Multiple contour extraction from graylevel images using an artificial neural network

Yedatore-Venkatakrishnaiya Venkatesh, S.Kumar Raja, Narasimha Ramya

Multiple Contour Extraction From Graylevel Images Using an Artificial Neural Network

Y. V. Venkatesh, *Senior Member, IEEE*, S. Kumar Raja, and N. Ramya

***Abstract*—For active contour modeling (ACM), we propose a *novel* self-organizing map (SOM)-based approach, called the batch-SOM (BSOM), that attempts to integrate the advantages of SOM- and snake-based ACMs in order to extract the desired contours from images. We employ feature points, *in the form of an edge-map (as obtained from a standard edge-detection operation)*, to guide the contour (as in the case of SOM-based ACMs) along with the gradient and intensity variations in a local region to ensure that the contour does not "leak" into the object boundary in case of faulty feature points (weak or broken edges). In contrast with the snake-based ACMs, *however*, we do not use an explicit energy functional (based on gradient or intensity) for controlling the contour movement. We extend the BSOM to handle extraction of contours of multiple objects, by splitting a single contour into as many subcontours as the objects in the image. The BSOM and its extended version are tested on synthetic binary and gray-level images with *both single and multiple objects*. We also demonstrate the efficacy of the BSOM on images of objects having both *convex* and *nonconvex* boundaries. The results demonstrate the superiority of the BSOM over others. Finally, we analyze the limitations of the BSOM.**

***Index Terms*—Active contour models (ACMs), edge detection, contour extraction, snakes, self-organizing map (SOM), time-adaptive self-organizing map (TASOM).**

I. Introduction

In COMPUTER VISION, extraction of boundaries (or contours) of objects from the images of a scene is needed for shape description, leading to object localization and recognition, but the conventional edge extraction techniques, being sensitive to (image) noise and intensity variations, often *do not* give us the true boundaries of objects in images. On the other hand, their outputs usually contain spurious or weak edges.[1] In order to overcome such problems, edge-linking techniques have been suggested [1]. It is now generally acknowledged that, without a higher-level information of the object itself, such techniques produce erroneous results. To be more specific, the boundary obtained from typical edge-linking techniques may be deformed or broken without some additional information (such as the geometry of the object). For lack of space and in view of the extensive literature on the topic, we refer here only to the most relevant ones.

Kass *et al.* [2] suggest a flexible framework called *active contour models* (ACM) or *snakes* with scope for using higher-level image information and treat the problem of contour extraction as one of energy minimization, with the energy functional consisting of terms corresponding to the internal, image and external forces. In this framework, an initial contour (snake), represented as a parametric curve (spline), deforms in order to minimize the energy functional with components as the internal energy of the spline, the image force and an external constraint force. By choosing the internal energy appropriately, Kass *et al.* [2] impose regularity on the shape of the curve. The image energy is chosen in such a way as to attract the snakes to the salient features in the image such as lines and edges, and the external energy term, defined by the user, generates forces that *push* the snake from one local minimum into another. The minimum of the energy functional is to be determined by solving Euler–Lagrange partial differential equations (in two variables), a solution to which requires a specification of the initial conditions in the form of a contour. However, it is found that such a contour must be close to the true boundary; otherwise, the initial contour may not move to the correct boundary of the object in low-contrast images (i.e., those with small gradient magnitudes). Further, if an object's boundary has concavities, we observe unsatisfactory convergence of the initial contour to it (inspite of being close). That is, it cannot invade boundary concavities, and may get stuck in a local minimum of an energy functional. In order to overcome this, Cohen [3] proposes a *balloon* model in which (as the name implies) the curve or surface is regarded as a balloon that is inflated by a pressure force. Xu and Prince [6] introduce an additional external force called *gradient vector flow* (GVF). For related material, see Leymarie and Levine [4]; Grzeszczuk and Levin [5]; Peterfreund [7] (who deals with the tracking of **nonrigid** objects not considered in the present paper) using the "velocity-snake"; Mclnerney and Terzopoulos [8] (who propose *topology-adaptive* (or T-) snake, a discrete approximation to the conventional snake); and Caselles *et al.* [9] and Malladi *et al.* [10] (for *geometric active contours*), in which the curve representing the contour evolves in such a way that the length of the curve, weighted by a function of gradient of the image along the curve, is minimized. The main drawback of geometric contours is *leakage* through object boundary in case of gaps or weak edges. To overcome it, Xie *et al.* [11] introduced *region-aided geometric snake* (RAGS) which integrates forces of gradient flow [6] with those of diffused regions.

Manuscript received November 9, 2004; revised March 23, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Christophe Molina.

Y. V. Venkatesh is with the Department of Electrical and Computer Engineering, Faculty of Engineering, National University of Singapore, Singapore 117576 and also with the Computer Vision Laboratory, Department of Electrical Engineering, Indian Institute of Science, Bangalore, 560012, India (e-mail: eleyvv@nus.edu.sg; yvvele@ee.iisc.ernet.in).

S. K. Raja and N. Ramya are with the Computer Vision Laboratory, Department of Electrical Engineering, Indian Institute of Science, Bangalore, 560012, India (e-mail: kumar@darbar.ee.iisc.ernet.in).

Digital Object Identifier 10.1109/TIP.2005.863934

[1]Weak edges are those which are perceived by the human eye as true edges but have *low* gradient magnitude values.

The most recent paper [12][2] deals with a charge-particle model (CPM) that derives inspiration from classical electrodynamics. In this model, the charges are attracted toward the contours of the objects of interest by an "electrostatic field" whose sources are computed based on the gradient-magnitude image. The forces of the snake model have their counterparts here, too. Initially, the charges can be placed inside, or outside or both inside and outside, too. According to the authors of [12], 1) the method is insensitive to initialization, 2) the capture range is increased, and 3) robustness against boundary leakage is also enhanced.

A. SOM-Based Network for Contour Extraction

An alternative approach to the (above-described parametric) ACMs is a two-layered *neural network*, based on a self-organizing map (SOM) [13]–[15]. In [13], the authors describe a class of constrained algorithms for object boundary extraction that includes deformable models.

