A Vision System with Real-Time Feature Extractor and Relaxation Network

Toshiro Kubota*, Terry Huntsberger†
Intelligent Systems Laboratory, Department of Computer Science
University of South Carolina, Columbia, SC 29208, USA
kubota@cs.sc.edu (803) 777-8306

Cecil O. Alford
Computer Engineering Research Laboratory,
Department of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30324, USA
alford@cerl.gatech.edu

Abstract

This paper reviews our on-going efforts on designing a real-time vision system. The system consists of a real-time feature extractor and a relaxation network. The filter system is capable of performing multiple 2D non-separable filters, multi-resolution decomposition, and steerable transform. The relaxation network is capable of performing various relaxation, diffusion and minimization operations. The paper shows several vision tasks which can be implemented effectively on the vision system.

Keywords: *real-time vision system, feature extraction, relaxation*

1 Introduction

There are at least two application areas that fast vision systems are important. The first one is for real-time applications. Such systems have to operate at a certain required rate to be useful. Applications include automatic target recognition/tracking, factory inspection, face recognition for security check, and robotics. Each application has different timing requirements. The other area is for algorithm development. Many vision algorithms are computationally intensive and time consuming to run on a sequential computer. As a matter of fact, the amount of computation for new algorithms tends to grow as more powerful microprocessors become available to the research community. Thus, we often find ourselves in the situation where we have to wait for hours to collect the results each time we change the value of one parameter. The research community can benefit greatly from a fast inexpensive general purpose vision system.

*Research supported in part under ARO Contract DAAH04-96-10326
†Research supported in part under ARO Contract DAAH04-96-10326 and ONR Contract N00014-94-1-1163

For these reasons, our focus is on designing a general purpose real-time vision system. The system has to be flexible so that various algorithms can be implemented on the platform. The architecture has to be flexible so that the speed of the system can be increased easily with additional hardware to meet various real-time requirements. These concerns have previously been studied in the context of massively parallel and mesh-connected array processor systems (see [13]).

Many vision algorithms and vision tasks can be divided into two stages in terms of the characteristics of the computations involved. They are the feature extraction stage and the localized iterative interaction (relaxation) stage. Some vision tasks which comply with this characterization are region growing, optic flow computation, shape from shading, non-linear diffusion, and Hopfield networks.

The feature extractor operates on either single or multiple images. Here the feature extractor is a combination of linear filters and pixel-wise operations. Thus,

$$\mathcal{F}(I) = \left(\prod_i p_i h_i \star \right) I \tag{1}$$

where $\mathcal{F}$ is the feature extractor, I is the input image, h_i is a convolution kernel, p_i is a pixel-wise operation, $\star$ represents convolution, and $\prod$ implies concatenation of operators. Pre-processing on input images such as noise reduction, temporal averaging and sensor compensation can be integrated into the feature extraction stage.

Feature extraction is a vital part of the computer vision system. It involves various types of filters. For real-time applications, the complexity of the feature extraction process often governs the speed of the system, and the system often settles for less effective but computationally affordable feature extractors.

In [10, 11, 12] a method for a fast feature extraction was proposed. With this method, it became possible to construct a compact vision system which is capable of operating various 2D non-separable spatial filters for feature extraction. The features of the system includes:

- real-time performance of multiple 2D non-separable spatial filters.
- real-time performance of multi-resolution image transform.
- real-time performance of steerable transform of an image.

The system has a simple pipeline structure suitable for VLSI implementation, and has very small latency and memory requirements compared to an FFT based system.

The architecture is general enough so that any types of spatial filters can be implemented on the system. They include Gabor filters, Gaussian filters, difference of Gaussian filters, Canny's edge detector [4], Laws' texture filters [14], various classes of wavelets [19], Freeman-Adelson's steerable filters [6], Simoncelli *et al* shiftable filters [20], Watson's cortex transform [23], Marr-Hildreth filters [17], Burt pyramid [3], Young Gaussian derivatives [24], and Perona deformable kernels [18].

The architecture is scalable in the sense that the number of filters to be implemented and the size of the filters can be increased by adding either additional chips or boards depending on the actual implementation. The amount of hardware increase is linear with respect to the number and the size of filters. The architecture is not dependent on the input image size.

