9-th Scandinavian Conference on Image Analysis
June 6-9, 1995, Uppsala, Sweden

Parallel computation of dense motion fields using a Hopfield network

Janusz Konrad[1], Marek Zaremba[2], Gerald Chan[3]
and Michel Gaudreau[3]

[1] INRS-Télécommunications, 16 Place du Commerce, Verdun
Québec, Canada, H3E 1H6, *e-mail: konrad@inrs-telecom.uquebec.ca*

[2] Dép. d'informatique, Université du Québec, 101 St-Jean Bosco, Hull
Québec, Canada, J8X 3X7, *e-mail: zaremba@uqah.uquebec.ca*

[3] Industry Canada, 300 Slater Street, Ottawa
Ontario, Canada, K1A 0C8

Abstract

Motion of pixels in time-varying images plays an essential role in video compression. Therefore, to build practical video coders motion estimation must be carried out in real time. Usually, simple motion models executed on a sequential processor achieve that goal; VLSI circuits implementing block matching are used in MPEG and H.261 coders. An alternative is to use more complex motion models that can be implemented on a parallel architecture, e.g., single-instruction multiple-data (SIMD) system. In this paper, we study a different approach to the parallelization of motion estimation, an approach based on neural networks. We formulate the problem in the context of a Markov random field (MRF) model, derive a cost function for minimization and propose a solution method using a Hopfield network. We simulate the network on a sequential processor and compare its performance with a sequential algorithm based on the Gauss-Newton minimization.

1 Introduction

Knowledge of 2-D motion is essential for the compression of video sequences. This is due to the fact that temporal correlation in dynamic images is the highest in the direction of motion, and therefore the most efficient redundancy removal is carried out along motion trajectories. This approach is used in motion-compensated predictive or hybrid coders, the most advanced video coders used in practice today. Several recent and emerging standards are based on this approach, e.g., MPEG, H.261. In a coder based on the above standards motion estimation for a complete image must be executed in real time, i.e., in less than 1/60-1/50 s.
In order to perform motion computation in real time, either the number of computed parameters must be reduced or a motion model permitting parallel computation must be used. In the first case, the simplest example is that of block matching

where single vector describes motion of a rectangular block of pixels. Although this model is inaccurate since real images do not have a block structure, it has been very successful because of its simplicity (MPEG, H.261); there exist specialized integrated circuits executing block matching at video rates. This approach, however suffers from blocking artifacts, especially visible at high compression ratios in areas where an object boundary crosses a block. Blocking artifacts can be eliminated by computing one motion vector for every pixel. This approach permits arbitrary object boundaries and substantially reduces prediction error. The reduction of the intensity-related bit rate is offset, however, by an increase of the motion-related bit rate. This trade-off is currently an area of intense research.

To permit parallel computation of motion, we have recently proposed MRF models [8, 9] that naturally lead to algorithms executed on SIMD processors [11]. Neural networks have also been proposed for parallel computation of motion. Zhou and Chellappa [12] have solved a discrete state space problem using a Hopfield network with binary outputs, while Fang *et al.* [2] have extended this algorithm. In this "winner-take-all" approach, one neuron is assigned to each possible motion state at each pixel. For the ITU-R 601 image size (digital TV), maximum displacement of 20 pixels and motion precision of 1/4-pixel, this network requires $(720 \times 480) \times (161 \times 161) \approx 9 \cdot 10^9$ neurons. Li and Wang [10] have also computed optical flow, as formulated by Horn and Schunck [5], using a Hopfield network. As it has been shown in [8], however, the algorithm of Horn and Schunck does not perform well for moderate or fast motion, and also cannot be applied for motion fields defined over arbitrary sampling lattices.

In this paper, we show how to compute motion over continuous state space, as proposed in [8], using Hopfield neural network with graded response [4]. First, we briefly review such a network. Then, we formulate the motion estimation problem and map it onto such a network. Finally, we compare experimentally the new approach with the sequential one proposed in [8].