In [14], the *classical* SOM is adapted (possibly for the first time) to the problem of extracting boundaries of objects in images, thereby dispensing with the explicit minimization of energy-based functionals. The network, with a fixed number of neurons in a chain topology, is created to be isomorphic to an initial contour (i.e., there is a one-to-one correspondence between the contour points and the neurons of the network), starting from which it evolves, by being subjected to *deformation* in order to map onto the nearest salient contour in the image. The training of the network uses a scheme similar to Kohonen's algorithm [16] (which implies an *unsupervised mode*), but is distinct from it in terms of architecture [see Fig. 1(a) and (b)]. The equation for weight-updation for the neurons is given by

$$\mathbf{w}^i(n+1) = \mathbf{w}^i(n) + \eta(n) * \exp\left(\frac{-\|\mathbf{w}^w(n) - \mathbf{w}^i(n)\|^2}{2\sigma(n)^2}\right) * (\mathbf{p} - \mathbf{w}^i(n)) \quad (1)$$

where $\mathbf{w}^i(n)$ is the weight of neuron i in nth iteration, η is the learning-rate parameter, the exponential term is the neighborhood-function with a parameter σ, and $\mathbf{p}$ is the input feature vector which is obtained from the edge map of the image. The parameters η and σ, which decrease with time, are common to all the neurons. At the end of the training, the network converges to the actual contour.

Such a SOM's primary **limitations** are: the initial contour must be 1) close to the object boundary and 2) *similar* to the shape of the object. These are due to the fact that the neighborhood function in weight-updation [see (1)] is solely based on the *physical distances between the neurons* and *not* the indices. Thus, weight-updation tapers off when the neurons are faraway from the winner, even though they are topological neighbors of it, and, since the number of points on the contour remains fixed, the final contour does not accurately represent the actual boundary. Further, if the initial contour is not specified by a continuous set of points, the algorithm fails to yield correct results.

The *time-adaptive self-organizing map* (TASOM) [15] is a modified form of the SOM-algorithm to overcome the above limitations. It employs, for each neuron, individual learning rates and neighborhood parameters which are updated on the basis of the environment conditions in each iteration and incorporates a mechanism to insert and delete neurons, thereby ensuring the continuity of the contour. However, since the movement of the contour is *solely* based on the edge map (feature points), the contour **leaks** through the boundary in the case of broken or weak edges, and the algorithm gives unsatisfactory results if the edges are thick or have very high concavities.[3] Its other weaknesses are: 1) it fails to converge when the η parameter becomes small; 2) its convergence is affected by noise; 3) it has many parameters which need to be changed for different images; and 4) it does not converge in the case of images of multiple objects (due to the perpetual insertion and deletion of neurons). Among these, it has been found that item 3 **cannot** be rectified in the framework of SOM and TASOM, but items 1 and 4 have been overcome by an appropriate modification of TASOM, called the MTASOM, in [17]. As far as item 2 is concerned, most of the SOM-based algorithms seem to be **ineffective** without some *pre-processing* (like Gaussian filtering[4]) of the original image.

This serves as a motivation for our *new*, radically different approach, combining the merits of both SOM- and snake-based ACMs, which overcomes **all** the above limitations. In the new algorithm, the *updation procedure of neuron weights is* unlike *Kohonen's* [16] and *is specific only to the problem of active contours*, but, in contrast with MTASOM, the new algorithm uses the **weights** of the neurons themselves as the **new positions** of the control points, and *hence* of the contour.

The rest of the paper is organized as follows. In Section II, we describe the new algorithm for contour extraction from images having only one object, and give the results of this algorithm in Section III. In Section IV, we extend it to extract contours of multiple objects from images, and illustrate it in Section V. Finally, we conclude the paper in Section VI. In view of space limitations, illustrations are reduced to a minimum. For more details and examples, see [18].

II. Proposed Algorithm

The **proposed algorithm** uses a neural architecture similar to that of the SOM-based ACMs. The feature vector it uses are the coordinate of edge points obtained from a standard edge-detection algorithm. In addition, the algorithm uses 1) *intensity variations* and 2) **gradient information** in a local region in order to guide the movement of the contour. There are two essential ideas used for controlling the neuronal weight updates. First, if a neuron is near a region boundary, it is forced to move along the normal to the contour or opposite to it, such that the gradient magnitude at the new location is greater than that at the current location. *The neuron is not allowed to move to the new position if there is a decrease in the gradient magnitude*, thereby ensuring proper convergence even if there are broken or weak edges. Note that **this approach does not use an explicit gradient energy term found in snake-based ACMs**. The second idea is that if

[2]This came to our attention after the complete draft of our paper had been prepared for submission.

[3]See illustrations in Section III.

[4]Note that, in snake-based ACMs, the image energy is computed from the *gradient* of a Gaussian blurred image which suppresses the effect of noise.

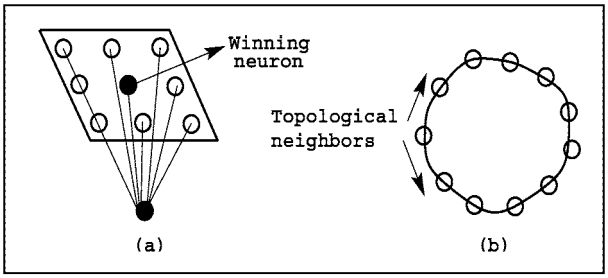


Fig. 1. (a) Lattice Topology employed in conventional **Kohonen-SOM** algorithms. (b) Chain Topology used in **SOM-based** ACMs.

the neuron is in a uniform region, it is moved closer to an appropriately chosen feature point (vector).[5] Further, in contrast with TASOM and MTASOM algorithms [15], [17], the proposed algorithm uses the **weights** of the neurons themselves as the **new positions** of the control points, and *hence* of the contour.

A. Description

The output of an edge-detection algorithm (as applied to the image) provides the feature points for training the neural network. Let $\mathfrak{X} = \{\mathbf{x_1}, \mathbf{x}_2, \ldots, \mathbf{x_m}\}$ be the set of the feature points, where $\mathbf{x}_k = (x_k, y_k)$ are the $x-$ and $y-$coordinates of the kth feature point. At epoch n, the contour to be deformed is modeled as a *sequence* of control points $\mathcal{P}(n) = \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{M(n)}$, where $\mathbf{p}_j = (x_j, y_j)$ is the position of the jth control point, and $M(n)$ is the number of control points in the nth epoch. In this algorithm, a set of neurons arranged in a chain-topology similar to that found in [14] and [15] is used [see Fig. 1(b)]. Let $\mathcal{W}(n) = \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{M(n)}$ be the sequence of neuronal weights, where $\mathbf{w}_i = (w_i^x, w_i^y)$ is the weight of i^{th} neuron. *Note that the number of nodes in the network and the number of the control points are equal, i.e., there is one-to-one correspondence between each of the control points and neurons in the network* **isomorphism**. The proposed algorithm involves the following steps: 1) initialization of the weights of the network to the initial position of the contour: $\mathcal{W}(0) \leftarrow \mathcal{P}(0)$; 2) initialization of the updates of the neuronal weights to zero; 3) finding the winning neuron for every feature point; 4) computation of updates for winner neurons and validation of the updates; 5) parallel updation of the weights of the network, and updation of the control points: $P(n) \leftarrow \mathcal{W}(n)$; 6) insertion and deletion of neurons to the network; 7) updation of the learning rate parameter η; and 8) repetition of the algorithm from Step 2 until the convergence criterion is satisfied.