The relaxation stage involves iterative operations on a local neighborhood. Thus,

$$\begin{aligned} \mathcal{R}(I) &= \mathcal{J}(I^{n-1}) \\ I^n &= \mathcal{J}(I^{n-1}) \end{aligned} \tag{2}$$

where $\mathcal{R}$ is the relaxation operator, $\mathcal{J}$ is a local neighborhood operator, I^n is the intermediate result at nth iteration with $I^0 = I$.

Various types of relaxation networks have been proposed in the literature and some have been implemented in hardware for some specific vision tasks [1, 2, 22]. They are either analog, digital or hybrid containing both analog and digital circuits. Silicon implementations which mimic the human retina are often called *silicon retina* and have gained much attention recently [9, 16]. They can also be categorized as relaxation networks. However, they are designed as sophisticated sensors rather than general purpose vision engines.

Section 2 reviews the real-time feature extraction system. Section 3 reviews various vision architectures designed for relaxation and proposes an improved new architecture. Section 4 provides several examples of how various vision algorithms/tasks can be performed on the architecture. Section 5 gives a summary and conclusions.

2 Feature Extraction System

The basic idea of the filter system is to approximate 2D non-separable filters into a set of separable filters. This way, an expensive 2D non-separable filter operation can be done by adding the results of inexpensive separable filters. The approximation can be expressed as

$$h(x,y) \approx \sum_{i=1}^{P} a_i(x) b_i(y). \tag{3}$$

Figure 1 shows the implementation of the separable scheme. It has a simple filter bank structure suitable for VLSI implementation. This section briefly reviews our previous work on the separable filter scheme. For more detail see [10, 11, 12].

2.1 SV/OSD

The system employs Singular Value Decomposition (SVD) to decompose a set of 2D filters into a sum of separable filters. The new method is called SV/OSD in this paper. The difference from the Treitel-Shank's classical decomposition method [21] using SVD is that the SV/OSD takes multiple filters and decomposes them into a separable form simultaneously while the Treitel-Shank's method decompose each filter separately.

Given a set of 2D filters $\{\psi_k(x,y)\}$ $(0 <= k < F_N)$ to be implemented, they are sampled within a discrete lattice to form a set of FIR filters, $\{\psi_k[m,n]\}$. For simplicity, the size of the filters is M by M. Combine all the FIR filters to form an M by MF_N matrix A as shown in Figure 2. The matrix A is called the *approximation matrix*. A set of M adjacent columns from the kM^{th} column through $(k+1)M-1^{th}$ columns constitute the filter ψ_k. SVD is performed on the approximation matrix to produce r_A vectors of the length M, r_A vectors of the length MF_N, and the corresponding eigenvalues where r_A is the rank of the approximation matrix. The first set of vectors are denoted as $\bar{u}_i$ $(0 <= i < r_A)$, the second set of vectors are denoted as $\bar{v}_i$ $(0 <= i < r_A)$, and the eigenvalues are denoted as w_{ii}. Then, h_i can be decomposed into a separable form by

$$\psi_k[m,n] = \sum_{i}^{r_A} w_{ii} \bar{u}_m \bar{v}_{kM+n}. \tag{4}$$

Assume the eigenvalues, w_{ii}, are sorted in decreasing order of magnitude. The advantage of SV/OSD over the Treitel-Shank method is its computational efficiency. Assume there are F_N filters to be implemented, each with an M by M mask size, and the decomposition requires P order approximation to achieve the required accuracy. Then the computational complexity for the Treitel-Shank method is $F_N * 2MP$. The computational complexity for the SV/OSD is $F_N * MP + MP = (F_N + 1) * MP$. Thus, the SV/OSD has less computation by the amount of $(F_N - 1) * MP$. When $F_N = 1$, the two methods become identical.

This computational advantage reflects directly into an implementation advantage. The Treitel-Shank method requires $2F_N P$ filters with size M, while the SV/OSD requires $(F_N + 1) * P$ filters with size M. This comparison is depicted in Figure 3.

The approximated filters are guaranteed to converge to the original filters as the approximation order increases. The SVD guarantees at least linear convergence, however, its convergence is close to exponential in most cases [10]. Due to the property of SV/OSD, Equation (4) is the best rank-P approximation of ψ in the least square sense.