2 Hopfield network for motion estimation

From the variety of neural networks we chose the Hopfield neural network [3], [4] for its recurrent structure. This structure permits a vast range of responses in comparison with non-recurrent networks. We chose a Hopfield network consisting of neurons with graded response [4], i.e., such that the output of each neuron takes on a continuity of values as oppose to two states [3]. In consequence the number of neurons in the network is reduced and each motion vector is calculated with a higher precision. Additional gain is due to the analog nature of computations; no digital operations are needed and thus calculations are very rapid (the speed is limited only by propagation constraints of the electrical signals). This is similar to the resistive nets proposed in [6].

Such a continuous Hopfield network is described by the following equations;

$$\frac{du_i(t)}{dt} = -\frac{\partial E(v)}{\partial v_i}, \qquad v_i(t) = f(u_i(t)), \tag{1}$$

where t is time, u_i and v_i are the internal state and the output of neuron i, and E is an energy function that is minimized by the network. The non-linearity f provides

a continuous transition between 0 and 1, and is often chosen as $f(y) = 1/(1+e^{-\xi y})$, with ξ being a constant that controls the slope of transition. For the specific case of quadratic energy function $E(v)$

$$E(v) = \frac{1}{2}\sum_{i,k} \alpha_{i,k} v_i v_k + \sum_i \theta_i v_i. \tag{2}$$

equations (1) become:

$$\frac{du_i(t)}{dt} = -\sum_k \alpha_{i,k} v_k(t) - \theta_i, \qquad v_i(t) = f(u_i(t)), \tag{3}$$

where θ_i is the input to neuron i and $\alpha_{i,k}$ is the synaptic weight from the output of neuron i to the input of neuron k.

3 Problem formulation

Since formulation is not the prime topic of this paper, we refer the reader to [8] and [9] for details. Below we briefly explain how we arrive at the cost function.
Let $g(\mathbf{x}, t)$ be an image pixel at spatio-temporal location $(\mathbf{x}, t)$ that belongs to image sampling lattice Λ_g. Let $\mathbf{d}(\mathbf{x}, t)$ be a 2-D displacement vector at $(\mathbf{x}, t) \in \Lambda_{\mathbf{d}}$, where $\Lambda_{\mathbf{d}}$ is the sampling lattice for the displacement vectors. Note that, in general, $\Lambda_{\mathbf{d}} \neq \Lambda_g$, which is an important case for interpolative coding or sampling structure conversion. Assuming that image intensities do not vary along motion trajectories, we can write the *displaced pixel difference* as follows

$$r(\mathbf{x}_i, t, \mathbf{d}) = g(\mathbf{x}_i + (1-\beta)\mathbf{d}(\mathbf{x}_i, t), t_+) - g(\mathbf{x}_i - \beta\mathbf{d}(\mathbf{x}_i, t), t_-) = 0, \quad (\mathbf{x}_i, t) \in \Lambda_{\mathbf{d}} \tag{4}$$

where t_- and t_+ denote time instants of images from which motion is calculated and $\beta = (t - t_-)/(t_+ - t_-)$. The above equation is true only for ideal data, i.e., data without noise, aliasing, distortion, etc. In practice, $r(\mathbf{x}_i, t, \mathbf{d})$ rarely equals zero, however it should be small. Thus, the task is to find such motion field $\mathbf{d}$ that minimizes, e.g., $r^2(\mathbf{x}_i, t, \mathbf{d})$ for all $i = 1, ..., N$, where N is the number of pixels in the image. Since for every $r(\mathbf{x}_i, t, \mathbf{d})$ there are two unknowns (two components of the vector $\mathbf{d}$), the problem is underconstrained. This is often referred to as the *aperture problem*; only one component of motion can be locally recovered. Since the problem is also *ill-posed*, we exploit the prior knowledge about the estimate by using a MRF model [9]. This model requires that motion fields be spatially smooth. We solve the motion estimation problem by minimizing the following cost function with respect to the displacement field $\mathbf{d}$