In Step 1, a neural network *isomorphic* to the initial contour is constructed, and the weights of the neurons are initialized to the initial position of the contour. The updates for the neuronal weights are then set to 0 (Step 2). Description of Step 3 needs the following definitions of certain functions and sets. For every feature point, there exists a neuron which is physically nearest (in Euclidean distance) to it, and we term these winning neurons as *first-level winners* (FLW). The winner neuron associated with the feature vector $\mathbf{x}_k$ is denoted as

$$n(\mathbf{x}_k) = \arg \min_j d(\mathbf{x}_k, \mathbf{w}_j) \tag{2}$$

[5]We use "point" and "vector" interchangeably whenever there is no cause for doubt.

where $d(\mathbf{x}, \mathbf{w})$ is the Euclidean norm between the feature vector and the neuron weight. It is possible that many feature points find the same neuron as a first-level winner. For every FLW i, we associate a set of feature points $\mathbb{X}(i)$ which makes the neuron a winner according to (2). In other words, $\mathbb{X}(i) = \{\mathbf{x} | i = n(\mathbf{x}), \mathbf{x} \in \mathfrak{X}\}$. It is clear that $\mathbb{X}(i) \subset \mathfrak{X}$. Also, for each neuron i, there exists a feature point $\mathbf{g}(i)$ among all feature points $\mathfrak{X}$ which is nearest to it, i.e., $\mathbf{g}(i) = \arg \min_{\mathbf{x} \in \mathfrak{X}} d(\mathbf{x}, \mathbf{w}_i)$. *Note that* $\mathbf{g}(i)$ may not belong to $\mathbb{X}(i)$. For each FLW i, we can find a feature point from the set $\mathbb{X}(i)$ which is physically nearest to it. We denote this feature point as $\mathbf{f}(i)$, given by

$$\mathbf{f}(i) = \arg \min_{\mathbf{x} \in \mathbb{X}(i)} d(\mathbf{x}, \mathbf{w}_i). \tag{3}$$

It is obvious that $d(\mathbf{f}(i), \mathbf{w}_i) \geq d(\mathbf{g}(i), \mathbf{w}_i)$. After finding the FLWs for all the feature points in Step 3, we compute their (i.e., FLWs') updates in Step 4 as follows:

$$\Delta \mathbf{w}_i(t) = \eta(t) p(\mathbf{f}(i) - \mathbf{w}_i(t)) \tag{4}$$

where $\eta(t)$ is the learning rate parameter, and p is the damping factor given by

$$p = \frac{\|\mathbf{g}(i) - \mathbf{w}_i(t)\|}{\|\mathbf{f}(i) - \mathbf{w}_i(t)\|}. \tag{5}$$

The updation (4) ensures that the first-level neuron i moves not only toward that feature point which finds it as a winner, but also is closest to itself according to (3). *This rule of updation is distinct from other conventional SOM-based ACM implementations*. Note that $0 < p \leq 1$. The role of the damping factor is to ensure a smooth evolution of the contour toward the object boundary, without penetrating it. Before updating, we validate the (*currently*) computed update for every neuron. This step is essential in order to prevent the contour from penetrating weak or broken edges. The unit vector $\mathbf{e}$ tangential to the contour at the position of the neuron i is defined as

$$\mathbf{e} = (e_x, e_y) = \frac{\mathbf{w}_{i+1} - \mathbf{w}_{i-1}}{\|\mathbf{w}_{i+1} - \mathbf{w}_{i-1}\|}.$$

The unit vector perpendicular to the above vector (i.e., normal to the contour) is given by $\mathbf{e}^{\perp} = (-e_y, e_x)$. For $k \in \{1, 2, \ldots N_r\}$, consider the pair of points

$$\mathbf{p}_a = \mathbf{w}_i + k\mathbf{e}^{\perp}; \quad \text{and} \quad \mathbf{p}_b = \mathbf{w}_i - k\mathbf{e}^{\perp} \tag{6}$$

where N_r is the number of neighborhood points considered for determining a region boundary (see Fig. 2). Let "sgn" denote the signum function, and I, the image intensity function. We define the quantity

$$u(i) \triangleq \sum_{k=1}^{N_r} \text{sgn}(I(\mathbf{p}_a) - I(\mathbf{p}_b)). \tag{7}$$

If a neuron is near a region boundary, then the sign of the difference between the image intensities at the points $\mathbf{p}_a$ and $\mathbf{p}_b$ must be the same for all k (see Fig. 2). Therefore, for a neuron which is near a region boundary, u defined in (7) must be N_r or $-N_r$. In our work, we have chosen $N_r = 2$. Note that if $N_r = 1$, then $u(i)$ will always be trivially equal to 1 [see (7)].

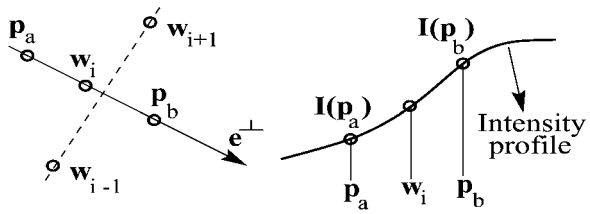


Fig. 2. Region boundary.

Hence, N_r should be ≥ 2. Larger values of N_r can be used for noisy images. Moreover, in order to make the algorithm robust with respect to noise or minor intensity variations, we consider a neuron to be in a uniform region if $|I(\mathbf{p}_a) - I(\mathbf{p}_b)| < G_{\max}$ for any k. In our experiments, we have explored values of $G_{\max}$ between 1 and 5.