2.2 Multi-resolution decomposition

Multi-resolution decomposition (MRD) is a technique to produce a hierarchical image representation suitable for many image processing and analysis algorithms. It produces multiple levels of image representation. Each level represents the content of an input image over a particular frequency region. The decomposition is done by a set of filters where each filter is tuned to each frequency region. Thus the decomposition can be written as,

$$d^k = f \star \psi^k \quad (1 <= k < L), \tag{5}$$

where d^k is the kth level output, f is the input image, and ψ^k is the kth filter.

In most cases, the set of filters are the results of dilating a prototype filter at different dilation factors. For computational convenience, the dilation factors are powers of 2. Thus,

$$\psi^k[m,n] = 2^{-k+1}\psi(2^{-k+1}m, 2^{-k+1}n) \quad . \tag{6}$$

It is often the case that the lower frequency portions of the decomposition are decimated to reduce the data size and the amount of computation without information loss. For the dyadic dilation case, the decimation factor is also a power of 2. This decimated MRD can be formulated as,

$$d^k_{m,n} = \sum_{i_x, i_y} f[i_x, i_y] 2^{-k+1} \psi(2^{-k+1} i_x - m, 2^{-k+1} i_y - n) \tag{7}$$

This is exactly the dyadic discrete wavelet transform. One of difficulties for computing a MRD is that the filter size grows exponentially as the dilation factor increases. The discrete wavelet transform ameliorates the problem elegantly by imposing the following constraint on the wavelets,

$$\psi(x) = \sum_i h[i]\psi(2x) \quad . \tag{8}$$

Then the decimated MRD can be computed recursively using the fixed length discrete filter h.

If the prototype filter does not satisfy the requirement (8), an approximation can be made to the filter using basic splines. The spline satisfies the dyadic constraint (8) with

$$h[n] = \tilde{g}[n] = 2^{-k-1/2} \begin{pmatrix} k+1 \\ n \end{pmatrix} \quad . \tag{9}$$

This spline approximation together with SV/OSD can compute a MRD efficiently. First, the 2D non-separable prototype filter needs to be decomposed into a set of separable filters. Each 1D filter is approximated using the spline approximation. The MRD is performed recursively using a set of fixed length discrete filters. Assume ψ is the prototype filter. SV/OSD is applied to obtain the separable approximation of the filter. Then

$$\psi(x,y) \approx \sum_{i=1}^{P} a_i(x) b_i(y). \tag{10}$$

Each 1D filter is approximated with the basic spline.

$$a_i(x) \approx \sum_j g_i^x[j]\phi(x-j) \tag{11}$$

$$b_i(x) \approx \sum_j g_i^y[j]\phi(x-j) \tag{12}$$

where ϕ is the basic spline which satisfies

$$\phi(x) = \sqrt{2}\sum_i \tilde{g}[i]\phi(2x-i) \quad . \tag{13}$$

The decomposition is performed recursively in the following manner.

$$d_{m,n}^k = \sum_{i_x,i_y} f[i_x,i_y] 2^{-k+1} h(2^{-k+1}i_x - m, 2^{-k+1}i_y - n)$$

$$\approx \sum_l \sum_{j_x} g_l^x[j_x - m] \sum_{j_x} g_l^y[j_y - n] c_{j_x,j_y}^{k-1}, \tag{14}$$

$$c_{m,n}^k = \sum_{i_x,i_y} f[i_x,i_y] 2^{-k} \phi(2^{-k}i_x - m)\phi(2^{-k}i_y - n)$$

$$= \sum_l \sum_{j_x} \tilde{g}[j_x - 2m] \sum_{j_y} \tilde{g}[j_y - 2n] c_{j_x,j_y}^{k-1}, \tag{15}$$

$$c_{m,n}^0 = \sum_{i_x,i_y} f[i_x,i_y]\phi(m-i_x)\phi(n-i_y). \tag{16}$$

Figure 4 shows this approximation process. The overall process is done in a pipeline fashion with filters in each level operated in parallel. It has a simple filter bank structure.