$$E(\mathbf{d}) = \sum_{i=1}^{N} [r^2(\mathbf{x}_i, t, \mathbf{d}) + \lambda \sum_{j \in \eta(i)} \|\mathbf{d}(\mathbf{x}_i, t) - \mathbf{d}(\mathbf{x}_j, t)\|^2], \tag{5}$$

where $\eta(i)$ is a spatial neighbourhood of $\mathbf{x}_i$ in the MRF sense and $\|\cdot\|$ denotes the L_2 norm. Without loss of generality we consider only the first-order neighbourhood in this paper, i.e., north, east, south and west neighbours.

4 Solution via analog Hopfield network

The cost function (5) is non-quadratic in $\mathbf{d}$ due to the dependence of the displaced pixel difference (4) on $\mathbf{d}$ via luminance g. In order to use the Hopfield network described in Section 2 the cost function (5) must be approximated by a quadratic form similar to (2). Therefore, at each $\mathbf{x}_i$ we expand $r(\mathbf{x}_i, t, \mathbf{d})$ in a Taylor series with respect to some displacement $\dot{\mathbf{d}}(\mathbf{x}_i, t)$ that is assumed known. Rejecting second- and higher-order terms, we have

$$r(\mathbf{x}_i, t, \mathbf{d}) \approx r(\mathbf{x}_i, t, \dot{\mathbf{d}}) + \nabla_{\mathbf{d}}^T r|_{\dot{\mathbf{d}}} \cdot (\mathbf{d}(\mathbf{x}_i, t) - \dot{\mathbf{d}}(\mathbf{x}_i, t)) \tag{6}$$

with the gradient calculated as follows

$$\nabla_{\mathbf{d}} r|_{\dot{\mathbf{d}}} = \begin{bmatrix} \frac{\partial g(\mathbf{x}_i - \beta\dot{\mathbf{d}}(\mathbf{x}_i, t_-))}{\partial x}\beta + \frac{\partial g(\mathbf{x}_i + (1-\beta)\dot{\mathbf{d}}(\mathbf{x}_i, t_+))}{\partial x}(1-\beta) \\ \frac{\partial g(\mathbf{x}_i - \beta\dot{\mathbf{d}}(\mathbf{x}_i, t_-))}{\partial y}\beta + \frac{\partial g(\mathbf{x}_i + (1-\beta)\dot{\mathbf{d}}(\mathbf{x}_i, t_+))}{\partial y}(1-\beta) \end{bmatrix} \triangleq \begin{bmatrix} r^x(\mathbf{x}_i, t, \dot{\mathbf{d}}) \\ r^y(\mathbf{x}_i, t, \dot{\mathbf{d}}) \end{bmatrix}.$$

The vector $\dot{\mathbf{d}}(\mathbf{x}_i, t)$ is supposed to be known, for example from the previous iteration of an iterative algorithm. Thus, the gradient $\nabla_{\mathbf{d}} r(\mathbf{x}_i, t, \dot{\mathbf{d}})$ must be recalculated with each change of $\dot{\mathbf{d}}(\mathbf{x}_i, t)$.
Using the linearization (6) we obtain the following quadratic approximation of cost function (5):

$$\begin{aligned} E'(\mathbf{d}) = & \sum_{i=1}^{N} \{[r(\mathbf{x}_i, t, \dot{\mathbf{d}}) + \nabla_{\mathbf{d}}^T r|_{\dot{\mathbf{d}}} \cdot (\mathbf{d}(\mathbf{x}_i, t) - \dot{\mathbf{d}}(\mathbf{x}_i, t))]^2 + \\ & \lambda \sum_{j \in \eta(i)} \|\mathbf{d}(\mathbf{x}_i, t) - \mathbf{d}(\mathbf{x}_j, t)\|^2\}. \end{aligned} \tag{7}$$