In order to declare whether a neuron is in a homogenous or uniform region, we define the function $R(i)$ as follows: $R(i) = 1$ for $|u(i)| = N_r; R(i) = 0$ for $|I(\mathbf{p}_a) - I(\mathbf{p}_b)| < G_{\max}$ for any k, and $R(i) = 0$, otherwise. If $R(i) = 0$, then the neuron is considered to be in an uniform region, and its update is set according to (4). However, near a region boundary (i.e., for $R(i) = 1$), we consider two cases: (1) $||\Delta\mathbf{w}_i(t)|| > 1$ and (2) $||\Delta\mathbf{w}_i(t)|| \leq 1$. We define the quantities: $\mathbf{u}_i = (\Delta\mathbf{w}_i(t))/(||\Delta\mathbf{w}_i(t)||)$ and $h = |\mathbf{e}^{\perp}.\mathbf{u}_i|$.

Case 1: $(||\Delta\mathbf{w}_i(t)|| > 1)$ The neuron is forced to move along the direction $\mathbf{e}^{\perp}$ (i.e., normal to the contour) or opposite to it $(-\mathbf{e}^{\perp})$. For a neuron i, consider the quantities: $q_{-1} = ||\nabla I(\mathbf{w}_i - h\mathbf{e}^{\perp})||; q_0 = ||\nabla I(\mathbf{w}_i)||; q_1 = ||\nabla I(\mathbf{w}_i + h\mathbf{e}^{\perp})||$; and $q_u = ||\nabla I(\mathbf{w}_i + h\mathbf{u}_i)||$, where $||\nabla I(.)||$ denotes the gradient magnitude function; q_0 is the gradient magnitude at the location of the neuron; q_{-1} and q_1 are the gradient magnitudes at points left and right of the neuron (by convention), along the contour normal; and q_u is the gradient magnitude at a point along the update direction.

Our objective is to make all neurons move toward gradient peaks, and thereby toward the true boundary. To this end, we do not allow any neuron to move to a point where the gradient magnitude is smaller than at its current location.

The weight update for a FLW (i.e., first-level winner, as defined earlier) neuron is computed as follows.

If $(q_1 < q_0)$ & $(q_{-1} < q_0)$
  $\Delta\mathbf{w}_i(t) = 0$
  else if $(q_{-1} \leq q_0 < q_1)$
    $\Delta\mathbf{w}_i(t) = h\mathbf{e}^{\perp}$
    else if $(q_1 \leq q_0 < q_{-1})$
      $\Delta\mathbf{w}_i(t) = -h\mathbf{e}^{\perp}$
      else if $(q_0 > q_u)$
        $\Delta\mathbf{w}_i(t) = 0$else $\Delta\mathbf{w}_i(t) = h\mathbf{u}_i$.

The first condition does not allow the neuron to move from a local maxima of the gradient magnitude. The second, third, and fifth conditions ensure that the neuron moves a toward a gradient peak. The fourth condition prevents the neuron from moving from a higher gradient location to a lower one.

Case 2: $(||\Delta\mathbf{w}_i(t)|| \leq 1)$ The neuron is moved along the update direction provided there is an increase in the gradient magnitude. For the neuron i, consider the quantities

$$\begin{aligned} q_{-1} &= ||\nabla I(\mathbf{w}_i - \mathbf{e}^{\perp})|| \\ q_0 &= ||\nabla I(\mathbf{w}_i)|| \\ q_1 &= ||\nabla I(\mathbf{w}_i + \mathbf{e}^{\perp})|| \\ q_u &= ||\nabla I(\mathbf{w}_i + \Delta\mathbf{w}_i(t))||. \end{aligned} \tag{8}$$

The weight update for a FLW neuron is computed as follows.

If $(q_0 > q_u)$
  $\Delta\mathbf{w}_i(t) = 0$
  else if $(q_1 < q_0)$ & $(q_{-1} < q_0)$
    $\Delta\mathbf{w}_i(t) = 0$
      else $\Delta\mathbf{w}_i(t) = \eta(t)p(\mathbf{f}(i) - \mathbf{w}_i(t))$

Similar to Case 1, the first and second conditions do not allow a neuron to move either to a position of lower gradient magnitude or away from a gradient peak. The maximum update among all the FLW-updates [according to (4) and validation] is found, and if the maximum update is very small, then the contour does not move significantly. In this case, the second-level winners (SLW) are invoked. The SLWs are picked from a set of neurons which have not won in the competition in the previous iteration, i.e., these neurons are not FLWs, and *they are not located near region boundaries* $(R(i) = 0)$. The feature points which are used to find this set of winners are those whose distances from their corresponding FLW neurons are greater than $\sqrt{2}d_{\max}$. *This constraint is used for preventing feature points which are very close to FLW neurons to have a SLW*. Let the feature $\mathbf{x}_k$ satisfy the property, $\min_{j \in \mathfrak{X}} d(\mathbf{x}_k, \mathbf{w}_j) > \sqrt{2}d_{\max}$. The set of neurons *which are not FLWs*, and for which $R(i) = 0$ is denoted as $\mathfrak{N}_{\mathfrak{s}}$. Mathematically, a neuron $s \in \mathfrak{N}_{\mathfrak{s}}$ is a SLW of feature $\mathbf{x}_k$ if $d(\mathbf{x}_k, \mathbf{w}_s) \leq d(\mathbf{x}_k, \mathbf{w}_j), \forall j \in \mathfrak{N}_{\mathfrak{s}}$. These SLW neurons have an important role to play when dealing with **contours with high concavities**. The updates for the SLW neurons are computed and validated in the same way as for the FLW neurons [according to (4)] (in the following, we illustrate with an example the advantage of using SLWs). All the neuronal weights are updated *in parallel* (Step 5) according to the equation, $\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \Delta\mathbf{w}_i(t)$. *Note that the parallel updation in our proposed algorithm is distinct from sequential updation used in the previous SOM-based ACMs* [14], [15]. **This prompts us to call this algorithm "batch-SOM" (BSOM) algorithm.**