The above decomposition algorithm decimates the image at every decomposition level. The first level decomposition produces the output as big as the input image. At the kth level of the decomposition, the width and height of the output image reduces to 2^{k-1} of the input. In some vision tasks, it is more desirable to keep the image size intact since the decimation process introduces aliasing and causes the decomposition to be shift variant. This undecimated MRD can be performed using the computational structure shown in Figure 4 with a little modification. The idea is to compute 4 sets of decimated MRDs at different spatial points. The same hardware can be shared for the 4 MRDs. For more detail, see [11].

2.3 Steerable system

On a steerable system, the directional preference of a filter can be adaptively controlled by interpolating the outputs of a set of basis filters. Thus, the steerability can be described as,

$$\psi_\theta(x,y) = \sum_i \gamma_i(x,y) q_i(\theta) \tag{17}$$

Such a steerable system can be designed by first decomposing the filter of interest into Fourier series along the orientation dimension. The transform decouples the orientation and the spatial dimensions.

$$\psi_\theta(x,y) = \frac{1}{2} a_0(x,y) + \sum_{n=1}^{\infty} \{\alpha_n(x,y) cos(n\theta) + \beta_n(x,y) sin(n\theta)\} \tag{18}$$

where

$$\alpha_n(x,y) = \frac{1}{L} \int_{-\pi}^{\pi} \psi_\theta(x,y) cos(n\theta) d\theta \quad , \tag{19}$$

$$\beta_n(x,y) = \frac{1}{L} \int_{-\pi}^{\pi} \psi_\theta(x,y) sin(n\theta) d\theta \quad . \tag{20}$$

By taking the Q $\{\alpha_i, \beta_i\}$ with the largest energy contents, the Qth order Fourier series approximation (FSA) is obtained. The approximated filter is approximately steerable.

$$\psi_\theta(x,y) \approx \sum_{i=1}^{Q} \gamma_i(x,y) q_i(\theta) \tag{21}$$

$$\{\gamma_i\} \subset \{\alpha_i, \beta_i\} \tag{22}$$

$$q_i(\theta) = \begin{cases} 1/2 & : \quad if \gamma_i = \alpha_0 \\ cos(n\theta) & : \quad if \gamma_i \in \{\alpha_i\}, i \neq 0 \\ sin(n\theta) & : \quad if \gamma_i \in \{\beta_i\} \end{cases} \tag{23}$$

The computational structure of this approximation is shown in Figure 5. SV/OSD can be applied to the basis filters $\{\gamma_i\}$ to reduce the amount of computation:

$$\psi_\theta(x,y) \approx \sum_{i=1}^{Q} q_i(\theta) \sum_{j=1}^{P} a_j(y) b_{ij}(x) \tag{24}$$

where

$$\psi_\theta(x,y) \approx \sum_{i=1}^{P} a_i(x) b_i(y). \tag{25}$$

Then the spline approximation can be applied to each 1D filter for efficient steerable MRD transform.

2.4 Architecture

The above feature extraction technique can be implemented with a 1D filter bank structure. There are two ways to implement 1D linear convolution. One is to use a sequence of multiply- accumulate units (pipelined filtering), and the other is to use a set of parallel multipliers followed by a network of adders (parallel filtering). Figure 6 shows the two schemes. They both requires M multipliers and $M-1$ adders for a length M filter. Pipelined filtering takes one input at a time sequentially. The input is multiplied

with all the filter coefficients at the same time, and each multiplication result is accumulated at an accumulator attached to the multiplication unit. The last accumulator holds the final output. Parallel filtering takes M inputs at a time and each input is multiplied with a filter coefficient. The result of the multiplications are added through the binary tree adder. Note that the parallel filtering scheme requires M independent memory banks to provide the parallel inputs to the filter.

Pipelined filtering is suitable for a sequential input stream, and parallel filtering is suitable for a parallel input stream. For SV/OSD, pipelined filtering is suitable for horizontal filters assuming that the inputs are coming in a raster order. Parallel filtering is suitable for vertical filters since the input can be provided in parallel such that the latency of the system reduces to $O(NM)$ where N is the image width.

