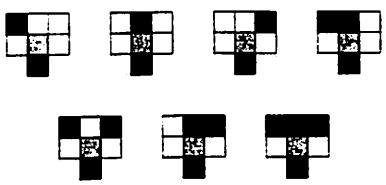


Figure 7: Examples of possible CP in the case of processing in the North direction.

In the Figure they are representated as:

Black cells: Activated pixels in the contour image before processing in the North direction.

White cells: Deactivated pixels in the contour image before processing in the North direction.

Grey cells: Collision points (also deactivated pixels before the processing in the North direction).

Dotted line cells: Pixels with any state (activated or deactivated).

All the CP of the Figure 7 are detected by a single CNN operation based on the template:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ -2 & -2 & -2 \\ 0 & 2 & 0 \end{bmatrix} \quad I = -5 \tag{12}$$

The examples of CP in Figure 7 are the more usual ones. However other CP that are not considered in that figure could appear. Figure 8 shows examples of this kind of CP.

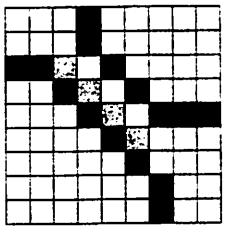


Figure 8: Example of possible CP not detected by the operation with template (12) in the processing in the North direction.

In order to take into account all the possible CP, multiple CNN operation are employed. Figure 9 shows a diagram of blocks for the *CPD* operation.

The operation *CPD(1)* is characterized by template (12). Then the principal and secondary diagonals are detected by the operation in *CPD(2)* and *CPD(3)*. All of the *CPD* blocks take as initial state and external input the contour image (from *CPE* output). Also the output of each operation is used as a fixed state map for the next operation in order to avoid the deactivation of previously detected CP. Templates for the operation in *CPD(2)* and *CPD(3)* are as follow:

$$A_{CPD(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{CPD(2)} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad I_{CPD(2)} = -5 \tag{13}$$

$$A_{CPD(3)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{CPD(3)} = \begin{bmatrix} 0 & -1 & 1 \\ 1 & -1 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad I_{CPD(3)} = -5 \tag{14}$$

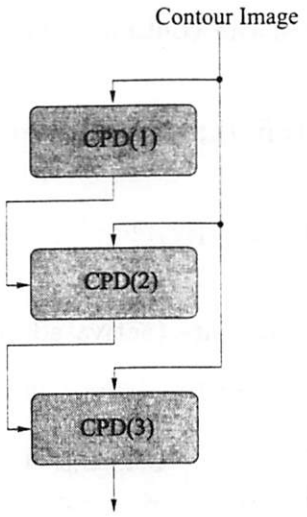


Figure 9: Block diagram for the *CPD* operation.

2.3 *CPE* module: Collision resolution

As we have already commented, the operation in the *CPD* module generates a one-pixel wide wall between two contours pieces that otherwise could collide. Given that situation, it is needed to decide which CP can be deactivated and what effect must cause in order to maintain the connectivity of the new contours. These CP are called active collision points (*ACP*). Following, the strategy to separate contours will be described. In order to make its understanding easier, a processing in a vertical direction will be considered (i.e. North or South processing):
A *HCP-A* will always have its North and South neighbors activated in the contour image. The first step to generate the rupture will be the deactivation of these pixels. Also lateral neighboring pixels must be activated in order to close the new contours. However it is not guaranteed that the new contours will be well defined. To achieve that it is required that the lateral neighboring pixels of the CP under consideration are *HCP-A* as well.
All these operations are carried out in the *CPE* module. The block diagram of its operation is shown in Figure 10.

CPE(1): Here it is detected if a pixel and its two lateral neighbors are *HCP-A*. The output of this block consists of a binary image whose black pixels are *ACP*. A template for this operation is as follows:

$$A_{CPE(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{CPE(1)} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \quad I_{CPE(1)} = -8 \qquad (15)$$

It uses the contour image as both initial state and external input.
CPE(2): Here the North and South neighbors of a *ACP* are deactivated (breakpoints are

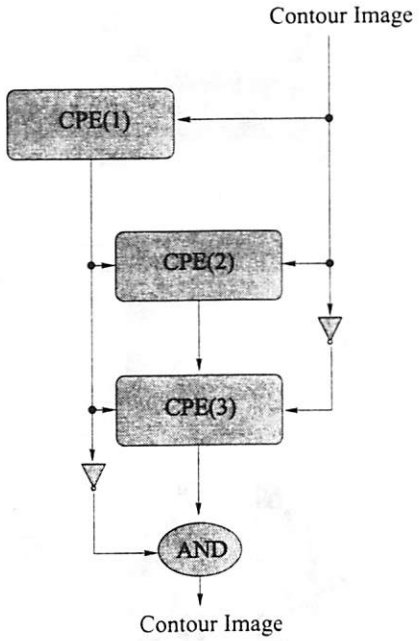


Figure 10: Block diagrams of processing steps in the *CPE* module.

generated). A template for this operation is:

$$A_{CPE(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{CPE(2)} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad I_{CPE(2)} = -1 \tag{16}$$

For this block, the *CPE(1)* output is introduced as external input and the contour image as both initial state and fixed state map. So the template only acts on the black pixels in the contour image.