In Step 6, those neurons which are not modified or updated during the epoch are deleted. Since only the winning neurons (at the first and second levels) are updated, all the nonwinners are deleted. Furthermore, if the distance between two *topologically neighboring* neurons is less than $d_{\min}$, then one of the two neurons is deleted. However, during deletion, if one of the neurons is FLW and the other SLW, then the latter is deleted. In case of both being FLWs or SLWs, any one of them can be deleted. After deletion of nodes, the network contains neurons which are either FLW or SLW. If the distance between any two topologically neighboring neurons is greater than $d_{\max}$,

then a "dense" set of nodes is inserted between the two nodes. Let i and j denote two topologically neighboring neurons, and $d_{ij} = ||\mathbf{w}_i - \mathbf{w}_j||$, the distance between the two nodes. Define quantity N as $N = \lfloor (2d_{ij})/(d_{\max}) \rfloor - 1$, where $\lfloor \rfloor$ is the floor operation. If d_{ij} is greater than $d_{\max}$ and $N > 0$, then the number of new neurons to be inserted[6] between winners i and j is N. The weight of the kth neuron inserted is given by

$$\mathbf{w}_{\text{new}}^{(k)} = \mathbf{w}_i + \frac{k(\mathbf{w}_j - \mathbf{w}_i)}{N+1}, \qquad k = 1, 2, \ldots, N. \quad (9)$$

For consistency, $d_{\max} > 2d_{\min}$. In order to have an inter-nodal distance which is *not less* than 1 pixel in the final contour, we choose $d_{\min} = 1.0$, and $d_{\max} = 2.3$.

In Step 7, the learning rate parameter is updated in such a way that the value of η falls very slowly until a certain **predetermined** number of epochs (μ) is reached, and then falls rapidly toward the final value of η. The rate at which the value of η falls is controlled by the slope parameter λ. The updation for the learning rate parameter is done as follows:

$$\eta(t) = \eta_f + (\eta_i - \eta_f)\left(\frac{1 + \exp(-\mu/\lambda)}{1 + \exp((t-\mu)/\lambda)}\right) \quad (10)$$

where η_i and η_f are the initial and final values of the learning rate parameter. Note that $0 < \eta_i, \eta_f < 1$ and $\eta_i \gg \eta_f$. In our experiments, we have chosen $\eta_i = 0.1$ and $\eta_f = 0.001$. A rough sketch of η *versus* t for $\mu = 5000$ and $\lambda = 1$ is shown in Fig. 3.

In Step 8, the network is checked for convergence, and if the convergence condition is not satisfied, the algorithm is repeated from Step 2. In our algorithm, we use a convergence criterion similar to that of snake-based ACM, wherein we use the gradient information to determine whether the contour approximates the desired boundary. Note that the snake-based ACM uses gradient energy and internal energy (smoothness) terms to determine the desired contour (corresponding to a local minima of a energy functional). However, the new **BSOM** algorithm does not use any explicit energy term for determining the convergence.

The **BSOM** network is said to have converged if the following two criteria are satisfied. First, the *neurons which are FLW or SLW must be at a gradient peak or have their update magnitudes less than* ϵ (for instance, $\epsilon = 0.001$). To verify the gradient peak criterion, consider a neuron i (either FLW or SLW), and the quantities q_{-1}, q_0 and q_1 defined in (8). The neuron i is said to be at a gradient magnitude peak if $q_0 > q_{-1}$ and $q_0 > q_1$. In other words, this criterion states that the gradient magnitude at the location of the neuron must be greater than at the points left and right of the neuron, along the contour normal. The updating criterion merely states that the movement of the winner neurons must be very small. The second criterion is that the *nonwinner neurons must be near a region boundary*. In order to verify this criterion, consider a neuron i (nonwinner), and the points $\mathbf{p}_a$ and $\mathbf{p}_b$ [defined in (6)] which are the neighbors along the contour-normal at the neuron location. Also, for $k \in \{1, 2 \ldots N_r\}$, consider another pair of points along a direction tangential to the contour: $\mathbf{p}_c = \mathbf{w}_i + k\mathbf{e}$ and $\mathbf{p}_d = \mathbf{w}_i - k\mathbf{e}$. Similar to the quantity $u(i)$ in (7), we define $v(i) = \sum_{k=1}^{N_r} \text{sgn}\,(I(\mathbf{p}_c) - I(\mathbf{p}_d))$. A nonwinner neuron i is said to be near the region boundary, if either $|u(i)| = N_r$ or $|v(i)| = N_r$, i.e., a nonwinner neuron must satisfy the region boundary condition in either the contour-normal or contour-tangential direction. The above convergence criteria are more effective than those of ACMs based on SOM [14] and TASOM [15], for the following reasons. (a) In [14], the algorithm is terminated after a user-specified number of iterations. However, this criterion may not yield the desired contour even after the specified number of iterations. (b) In order to overcome the above problem, the algorithm in [15] checks whether 1) every neuron is very near a feature point and 2) no new neuron has been inserted in an epoch to ensure convergence. These criteria seem to be very strict, and are not satisfied when the initial rough edge map (feature points) is broken. **The new BSOM overcomes these drawbacks of SOM and TASOM by employing weaker criteria so as to achieve proper convergence even in the case of weak or broken edges**.

[6]This method of insertion of neurons is similar to that of [19].

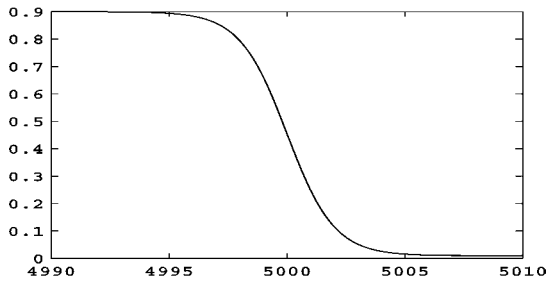
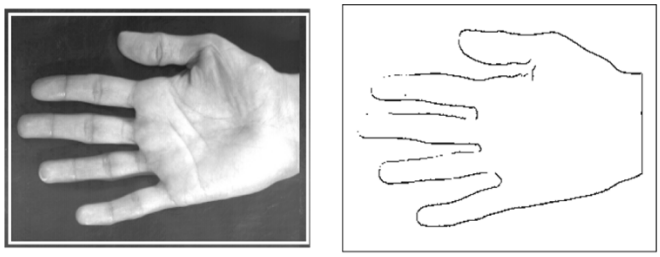


Fig. 3. Plot of the learning rate with respect to time (number of epochs).