Assume that horizontal filters are implemented with the pipelined filtering scheme, while vertical filters are implemented with the parallel filtering scheme. A separable filter pair (a_i and b_i in (3)) can be implemented as the horizontal filter first followed by the vertical filter, or vice versa. As noted above, each vertical filter requires a set of parallel input buffers. With the vertical-horizontal filter order, the buffer can be shared among all the vertical filters. Therefore it is more advantageous in terms of the memory requirement to employ the vertical-horizontal filter order.

A VLSI implementation of the filter system consists of 3 custom designed VLSI chips; horizontal filter chip (HFC), vertical filter chip (VFC) and separable filter chip (SFC). They contain multiple 1D filters. A simplified view of each filter is given in Figure 7. Each filter in a HFC is implemented using the pipelined filtering scheme. Each filter in a VFC is implemented using the parallel filtering scheme. There are a pair of horizontal and vertical filters in a SFC. They are implemented using the pipelined and parallel filtering schemes, respectively. With them, various filter extraction systems can be constructed.

Figure 8 shows the system configuration for SV/OSD. It is implemented with HFCs and VFCs. The number of chips required depends on the approximation order, the number of filters, and the size of the filters. An estimate of the number of filters for typical configurations is given in [12].

Figure 9 shows the system configuration for a decimated MRD. It is implemented with HFCs, VFCs and SFCs. Both the basic spline filters ($\phi[]$s) and the low pass banks ($\tilde{g}$) are implemented with SFC, while the high pass banks (g^x and g^y) are implemented with VFCs.

Figure 10 shows the system configuration for FSA. It is similar to the SV/OSD configuration except for the interpolation units which are implemented with a VFC.

3 Relaxation Network

Most relaxation networks including silicon retinas are variations of mesh connected SIMD machines, because the spatial distribution and the local interaction of data are suitable for the mesh topology. The disadvantage of this scheme is that the system performance is heavily dependent on the image size, and the cost of the system is high.

Our approach is to introduce parallelism in the time (or iteration) domain and employ a pipeline structure for the spatial domain. This scheme is modular consisting of multiple processing units (PU), and has the following advantages.

1. The number of PUs can be expanded easily.

2. The system is not dependent on the image size.

3. The system cost is dependent on the number of PUs, thus one can build an inexpensive system with fewer PUs.

4. The system is scalable in the sense that the performance improvement is approximately linear as the number of PUs increases.

The last point needs more explanation. Assume some relaxation algorithm took N_I iterations to converge. Then the system performance increases linearly until the number of PUs reaches N_I.

Disadvantages of the scheme are the following:

1. It introduces larger latency.

2. It requires an intermediate buffer.

Assume the image size is $N \times N$, the system has K PUs, and the algorithm converges after N_I iterations. Then the latency with the mesh topology is $O(N^2 + N_I)$ while the one with our scheme is $O(N^2 NI/K + N_I)$.

3.1 Processing unit

Figure 11 shows the functional diagram of a PU. The design takes advantage of the simple and repetitive nature of the local neighborhood interaction. It consists of a local processor, global processor, local memory, input buffer, local memory address generator (LMAG), and an input buffer address generator (IBAG).

The local memory holds various parameter values for the relaxation operations and look-up tables for some mathematical functions which are not supported in the local processor. The LMAG provides appropriate addresses to the local memory. The input buffer contains the image data. Since the local processor is restricted to neighborhood operations, the buffer needs to contain only a few rows of data. The IBAG provides appropriate addresses to the input buffer. The address computed by LMAG and IBAG can be either an absolute address or an address relative to the current pixel location. At this point, the design of the local processor and the global processor have not been determined fully. However, our philosophy is to keep their design very simple and their functionality to a minimum, since the local neighborhood operations are usually very simple and a simpler design allows higher compaction of PUs in a system.

Figure 12 shows the structure of the local processor. It consists of an input module, output module, 2 multipliers, 2 adders, logical operator, shift register, instruction cache and a register file. The input module directs 8 inputs (2 inputs from the input buffer, 2 inputs from the local memory, 2 inputs from the register file, and 2 feedback paths) to the inputs of 6 arithmetic units. The output module directs the outputs of the 6 arithmetic units and 4 external inputs (from the input buffer and the local memory) to the input module through the feedback paths, to the local memory, to the register file, and/or to the global processor. The instruction cache is loaded with instruction sets at the beginning of the relaxation. The data from the local memory and the input buffer are fetched using the addresses from the address generators. The local processor and the address generators are synchronized so that the instruction set and the address sets repeat for every pixel location. This processor organization is similar to the dual pipeline architecture of Huntsberger and Wood [8] and the SLAP PEs developed at Carnegie Mellon [5].