We write this function in a simplified form

$$E'(\mathbf{d}) = \sum_{i=1}^{N} \{[r_i + r_i^x(d_i^x - \dot{d}_i^x) + r_i^y(d_i^y - \dot{d}_i^y)]^2 + \lambda \sum_{j \in \eta(i)} (d_i^x - d_j^x)^2 + (d_i^y - d_j^y)^2\} \tag{8}$$

where

$$\begin{aligned} & r_i = r(\mathbf{x}_i, t, \mathbf{d}), \quad r_i^x = r^x(\mathbf{x}_i, t, \mathbf{d}), \quad r_i^y = r^y(\mathbf{x}_i, t, \mathbf{d}), \\ & d_i^x = d^x(\mathbf{x}_i, t), \quad d_i^y = d^y(\mathbf{x}_i, t), \quad \dot{d}_i^x = \dot{d}^x(\mathbf{x}_i, t), \quad \dot{d}_i^y = \dot{d}^y(\mathbf{x}_i, t), \end{aligned}$$

and d^x and d^y denote the horizontal and vertical components of $\mathbf{d}$. Expanding the square and regrouping the terms we write (8) in a form similar to (2):

$$\begin{aligned} E'(\mathbf{d}) = \sum_{i=1}^{N} & [\lambda|\eta(i)| + (r_i^x)^2](d_i^x)^2 + [\lambda|\eta(i)| + (r_i^y)^2](d_i^y)^2 + 2r_i^x r_i^y d_i^x d_i^y + \\ & \lambda \sum_{j \in \eta(i)} [(d_j^x)^2 + (d_j^y)^2] - 2\lambda \sum j \in \eta(i)[d_i^x d_j^x + d_i^y d_j^y] + \\ & (2r_i r_i^x - 2(r_i^x)^2 \dot{d}_i^x - 2r_i^x r_i^y \dot{d}_i^y) d_i^x + (2r_i r_i^y - 2(r_i^y)^2 \dot{d}_i^y - 2r_i^x r_i^y \dot{d}_i^x) d_i^y, \end{aligned} \tag{9}$$

where $|\eta(i)|$ denotes the number of sites in the neighbourhood of $\mathbf{x}_i$ (4 in the interior and 2 or 1 at image boundary). In the above energy there are quadratic terms

of the type $\alpha d_i d_k$ as well as linear terms of the type θd_i. Thus, we can calculate the synaptic weights $\alpha_{i,k}$ and inputs θ_i from (2) by performing a transformation. Using a second summation to distinguish between the horizontal and vertical motion components (d_i^x and d_i^y), (9) can be rewritten as follows:

$$E'(\mathbf{d}) = \sum_{i=1}^{N}\sum_{m=1}^{2} \{[\lambda|\eta(i)| + (r_i^m)^2](d_i^m)^2 + \sum_{n\neq m} r_i^m r_i^n d_i^m d_i^n + \lambda \sum_{j\in\eta(i)} (d_j^m)^2 - 2\lambda \sum_{j\in\eta(i)} d_i^m d_j^m + 2r_i r_i^m d_i^m - 2r_i^m d_i^m \sum_{p=1}^{2} r_i^p \dot{d}_i^p\}. \quad (10)$$

The superscripts n, m, p denote the component of a motion vector or a gradient; $n = m = p = 1$ denotes the horizontal component, i.e., d_i^x or r_i^x, while $n, m, p = 2$ denotes the vertical component, i.e., d_i^y or r_i^y.
Since our goal is to minimize $E'(\mathbf{d})$ using a Hopfield network, we write the reference energy (2) for the case of unknowns $\{d_i^m, i = 1, ..., N, m = 1, 2\}$ as follows:

$$E'(\mathbf{d}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{2}\sum_{k=1}^{N}\sum_{n=1}^{2} \alpha_{i,m;k,n} d_i^m d_k^n + \sum_{i=1}^{N}\sum_{m=1}^{2} \theta_{i,m} d_i^m. \quad (11)$$