III. Results

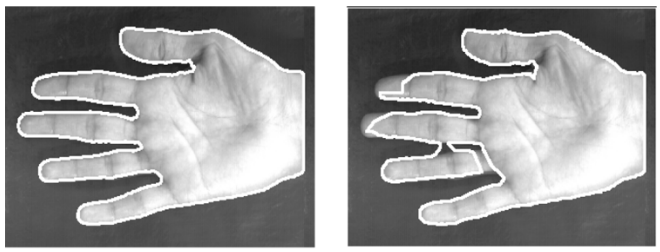
The BSOM algorithm has been tested on the following types of images: 1) synthetic binary, 2) gray (captured in a laboratory set up), and 3) biomedical (brain and lung MRI). On binary and natural images, our algorithm has been applied with and without the damping factor p [defined in (5)]. It is found that the damping factor facilitates convergence to the correct contour [18]. As applied to binary images, we have also studied the effect of noise on its convergence: **The BSOM algorithm gives a good result when the input image is filtered with a Gaussian filter; otherwise, for proper convergence, the initial contour must be close to the actual boundary** [18]. For comparison with TASOM, 1) the utility of SLWs enabling a contour to move into boundary concavity and 2) the effect of a broken edge map have also been analyzed.

A remark on the objective criteria for establishing the accuracy of the extracted contours is in order. As far as synthetic images are concerned, the exact boundaries are known by the synthesis procedure itself. In this case, the contours have been checked against the exact boundaries of objects. On the other hand, for natural images, there seems to be no satisfactory analytical criterion to establish the correctness of the extracted contours. In our experiments, we have superimposed the extracted contour on the original image, and checked visually whether the output of the BSOM-algorithm matches with our visual interpretation of the actual contour. The general observation is that the extracted contours do match satisfactorily the underlying actual contour. See [18] for more examples. In what follows, the



(a) (b)

Fig. 4. (a) Hand image along with the initial contour. (b) The set of feature points (**faulty**) for the hand image.



(c) (d)

Fig. 5. (c) Final converged contour obtained using the BSOM algorithm. (d) Final contour obtained using TASOM.

the images are shown in grayscale (or black for binary images) with the contours overlaid in white for easy visualization (this applies to the results in Section V also).

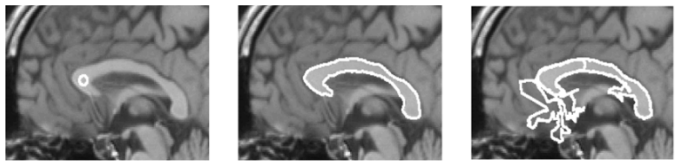
In order to study the effects of a broken edge map (i.e., **faulty feature points**), we show, in Fig. 4(a) and (b), a "hand-image" with (a) the initial contour and (b) feature points, and, in Fig. 5(c) and (d), the results of BSOM and TASOM algorithms. The latter is solely guided by the feature points, and hence the contour leaks through the weak edges. In contrast, in the BSOM, information from gradient and intensity variations is used to prevent the contour from penetrating the true boundary.

Fig. 6(a) shows a synthetic spiral with the initial contour. The contour obtained from the BSOM is shown in Fig. 6(b) and from the TASOM in Fig. 6(c), which does not correspond to the desired boundary due to high concavities in the original contour. In Fig. 7, an MRI image of the brain, it is found that the contours obtained by the BSOM visually (i.e., *perceptually*) match with the actual contours; and are *superior* to those of TASOM and can also be compared favorably with those of [11] and [20].

As far as **computational requirements** are concerned, the complexity of the algorithm is a function of 1) the number of feature points, 2) concavities in the pattern to be detected, and 3) the learning rate. If these are fixed, then the BSOM, in view of the parallel updating scheme employed, is found to be much faster than other algorithms, including TASOM and MTASOM. In Table I, we have given the execution times $(\mathcal{T})$ on a Linux platform with a Pentium-III 450-MHz CPU, for BSOM and TASOM as applied to typical images. $\mathcal{T}$ is found to depend more on the **complexity of the object** rather than on the number of feature points. In other words, if there are many **concavities** in the object boundary, then $\mathcal{T}$ required is more. For instance, $\mathcal{T}$ for the "spiral" image is more than that of the "brain" image, even though the number of feature points in the latter

(a) (b) (c)

Fig. 6. (a) "Spiral" image along with the initial contour. (b) Final converged contour obtained using the BSOM algorithm. (c) Final contour obtained using TASOM.



(a) (b) (c)

Fig. 7. (a) Brain MRI image along with the initial contour. (b) Final contour obtained using the BSOM algorithm. (c) Final contour obtained using TASOM algorithm.

TABLE I
$\mathcal{T}$ FOR BSOM AND TASOM AS APPLIED TO TYPICAL IMAGES

Image name	No: of feature points	Execution time, $\mathcal{T}$	
		BSOM	TASOM
"hand"	2036	51s	521s
"spiral"	1927	110s	1246s
"brain"	2629	15s	169s

is greater than that of the former. This is because, the "spiral" image has more concavities compared to the "brain" image. BSOM needs a larger memory but is amenable to optimization to further reduce $\mathcal{T}$.

IV. EXTRACTION OF MULTIPLE CONTOURS

When images containing multiple objects [Fig. 9(a)] are the inputs to the BSOM algorithm, we observe that 1) after a few epochs, all the FLW neurons [shown in white in Fig. 9(b)] latch on to the nearest feature points, and 2) the segments of *contiguous nonwinners* (indicated in black) are formed in regions where the objects are separated. This is because *the BSOM algorithm is capable of handling* (like most of the ACM-based algorithms) *only one contour.* We now generalize the BSOM algorithm to extract contours of multiple objects. To this end, we first initialize the contour enclosing all the objects in the image, and a neural network isomorphic to the initial contour is created in a manner similar to the BSOM algorithm. The goal is to split the contour, *as the neural network evolves*, into "subcontours" in such a way that each of the subcontours latches on to the boundary of the nearest object. Briefly, the strategy adopted is as follows: after each epoch of the BSOM algorithm, we check every contour for a splitting criterion. When the splitting criterion is satisfied, the contour is ready to be split. With respect to that contour, we check for segments of *contiguous* nonwinner neurons *which usually exist in the gaps between the objects*. We find valid points in these segments for splitting the contour, and then split the contour. However, we desire that *a contour should not be split in such a way that, the resultant subcontours*