Figure 13 shows the structure of the global processor. The structure is very similar to the local processor, however, less parallelism is employed here compared to the local processor. It consists of an input module, output module, multiplier, adder logical operator and a shift register. There are only one multiplier and one adder. It allows only one input from the local memory, the input buffer and the register file. There is only one feedback path.

Figure 14 shows the structure of the address generator. The pixel counter determines the base address by the size of data associated for each pixel. At the beginning of the relaxation, the address cache is loaded with either the absolute address or the offset relative to the base address.

3.2 Architecture

Figure 15 shows the system architecture of the relaxation network. It consists of an interface module and multiple PUs connected in cascade. The output of the last PU in the chain is fed back to the beginning of the chain, and the input to the first PU can be chosen between the outputs of the interface module and the feedback. For a static image analysis, the features enter into the PU chain, and the multiplexer is locked in to select the feedback. Then the relaxation network proceeds with its computation until it settles to the convergence limit. For an image sequence analysis, the multiplexer selects new features when they arrive to its input ports. The stable state of the relaxation network can be maintained as an initial state for the new set of image features.

The interface module performs data conversion if necessary between the feature extractor and the relaxation network. It also converts the parallel outputs from the feature extractor to the sequential input for the relaxation network.

4 System Examples

This section provides examples of how various vision tasks can be implemented using the vision system. The example tasks are optical flow computation, shape from shading, K-means clustering and Mumford-Shah segmentation.

4.1 Optical flow computation

this process requires at least two consecutive images to compute optical flow using the original Horn and Schunck algorithm.

The feature extractor can be used to remove noise by smoothing and temporal dither by a low-pass temporal filter. The main portion of the computation is left for the relaxation network. An estimate of optical flow is obtained at the minimum of the following energy equation.

$$E = (f_x u + f_y v + f_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2) \ , \tag{26}$$

where (u, v) is an optical flow estimate, u_x, u_y, v_x and v_y are the spatial differences of the flow field, f_x and f_y is the spatial difference of the input image f, f_t is the temporal difference of the images, and λ is the Lagrange multiplier. The relaxation for the two-frame optical flow computation becomes

$$\begin{aligned} u^n &= \bar{u}^{n-1} - f_x P/D \ , \\ v^n &= \bar{v}^{n-1} - f_y P/D \ , \end{aligned} \tag{27}$$

where $\bar{u}$ and $\bar{v}$ is the local average of the estimate, and

$$P = f_x \bar{u} + f_y \bar{v} + f_t$$
$$D = \lambda^2 + f_x^2 + f_y^2. \tag{28}$$

The feature extractor and the relaxation network are set up in the following way. The feature extractor simply computes the spatial differences and the temporal difference of the input images. These features as well as D^{-1} are stored in the local memory of each PU as parameters. The system loads the parameters to the memory as the relaxation proceeds. The local processor computes $\bar{u}$, $\bar{v}$, u^n, and v^n using (27). The global processor computes $u^n - u^{n-1}$ and $v^n - v^{n-1}$ to check the convergence of the relaxation.

4.2 Shape from shading

Assume that the location of light sources and the surface reflectance are known. The algorithm works in a similar way to the optical flow computation. It solves for the surface normal (p, q) based on the reflectivity constraint and the smoothness constraint. Denote the reflectance map of the surface of interest as $R(p, q)$ and the image pixel intensity as $I(x, y)$. Then the reflectivity constraint is

$$I(x, y) = R(p(x, y), q(x, y)) \ . \tag{29}$$

The smoothness constraint is

$$p_x^2 + p_y^2 + q_x^2 + q_y^2 = 0 \ . \tag{30}$$

where p_x, p_y, q_x and q_y are the partial derivatives of p and q with respect to x and y. Typically, the goal is to minimize the combined energy function with the Lagrange multiplier:

$$E_s(x, y) = (I - R)^2 + \lambda(p_x^2 + p_y^2 + q_x^2 + q_y^2) \ . \tag{31}$$

Then the relaxation operation becomes

$$p^n = \bar{p}^{n-1} + \frac{1}{\lambda}(I - R)R_p$$
$$q^n = \bar{q}^{n-1} + \frac{1}{\lambda}(I - R)R_q \ , \tag{32}$$

where $\bar{p}$ and $\bar{q}$ are the local average of p and q, respectively, and R_p and R_q are the derivatives of R with respect to p and q, respectively.