Comparing the corresponding terms in (10) and (11), we obtain the synaptic weights α and the inputs θ:

$$\alpha_{i,m;k,n} = 2[2\lambda|\eta(i)| + (r_i^m)^2]\delta_{i,k}\delta_{m,n} + 2r_i^m r_i^n \delta_{i,k}(1-\delta_{m,n}) - 4\lambda\delta_{m,n}\sum_{j\in\eta(i)} \delta_{j,k} \quad (12)$$

$$\theta_{i,m} = 2r_i r_i^m - 2r_i^m \sum_{p=1}^{2} r_i^p \dot{d}_i^p, \quad (13)$$

where $\delta_{i,k}$ is the discrete Dirac's impulse.
Note that the inputs θ depend only on images g and on intermediate solutions $\dot{\mathbf{d}}$. The synaptic weights α, however, depend on g, $\dot{\mathbf{d}}$, the regularization parameter λ and on the neighbourhood η. Due to the summation of $\delta_{j,k}$ in (12), non-zero weights α interconnect only those neurons which correspond to sites that are neighbours of each other. Therefore, the system is very sparse with only a few weights connected to every neuron's input (5 for the first-order neighbourhood). Since λ and η are fixed, their contribution to α can be precomputed. However, since $\dot{\mathbf{d}}$ evolves in time, r's in the synaptic weights α and the inputs θ must be recalculated with each change of $\dot{\mathbf{d}}$. Usually, several updates of $\dot{\mathbf{d}}$ are needed to arrive at a good estimate of $\mathbf{d}$. The computation of r's must be done before down-loading α's and θ's into the network. This will require a specialized processor capable of handling pixel interpolation (for the entire frame) in real time, e.g., a *hybrid* filter that takes a discrete-time input and produces a continuous-time output. Given α's and θ's, the Hopfield network converges to a solution very quickly; the speed is only limited by the propagation constraints of electrical signals.
Finally, the discrete-time update equations for the network to be simulated on a sequential computer are ($m = 1, 2$):

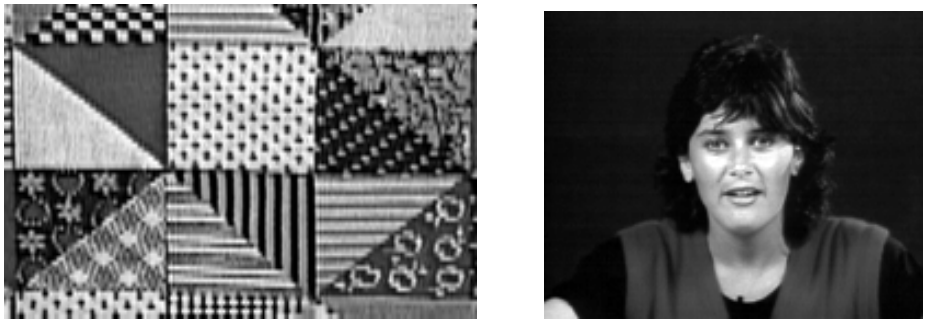
$$u_i^m(t+1) = u_i^m(t) - \sum_{k=1}^{N}\sum_{n=1}^{N} \alpha_{i,m;k,n} d_k^n(t) - \theta_{i,m}, \quad d_i^m(t) = f(u_i^m(t)). \quad (14)$$

5 Simulation results

We have simulated the above Hopfield network in floating-point precision using update equations (14), synaptic weights (12), inputs (13) and nonlinearity $f(\cdot)$ with ξ=0.0003. To test the proposed algorithm we chose interpolative coding based on motion-compensated interpolation. In this application, motion vectors must be computed at such locations $(\mathbf{x}_i, t)$ that $t \neq t_-$ and $t \neq t_+$. The image at time t is reconstructed at the receiver by means of estimating motion vectors from the transmitted images at t_- and t_+ followed by filtering along the estimated trajectory. If the error between the reconstructed image $\hat{g}$ and the original image is too large, it may be transmitted as well. For the simple case of 2 images used in motion estimation, the reconstruction can be described as follows:

$$\hat{g}(\mathbf{x}, t) = (1-\beta)g(\mathbf{x} - \beta\mathbf{d}(\mathbf{x}, t), t_-) + \beta g(\mathbf{x} + (1-\beta)\mathbf{d}(\mathbf{x}, t), t_+).$$

For cases of filtering with longer temporal support see [1].