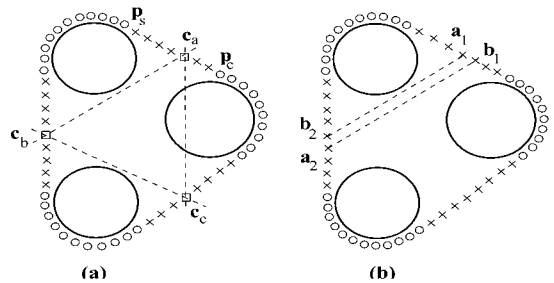


Fig. 8. (a) Contiguous segments of nonwinners and their centroids. (b) Splitting of the contour.

cut across the actual boundaries of the objects. We begin with an initial contour enclosing all the objects, and execute all the steps in the BSOM algorithm. For extracting multiple contours, after each epoch of the basic BSOM algorithm, we check if any contour can be split based on a "splitting criterion" (explained below). If any of the contours is split, we update the contour set $\mathfrak{P}$ to include the newly created contours. After performing the steps of updation, deletion and insertion of neurons of BSOM, for each contour $\mathcal{P}_k \in \mathfrak{P}$ we check for the following split-criterion: *Let i denote a FLW in $\mathcal{P}_k$. If all the FLWs satisfy the region boundary criterion i.e., $R(i) = 1$, then the contour $\mathcal{P}_k$ is said to satisfy the split-criterion*. Subsequently, for splitting the contour, we identify contiguous segments of nonwinners and compute the centroid of these segments. The splitting of a contour is always along the centroids of a chosen pair of segments. In Fig. 8(a), the line joining the centroids $\mathbf{c}_a$ and $\mathbf{c}_c$ contains feature points (edge points of a circle). Therefore, a split along $\mathbf{c}_a$ and $\mathbf{c}_c$ is considered invalid. However, a split either from $\mathbf{c}_a$ to $\mathbf{c}_b$ or from $\mathbf{c}_b$ to $\mathbf{c}_c$ is valid. Among these two valid splits, the one with the least distance between the centroids (points of cutting) is chosen to divide the contour into two [see Fig. 8(b)]. Further details of the splitting procedure are given in [18]. In the course of execution of the algorithm, for finding the SLWs for a feature $\mathbf{x}_k$, an additional constraint is enforced: the contour in which the FLW of $\mathbf{x}_k$ is present is identified (this is unique), and only nonwinner neurons in this contour are allowed to become second-level winners for $\mathbf{x}_k$. In other words, *the contour ID associated with the FLW of $\mathbf{x}_k$ and the SLW must be the same*. This is done to ensure that there is no "interference" between the contours, i.e., neurons in one contour do not become winners of feature points enclosed by a different contour. The remaining steps are the same as in BSOM. The complexity of the algorithm depends not only on the parameters mentioned earlier in Section III but also on the number of contours in the image. However, this dependence is marginal (see Table II).

V. Results For Multiple Objects

The extended BSOM algorithm has been tested on binary and gray images with convex and nonconvex contours.

1) Binary image with the initial contour, Fig. 9(a) and contour after some epochs when the split criterion is satisfied, Fig. 9(b). Nonwinners in the contour are shown in black, and the FLWs in white. Splitting occurs at the centroids of segments which have contiguous nonwinners, and the two new contours do not intersect each other nor do they intersect any object boundary. As the contours evolve, each undergoes further splitting. The various stages are shown in Figs. 9 and 10.
2) Gray-level image with initial and final contours, Fig. 11(a) and (b).

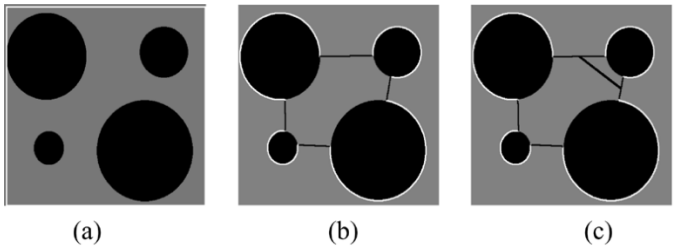


Fig. 9. (a) Image containing four objects with the the initial contour enclosing all objects. (b) The output of the BSOM algorithm after a few epochs. (c) Contour after first split.

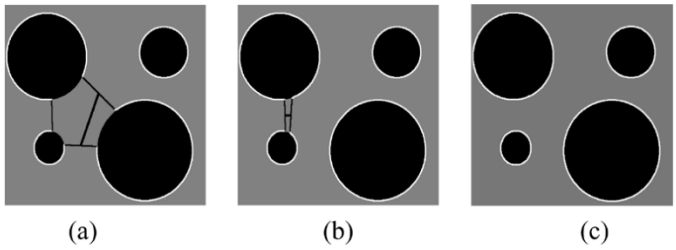


Fig. 10. Continuation of Figs. 9(b) and (c). (a) Contour after second split. (b) Contour after third split. (c) The converged contours.

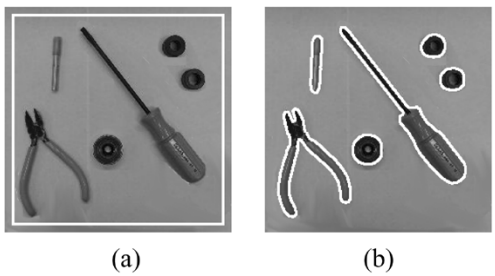


Fig. 11. (a) Image along with initial contour. (b) The converged contours.

TABLE II
Execution Time for BSOM Algorithm as Applied to Images With Multiple Objects

Image name	No: of objects	No. of features	Execution time of BSOM
"4 circles"	4	2406	40s
"Tool set"	6	1890	22s

In general, the final contour encloses only the outer boundaries of the objects in the images. *The algorithm cannot detect contours inside a object, unless the initial contour is given inside the boundary*.

Table II lists the execution times of our algorithm on some typical images containing multiple objects. Note that the execution time depends significantly on the concavities of the objects, and only marginally on the number of objects.