CPE(3): Here the lateral neighbors to each *ACP* are activated (the connectivity of the new contours is guaranteed). A template for this operation is:

$$A_{CPE(3)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_{CPE(3)} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad I_{CPE(3)} = 1 \tag{17}$$

The output of the *CPE(2)* block is introduced as external input and the contour image as initial state. The complementary image of the contour image is used as fixed state map. Therefore the template only acts on the white pixels in the contour image. Finally it is needed to deactivate all those *ACP* that were activated during the *CPE(3)* operation (the propagation of the rupture is allowed). This is carried out by means the logical AND operation as shown in Figure 10.

In order to illustrate the operation of the *CPE* module, Figure 11 shows a single example

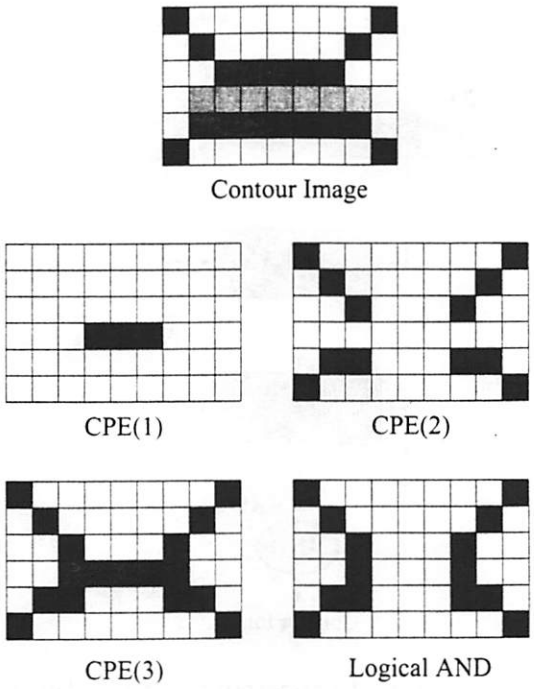


Figure 11: Example of the operation in the module *CPE*.

of separation of two contours (North processing).

The first picture in Figure 11 corresponds to the contour images (the grey pixels represent collision points).

In order to resolve the collision in the vertical direction (processing West or East) the previously proposed templates must be rotated 90 degrees.

Since one contour pixel can generate either three *HCP* or three *VCP*, the consideration of *DCP* would not be needed in almost any case (note that only one *ACP* is enough to resolve a collision).

Sometimes, separated *ACP* can appear between the same contours. That situation leads to the appearance of residual contours which will collapse in a pixel because of the internal energy influence. The elimination of these residual pixels may be important because they can obstruct the evolution of well-defined contours. Their elimination is straighforward by using a single CNN operation based on the template:

$$A_{RCE} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad I_{RCE} = -2 \tag{18}$$

which deactivates those pixels with less than two neighboring pixels belonging to any contour.

3 Examples

In order to show the validity of the proposed structure, we will show some simulations results. In all of the following cases, the external energy images were generated in such a way that the grey-level of each pixel is a function of the distance to the closest edge. The energy is assumed to be inversely proportional to the intensity. As a result the contours will move along decreasing energy directions.
The first example shows the consecutive adaptation of an active contour to the final contour based on different guide information images. Figure 12 shows the initial contour and the external information images.

Firstly, the image in Figure 12(a) was used as contour image and the image in Figure

(a) (b) (c)

Figure 12: Set of artificially generated test images. (a): Initial contour image, (b) and (c): External information images.

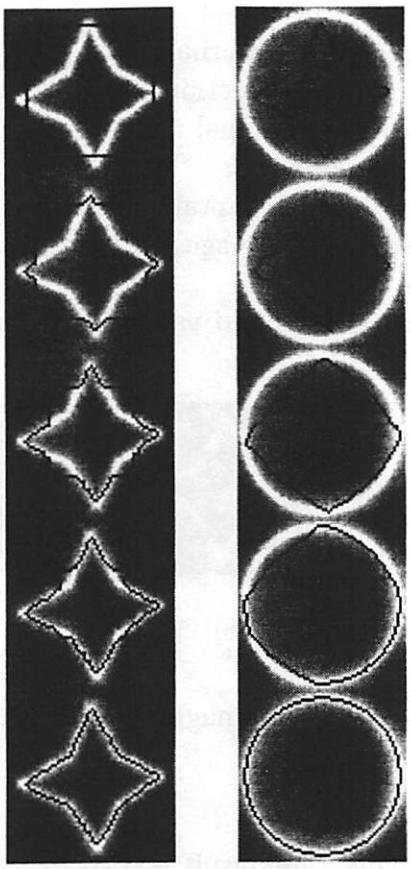
12(b) as external information image. The result is used as the contour image for the simulation guided by the image in Figure 12(c). Figure 13 shows the evolution of the active contour superimposed on the external information images.