(a) *Courtepointe* (b) *Miss America*

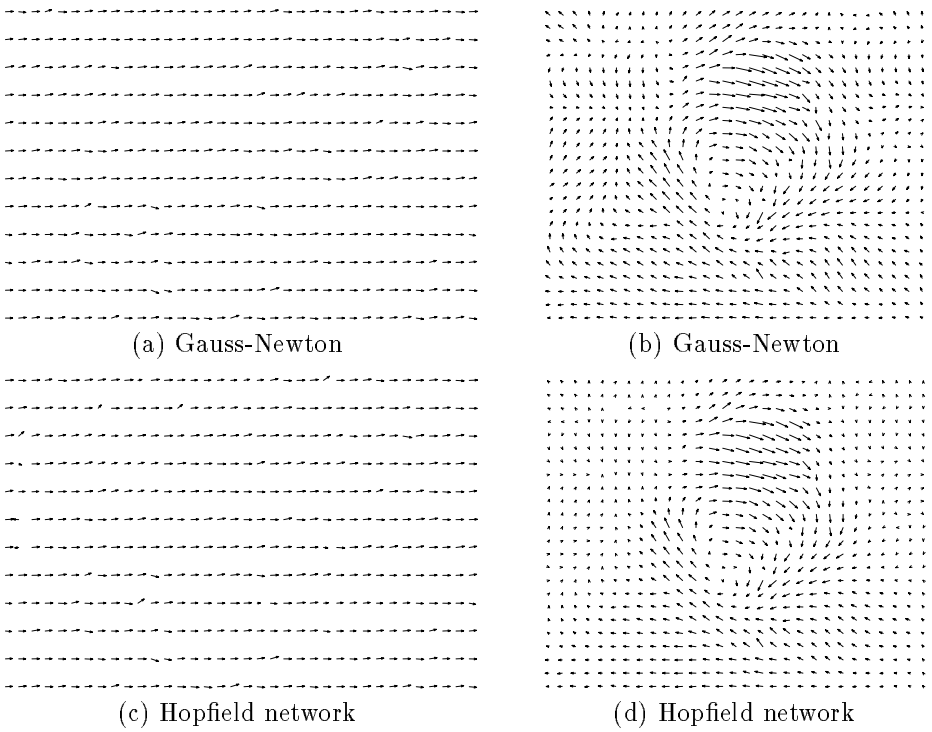
Figure 1: Test images: (a) *Courtepointe* at $t = 1$ (interlaced); and (b) *Miss America* at $t = 2$ (progressive)

We have carried out tests of the proposed algorithm on three image sequences with natural motion: *Courtepointe* containing only translational motion due to camera pan, *Femme et arbre* and *Miss America* both containing complex motion of a human body. The sequence *Miss America* is progressive while the other sequences are interlaced. We have chosen $t = 1$ with $t_- = 0$ and $t_+ = 2$ for *Courtepointe* and *Femme et arbre*. For *Miss America*, however, we have used $t = 2$ with $t_- = 0$ and $t_+ = 4$, which better corresponds to videophone applications where sequences are often temporally subsampled by 3-4 before encoding. The original images are shown in Figure 1. In order to carry out image interpolation, which is needed for vectors with continuous coordinates, we use bicubic interpolation [7].
As the reference for the proposed approach we use the algorithm described in [8]. This method is based on Gauss-Newton minimization of the cost function (5) and gives reliable motion estimates for arbitrary Λ_g and $\Lambda_{\mathbf{d}}$. The method can be implemented in parallel using an SIMD architecture [11], but here it has been simulated sequentially using Gauss-Seidel relaxation to solve the resulting linear system.
To evaluate numerically the performance of both algorithms we have computed the root mean-squared reconstruction error between g and $\hat{g}$ (Table 1). Note that the