The feature extractor passes the pixel intensity I to the PU's local memory. Another set of parameters stored in the memory is the reflectance map R. The local processor computes $\bar{p}$, $\bar{q}$, and updates p and q using (32). The global processor computes $p^n - p^{n-1}$ and $q^n - q^{n-1}$ to check the convergence of the relaxation.

4.3 K-means clustering

For every feature vector, the distance between the vector and each cluster centroid is computed in the feature vector space, and the feature is grouped into the cluster whose centroid is the closest one to the feature. At the end of each iteration, the cluster centers are updated by simply taking the average of the features in the cluster. The algorithm converges when no feature changes its cluster group.

First, the feature extractor needs to extract reliable features. Gabor multi-resolution features are easily obtained by the extractor. The set of features becomes the parameter for the relaxation and is stored in the local memory. The cluster centroids at each relaxation iteration are stored in the input buffer. The local processor computes the distance of each feature vector from each cluster centroid and determines the closest centroid to the feature vector. The global processor updates the cluster centroids for the new cluster result. It also compares the difference between the old centroids and the new centroids to check the convergence of the relaxation.

4.4 Mumford-Shah segmentation

The segmentation is obtained by minimizing the energy function

$$E = \int \alpha_c (s-g)^2 + \mu_c \|\nabla s\|^2 (1-l) + \nu_c l dx dy \ , \tag{33}$$

where $g(x,y)$ is the measured local statistics of data or the extracted local features for segmentation, $s(x,y)$ is the surface process which measures the global statistics of the particular region, $l(x,y)$ is the line process which represents the boundaries between different regions, ∇ is the gradient operator, and $\|\cdot\|$ is the norm in the feature vector space. If Gabor multi-resolution features are used for the segmentation, the above energy function can be extended to include interaction between the orientation and scale dimension of the Gabor 4D features. For more detail, see [15]. The model is modified as

$$E = \int \alpha (s-g)^2 + \gamma_\sigma \left\| \frac{ds}{d\sigma} \right\|^2 + \gamma_\theta \left\| \frac{ds}{d\theta} \right\|^2 + \mu \|\nabla s\|^2 (1-l) + \nu l dx dy \ , \tag{34}$$

where γ_θ and γ_σ are parameters.

There are two relaxation processes in the model. One for the surface process and the other for the line process. The relaxation for the surface process is

$$s^n = (1-r_s) s^{n-1} + r_s (g + \gamma_\sigma \frac{\delta^2 s}{\delta \sigma^2} + \gamma_\theta \frac{\delta^2 s}{\delta \theta^2} + \nabla(\mu \nabla (1-l))) \ , \tag{35}$$

where r_s is the rate of the descent. Using the mean field method, the relaxation for the line process is [7]:

$$l = \frac{1}{1 + e^{(\nu - \mu |\nabla s|^2)/T}} \tag{36}$$

where T is the temperature, which starts with a high value and cools down as the iteration proceeds.

The local memory holds the feature set. The input buffer contains the intermediate state of the surface and the line processes. The local processor performs the interaction of (35) and (36).

5 Conclusion

The main characteristics of our vision system are the following.

- Various spatial filters are implemented using approximation techniques for faster processing.
- The relaxation network explores the parallelism in the temporal (iteration) domain rather than the spatial domain.

A major portion of the feature extractor has a filter bank structure and can be integrated into VLSI chips. A major portion of the relaxation network is a chain of processing units. The design of the PU is tailored for the nature of local iterative operations.

Future research work includes performance evaluation of the system on various vision tasks, detailed hardware specification of the PU, design of instruction sets for the PU, and VLSI implementations of VFC, HFC, SFC, and the PU.