As we have already mentioned, sometimes influence of the internal energy is not needed because of the good quality of the external information. However, in almost all practical cases, the presence of noise impedes the generation of suitable guide information images and therefore a final contour may not be reached. The internal energy leads to smoothing of the contour shape as it is illustrated in the sequence in Figure 14 (obviously, the lack of external energy leads to collapse of the contour).

In order to illustrate the influence of the internal energy we have carried out simulations from the set in Figure 15 which corresponds with a highly irregular initial contour and a noisy external information image.

In Figure 16 it can be seen the result of simulations from the image set in Figure 15 with and without internal energy influence respectively.

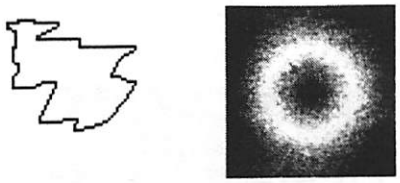
As we can see in Figure 16(a) the active contour can not continue its evolution towards the desired final contour because of the appearance of *anchored pixels* which have remained stuck due to the low quality of the guide information image. In these cases where the external information is insufficient, internal energy plays an important role. In fact, the smoothing effect permits the anchored pixels to escape from their locations and continue



(a) (b)

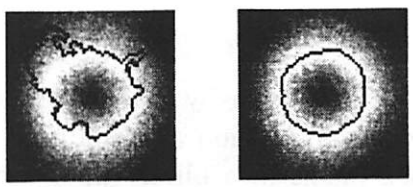
Figure 13: Evolution of the initial contour shown in Figure 12(a) guided by the external information images in Figure 12(b) and 12(c).

Figure 14: Example of a contour evolution based on internal energy influence.



(a) (b)

Figure 15: Set of artificially generated test images. (a): Initial contour image. (b): External information image corrupted with Gaussian noise ($\sigma = 100$).



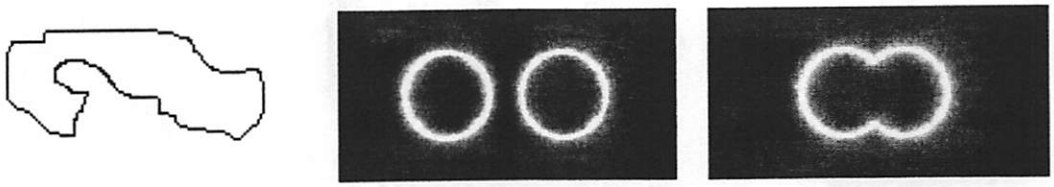
(a) (b)

Figure 16: Results of guiding the initial contour in Figure 15(a) with the noisy external information image in Figure 15(b). (a): Simulation without internal energy influence. (b): Simulation with internal energy influence.

towards the final contour (Figure 16(b)).

Finally, in order two show the behaviour of the proposed algorithm in situations that requiere any topological transformations, we have carry out simulations using the image set in Figure 17.

In Figure 17 the processing is initiated with the contour image 17(a) and guided by the



(a) (b) (c)

Figure 17: Set of artificially generated test images. (a): Initial contour image, (b) and (c): External information images.

external information in 17(b). After reaching convergence, the obtained contour image is used to be guided for the external information in Figure 17(c). Therefore that simulation will require the splitting and merging of different snakes.

In Figure 18(b) we can see that a residual contour has been generated. These residual contours usually appear far from any object of interest. Therefore the internal energy is strong enough to collapse them.

4 Conclusions

In this work, an approach to segmentation by active contours using CNN is discussed. It consists of an iterative process of expansion and thinning of the active contours which are represented as a set of black pixels in a binary image, called a contour image. The contours are shifted pixel by pixel to adapt their shape to those of the interesting object in the scene. This movement is guided by external forces that lead the contours towards salient features on the image under consideration. Also it obeys the influence of internal energy terms which lead to a smoothing effect on the contour shape. Furthermore the problem of the topological transformations of the snakes, needed for the processing of multiple active contours is approached. The strategy consists on the detection of possible collision between different contours pieces. This allows the algorithm to avoid the collision and carries out either splitting or merging of the contours when the continuity of the new contours are guaranteed.