VI. Conclusion

We have proposed a novel algorithm, called the BSOM algorithm, for multiple contour extraction in binary and gray-level images, by integrating the conventional SOM and snakes. It uses 1) the feature points for guiding the contour (as is done in conventional SOM-based ACM) and 2) the gradient and intensity variation information for controlling the movement of the contour. Some of its novelties and advantages are as follows. 1) The contour converges to the nearest salient boundary, resulting in robustness of the algorithm to (a) feature points existing far away and (b) thick edges. 2) The concept of *second-level winners* facilitates the movement of the contour into boundary concavities. 3) Use of **intensity variations** and **gradient information** to ensure that the contour does not penetrate the true boundary. 4) There is *no explicit gradient energy term in contrast with the snake-based ACM*. The BSOM algorithm has been extended to extract contours of multiple objects in images by introducing a splitting procedure. The extracted contours *visually match* with the actual contours. The proposed method is quite distinct from the charged-particle model (CPM) of [12] (see footnote on page 1), and has matching characteristics. A few illustrations of the applications of the algorithm to 1) synthetic binary, 2) gray, and 3) biomedical images are presented to show its superiority, in speed and performance, to the existing results in active contour modeling.

The limitations of the new algorithm are: 1) contours inside an object cannot be extracted if the initial contour is outside the object (or objects) and 2) noisy images require preprocessing.

Acknowledgment

The authors would like to thank the reviewers for their valuable comments and suggestions which have resulted in the present, vastly improved version of this paper.

References

[1] A. Martelli, "Edge detection using heuristic search methods," *Comput. Graph. Image Process.*, vol. 1, pp. 169–182, 1972.

[2] A. W. M. Kass and D. Terzopoulos, "Snakes: Active contour models," in *Proc. 1st IEEE Conf. Computer Vision*, 1987, pp. 259–268.

[3] L. D. Cohen, "On active contour models and balloons," *Comput. Vis., Graph., Image Process.: Image Understand.*, vol. 53, no. 2, pp. 211–218, 1991.

[4] C. Xu and J. L. Prince, "Snakes, shapes and gradient vector flow," *IEEE Trans. Image Process.*, vol. 7, no. 3, pp. 359–369, Mar. 1998.

[5] F. Leymarie and M. D. Levine, "Tracking deformable objects in the plane using an active contour model," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 5, pp. 617–634, May 1993.

[6] R. P. Grzeszczuk and D. N. Levin, "Brownian strings: Segmenting images with stochastically deformable contours," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 9, pp. 1100–1114, Sep. 1997.

[7] N. Peterfreund, "The velocity snake: Deformable contour for tracking in spatio-velocity space," *Comput. Vis. Image Understand.*, vol. 73, pp. 346–356, 1999.

[8] T. McInerney and D. Terzopolous, "T-snakes: Topology adaptive snakes," *Med. Image Anal.*, vol. 4, pp. 73–91, 2000.

[9] V. Caselles, F. Catte, T. Coll, and F. Dibos, "A geometric model for active contours," *Numer. Math.*, vol. 66, pp. 1–31, 1993.

[10] R. Malladi, J. Sethian, and B. Vemuri, "Evolutionary fronts for topology independent shape modeling and recovery," in *Proc. 3rd Eur. Conf. Computer Vision*, 1994, pp. 1–13.

[11] X. Xie and M. Mirmehdi, "RAGS: Region-aided geometric snake," *IEEE Trans. Image Process.*, vol. 13, no. 5, pp. 640–652, May 2004.

[12] A. C. Jalaba, M. H. F. Wilkinson, and J. B. T. M. Roerdink, "CPM: A deformable model for shape recovery and segmentation based on charged particles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1320–1335, Oct. 2004.

[13] A. J. Abrantes and J. S. Marques, "A class of constrained clustering algorithms for object boundary extraction," *IEEE Trans. Image Process.*, vol. 5, no. 11, pp. 1507–1521, Nov. 1996.

[14] Y. V. Venkatesh and N. Rishikesh, "Self organizing neural networks based on spatial isomorphism for active contour modeling," *Pattern Recognit.*, vol. 33, pp. 1239–1250, 2000.

[15] H. Shah-Hosseini and R. Safabakhsh, "A TASOM-based algorithm for active contour modeling," *Pattern Recognit. Lett.*, vol. 24, pp. 1361–1373, 2003.

[16] S. Haykin, *Neural Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[17] Y. V. Venkatesh, S. K. Raja, and N. Ramya, "On the extraction of multiple contours from binary images using an artificial neural network," *Image Vis. Comput.*, to be published.

[18] ——, "Contour Extraction From Graylevel Images of Single and Multiple Objects Using an Artificial Neural Network," Dept. Elect. Eng., Indian Inst. Sci., Bangalore, India, 2004. Tech. Rep.

[19] S. P. Luttrell, "Hierarchical vector quantization," *Proc. Inst. Elect. Eng.*, vol. 136, pp. 405–413, 1989.

[20] N. Ray, S. T. Acton, T. Altes, E. E. de Lange, and J. R. Brookeman, "Merging parametric active contours within homogeneous image regions for MRI-based lung segmentation," *IEEE Trans. Med. Imag.*, vol. 22, no. 2, pp. 189–199, Feb. 2003.

Y. V. Venkatesh (SM'91) received the Ph.D. degree from the Indian Institute of Science (IIS), Bangalore.

He was an Alexander von Humboldt Fellow at the Universities of Karlsruhe, Freiburg, and Erlangen, Germany; a National Research Council Fellow (USA) at the Goddard Space Flight Center, Greenbelt, MD; and a Visiting Professor at the Institut fur Mathematik, Technische Universitat Graz, Austria. His present research interests include computer and robotic vision; signal processing; pattern recognition; remote sensing and hyperspectral image analysis; and neural networks. He is currently a Professor at IIS, where, until recently, he was the Dean of Engineering Faculty and, earlier, the Chairman of the Electrical Sciences Division.

Dr. Venkatesh is a Fellow of the Indian Academy of Sciences, the Indian National Science Academy, and the Indian Academy of Engineering. He is on the editorial board for the *International Journal of Information Fusion*.

S. Kumar Raja received the B.Tech. degree from the Indian Institute of Technology, Chennai, in 1997, and the M.Eng. degree (in signal processing) from the Departments of Electrical Engineering and Electrical Communication Engineering, Indian Institute of Science (IIS), Bangalore, in 1999.

Currently, he is a Project Associate in the Computer Vision Laboratory, Department of Electrical Engineering, IIS. His research interests include computer vision, wavelets, and neural networks.

N. Ramya received the B.E. degree in electronics and telecommunication engineering from Bangalore University, Bangalore, India, in 2001, and the M.Sc. (Engg.) degree in electrical engineering from the Indian Institute of Science, Bangalore, in 2005.

She is currently an Intern in the Industrial Imaging Laboratory, John F. Welch Technology Center, Bangalore. Her research interests concern image processing, computer vision, and robotics.