	Courtepointe	*Femme et arbre*	*Miss America*
Gauss-Newton	12.15	5.64	5.15
Hopfield network	12.18	5.46	5.15

Table 1: RMS reconstruction error for tested images.

error is almost identical for the Gauss-Newton minimization and for the minimization based on Hopfield network; the proposed approach can be judged equivalent to the reference algorithm. Also subjectively both algorithms give very similar motion fields (Figure 2). The reconstructed images (not shown) look virtually identical as well.



(a) Gauss-Newton (b) Gauss-Newton

(c) Hopfield network (d) Hopfield network

Figure 2: Displacement fields estimated using Gauss-Newton minimization (a,c) and Hopfield network with $\xi = 0.0003$ (b,d). Displacement fields are subsampled by 6 in each direction and magnified by 3 to improve visibility.

6 Summary and conclusions

We have shown how to solve motion estimation in the continuous state space using a Hopfield network with graded response. Simulating this approach on a sequential

processor we have demonstrated that solutions obtained using Hopfield network are subjectively and objectively almost identical to the solutions resulting from the reference algorithm. We have not addressed here the issue of off-line computation of inputs and synaptic weights. Since the network is used solely as an optimization tool, no self-learning rules apply. In the current implementation the inputs and weights have to be recalculated every time the intermediate solution $\dot{\mathbf{d}}$ changes. We are currently studying how to efficiently perform this calculation.

This work was supported by Industry Canada under a contract within "Programme de développement et de promotion des centres d'excellence de langue française".

References

[1] M. Chahine and J. Konrad, "Motion-compensated interpolation using trajectories with acceleration," in *Proc. IS&T/SPIE Symp. Electronic Imaging Science and Technology, Digital Video Compression: Algorithms and Technologies 1995*, vol. 2419, Feb. 1995.

[2] W.-C. Fang, B. Sheu, and J.-C. Lee, "A VLSI neuroprocessor for real-time image flow computing," in *Proc. IEEE Int. Conf. Acoustics Speech Signal Processing*, pp. 2413–2416, May 1991.

[3] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554–2558, 1982.

[4] J. J. Hopfield, "Neurons with graded response have collective computational properties like those with two-state neurons," *Proc. Natl. Acad. Sci. USA*, vol. 81, pp. 3088–3092, 1984.

[5] B. Horn and B. Schunck, "Determining optical flow," *Artif. Intell.*, vol. 17, pp. 185–203, 1981.

[6] J. Hutchinson, C. Koch, J. Luo, and C. Mead, "Computing motion using analog and binary resistive networks," *Computer*, vol. 21, pp. 52–63, Mar. 1988.

[7] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Trans. Acoust. Speech Signal Process.*, vol. ASSP-29, pp. 1153–1160, Dec. 1981.

[8] J. Konrad and E. Dubois, "Comparison of stochastic and deterministic solution methods in Bayesian estimation of 2D motion," *Image Vis. Comput.*, vol. 9, pp. 215–228, Aug. 1991.

[9] J. Konrad and E. Dubois, "Bayesian estimation of motion vector fields," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-14, pp. 910–927, Sept. 1992.

[10] H. Li and J. Wang, "Computing optical flow with a recurrent neural network," *Intern. J. Pattern Recognit. Artif. Intell.*, vol. 7, no. 4, pp. 801–814, 1993.

[11] E. Memin, *Algorithmes et architectures parallèles pour les approches markoviennes en analyse d'images*. PhD thesis, l'Université de Rennes I, June 1993.

[12] Y. Zhou and R. Chellappa, "Computation of optical flow using a neural network," in *Proc. IEEE Int. Conf. Neural Networks*, pp. II–71–II–78, July 1988.