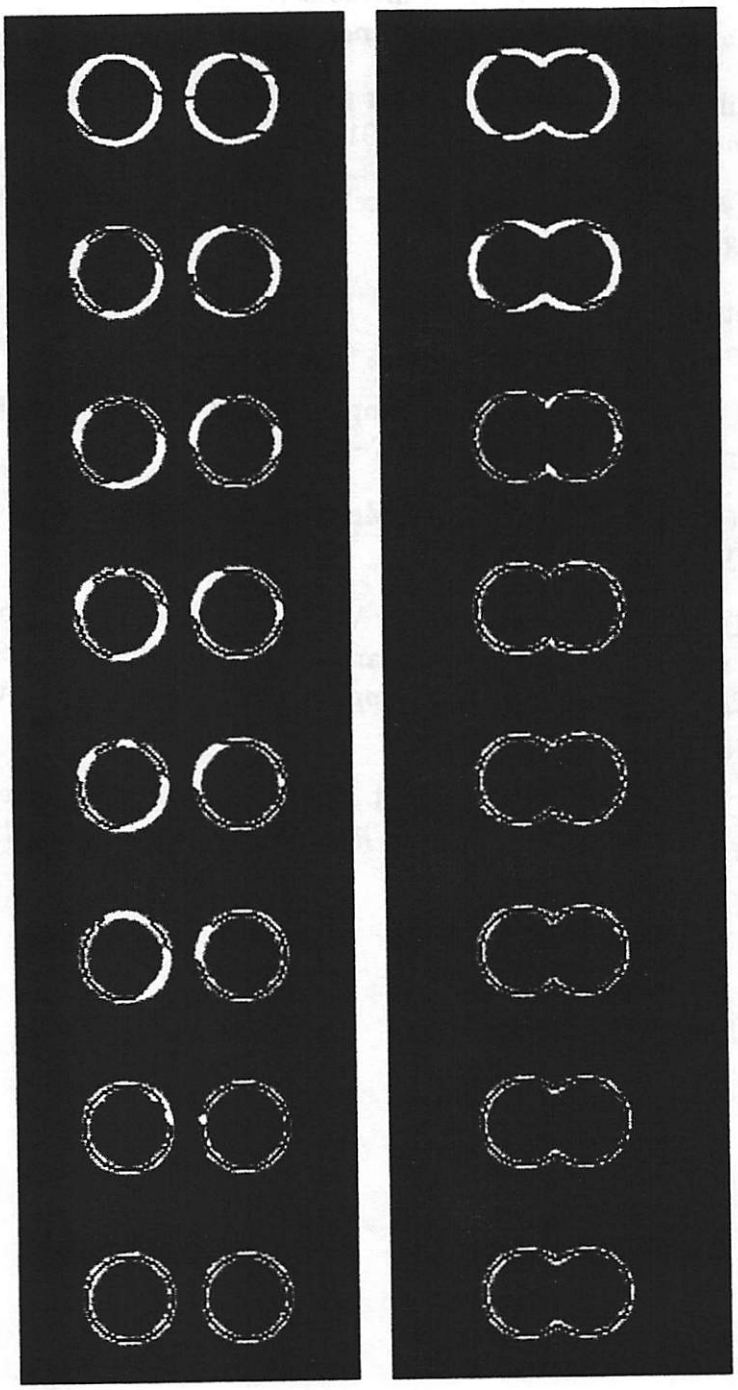


Figure 18: Evolution of the contour in Figure 17(a) guided by different external energy images.

References

[1] L.D. Cohen and I. Cohen. Finite element methods for active contour models and ballons for 2d and 3d images. *IEEE Trans. Patt. Anal. Machine Intell.*, (15):1131–1147, 1993.

[2] H. Harrer. Multilayer discrete-time cellular neural networks using time-variable templates. *IEEE Trans. Circuits Syst.*, 40(3):191–199, 1993.

[3] H. Harrer and J.A. Nossek. Discrete-time cellular neural networks. *Int. Journal Circ. Th. Appl.*, 20:453–467, 1992.

[4] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contours models. *Int. J. Computer Vision*, (1):321–331, 1988.

[5] P. Radeva and J. Serrat. Rubber snake: Implementation on signed distance potential. In *Vision Conference SWISS'93*, pages 187–194, 1993.

[6] T. Roska and L.O. Chua. CNN Universal Machine: An analogic array computer. *IEEE Trans. Circuits Syst.*, (3):163–173, 1993.

[7] D.L. Vilariño, D. Cabello, M. Balsi, and V.M Brea. Image segmentation based on active contours using discrete-time cellular neural networks. In Vedat Tavsanoglu. editor, *Fifth IEEE International Workshop on Cellular Neural Networks and Their Applications*, pages 331–336, 1998.

[8] D.L. Vilariño, D. Cabello V.M. Brea, and J.M. Pardo. Discrete-time CNN for image segmentation by active contours. *Pattern Recognition Letters*, 19(8):721–734, 1998.