References

[1] K. Banerjee, P. S. Sastry, K. R. Ramakrishnan, and Y. V. Venkatesh, "An SIMD machine for low-level vision," *Information Sciences,* vol. 44(1), pp. 19-50, 1988.

[2] N. G. Bourbakis and J. S. Mertoguno, "Kydon: An Autonomous, Multi-layer Image-understanding System: Lower Layers," *Engineering Application Artificial Intelligence,* vol. 9(1), pp. 43–52, 1996.

[3] P. J. Burt, "Multiresolution techniques for image representation, analysis, and 'smart' transmission," *Proc. SPIE Conf. on Visual Communication and Image Processing,* pp. 2–15, 1989.

[4] J. Canny, "A computational approach to edge detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 8, pp. 679–698, 1986. J. Canny, "Semantic network array processor and its applications to image understanding," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 9(1), pp. 153–160, 1987.

[5] A. L. Fisher and P. T. Highnam, "The SLAP image computer," in *Parallel Architectures and Algorithms for Image Understanding,* (Ed: V.S.P. Kumar), Academic Press, San Diego, CA, 1991.

[6] W. T. Freeman and E. H. Adelson, "The design and use of steerable filters," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 13(9), pp. 891–906, 1991.

[7] D. Geiger and F. Girosi, "Parallel and deterministic algorithms from MRF's: Surface reconstruction," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 13(5), pp. 401–412, 1991.

[8] T. L. Huntsberger and W. R. Wood, "FLASH, A parallel architecture for computer vision in uncertain environments," in *Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pp. 280–283, 1985.

[9] C. Koch and B. Mathur, "Neuromorphic vision chip," *IEEE Spectrum,* vol. 33(5), pp. 38–46, 1996.

[10] T. Kubota and C. O. Alford, "Computation of orientational filters for real-time computer vision problems I: Implementation and methodology," *Real-Time Imaging Journal,* vol. 1, pp. 261-281, 1995.

[11] T. Kubota and C. O. Alford, "Computation of orientational filters for real-time computer vision problems II: Multi-resolution image decomposition," *Real-Time Imaging Journal,* vol. 2, pp. 91-116, 1996.

[12] T. Kubota and C. O. Alford, "Computation of orientational filters for real-time computer vision problems III: Steerable system and VLSI architecture," to appear in *Real-Time Imaging Journal.*

[13] V. K. P. Kumar, *Parallel Architectures and Algorithms for Image Understanding*, Academic Press, San Diego, CA, 1991.

[14] K. I. laws, "Rapid texture identification," *Proceedings of SPIE,* vol. 238, pp. 376–380, 1980.

[15] T. S. Lee. "A Bayesian framework for understanding texture segmentation in the primary visual cortex," *Vision Res.*, vol. 35(18), pp. 2643–2657, 1995.

[16] M. A. Mahowald and C. Mead, "The Silicon Retina," *Scientific American,* vol. 264(5), pp. 76–82, 1991.

[17] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London,* vol. 207, pp. 187–217, 1980.

[18] P. Perona, "Deformable kernels for early vision," *Proc. IEEE Trans. on Computer Vision and Pattern Recognition,* pp. 222–227, 1991.

[19] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine,* October, 1991.

[20] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, "Shiftable multiscale transforms," *IEEE Trans. on Information Theory,* vol. 38(2), pp. 587–607, 1992.

[21] S. Treitel and J. L. Shanks, "The design of multistage separable planner filters," *IEEE Trans. on Geoscience Electronics,* vol. 9(1), pp. 10–27, 1971.

[22] W. Wang, J. Gu, and T. C. Henderson, "A pipelined architecture for parallel image relaxation operations," *IEEE Trans. on Circuits and Systems,* vol. 34(11), pp. 1375–1384, 1987.

[23] A. B. Watson, "The cortex transforms: Rapid computation of simulated neural images," *Computer Vision, Graphics, and Image Processing,* vol. 39, pp. 311–327, 1987.

[24] R. A. Young, "The gaussian derivative model for spatial vision: I. Retinal mechanism," *Spatial Vision,* vol. 2(4), pp. 273–293, 1987.