Semiannual Technical Summary

Multi-Dimensional Signal Processing Research Program

30 September 1982

Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-80-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

MULTI-DIMENSIONAL SIGNAL PROCESSING RESEARCH PROGRAM

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ROME AIR DEVELOPMENT CENTER

1 APRIL — 30 SEPTEMBER 1982

ISSUED 14 FEBRUARY 1983

LEXINGTON MASSACHUSETTS

The originals of the color figures appearing in this document are color photographic prints. Recipients of this document desiring a similar quality reproduction may obtain the full set at the cost of $20.00, by submitting a check payable to Massachusetts Institute of Technology to:

Lincoln Laboratory
Massachusetts Institute of Technology
Publications Group
P.O. Box 73
Lexington, MA 02173-0073

ABSTRACT

This Semiannual Technical Summary covers the period 1 April 1982 through 30 September 1982. It describes the significant results of the Lincoln Laboratory Multi-Dimensional Signal Processing Research Program sponsored by the Rome Air Development Center, in the areas of target detection, adaptive contrast enhancement, and image processing architectures.

TABLE OF CONTENTS

1. INTRODUCTION AND SUMMARY

The Lincoln Laboratory Multi-Dimensional Signal Processing Research Program was initiated in FY80 as a research effort directed toward the development and understanding of the theory of digital processing of multi-dimensional signals and its application to real-time image processing and analysis. A specific long-range application is the automated processing of aerial reconnaissance imagery. Current research projects that support this long-range goal are image modeling for segmentation, classification and target detection; techniques for adaptive contrast enhancement; and multi-processor architecture to implement image processing algorithms.

This Semiannual Technical Summary discusses results in three areas. In Section 2 we present a detailed, technical description of our efforts in target detection. The problem is formulated as a significance test to determine whether a small region contains one or more pixels (picture elements) that do not match the measured statistical properties of the background. Examples are shown in which the resulting detection algorithms are applied to aerial reconnaissance photographs.

Section 3 contains program documentation information for the adaptive contrast enhancement software delivered to RADC/IRRE earlier this year. It includes a description of all the program modes, parameters, and user interaction as well as the typescript from a sample run.

In Section 4 we discuss our latest thoughts on a multi-processor architecture for image processing applications. Attention is focused on the architectural requirements for a single nodal processor. (As currently planned, the multi-processor will consist of 16 such nodal processors.) The architectural study is by no means complete, but we have investigated several architectural principles that appear to support the goals of high computational throughput and ease of programming. In FY83, we plan to continue in this vein, refining the architecture and developing an instruction set for the nodal processors.

2. TARGET DETECTION BY TWO-DIMENSIONAL LINEAR PREDICTION

2.1 Introduction

This research relates to the problem of detecting targets (i.e., anomolous areas) in aerial photographs. We define the target detection problem as the detection of man-made objects in a textured background (e.g., trees, grass, fields).

In a previous Semiannual Technical Summary [1], we developed a target detection algorithm based on the coefficient change function (CCF). We looked for changes in the parameters of a continuously varying autoregressive model of aerial photographs. We hypothesized, however, that the two-dimensional (2-D) prediction error may provide a significant improvement over this CCF algorithm.

In this section, we develop such an algorithm and provide a rigorous theoretical basis for it in terms of significance testing [2]. Our detection algorithm is derived from the fact that significance testing can be expressed in terms of the error residuals of 2-D linear prediction. We first develop the algorithm under a stationary Gaussian assumption and then proceed to generalize it for the case where the background is nonstationary. This more general test involves determining the error residuals from an optimal space-varying predictor. The error residuals are combined over a small area, suitably normalized and, finally, compared to a threshold. Since a causal 2-D prediction filter is associated with our significance test, we can interpret this as modeling the background by a 2-D causal space-varying autoregressive random process. The parameters of this autoregressive model, therefore, need to be estimated from the background.

Our detection algorithm (as we shall demonstrate with examples) has been applied successfully to both synthetic images and actual reconnaissance photographs obtained from the Rome Air Force Development Center (RADC).

2.2 Significance Testing

The problem of target detection in images is finding small areas in an image whose statistical properties do not match those of the surrounding area or background. Since the target statistics are generally unknown (it is desired to detect broad classes of targets), and the background statistics may be known or can be estimated, the problem is inherently different from a classic detection or classification problem. Essentially, the only question that can be asked is "Does a set of pixels under examination represent background or does it represent something else?" The area of statistics that addresses such questions is called significance testing [2]. The basic idea is illustrated in Figure 2-1. A measurement is made of some random phenomenon characterized by probability density p(x). Critical regions (i.e., regions of low probability) are chosen corresponding to unlikely events. If measurements fall in the critical region, we reject the hypothesis that the measurements really belong to the density p(x).

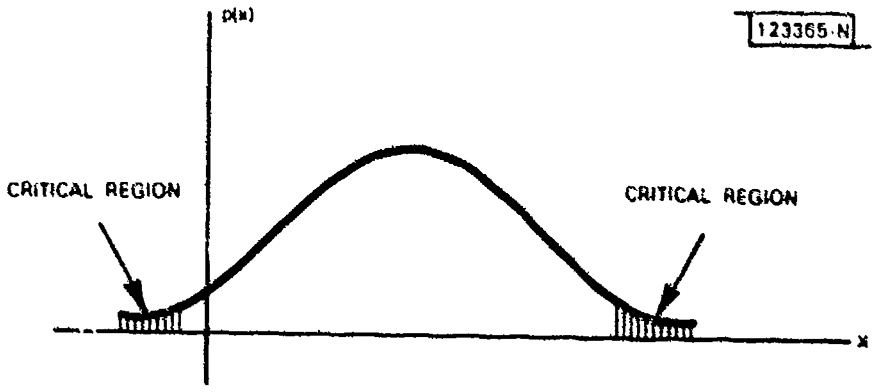


Fig. 2-1. Illustration of significance testing.

In our case, the probability density is that of the background. If a set of measurements falls in the critical region, we reject the hypothesis that the pixels form a part of the background; that is, we decide that they represent (at least partly) a target. The significance level of the test is determined by the probability of events in the critical region. For example, if this probability is .05, then the significance test is at the 5% level.

The reasoning involved in the significance test may seem indirect. However, since only the background has known or estimable statistics, this is a logical line of reasoning to take.

2.2.1 Formulation

Given an image and small region S at (n,m), let the image points in S (see Figure 2-2) be denoted by

$$\underline{x} = (x_1, x_2, \ldots, x_N)^T \tag{1}$$

We want to decide whether the points in S corresponds to a homogeneous random field with probability density $p(\underline{x})$ (i.e., S contains just background) or whether S contains something other than the homogeneous random field (object possibly present). We want to do this for all (n,m).

Let the background correspond to a stationary Gaussian random process with mean $\underline{m} = E[\underline{x}]$ and covariance $K = E[(\underline{x}-\underline{m})(\underline{x}-\underline{m})^T]$. Then

$$p(\underline{x}) = \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}(\underline{x}-\underline{m})^T K^{-1}(\underline{x}-\underline{m})\right] \tag{2}$$

We shall plot the probability density function and determine a <u>critical region</u> of small probability (α) which is the level of significance (see Figure 2-3). Let H_0 be the hypothesis that the points in S correspond to the homogeneous random field. Then we accept H_0 (decide "background only") if the points in S do <u>not</u> fall in the critical region. Otherwise, we reject H_0 (decide "more than just background"). This is repeated for every (n,m).

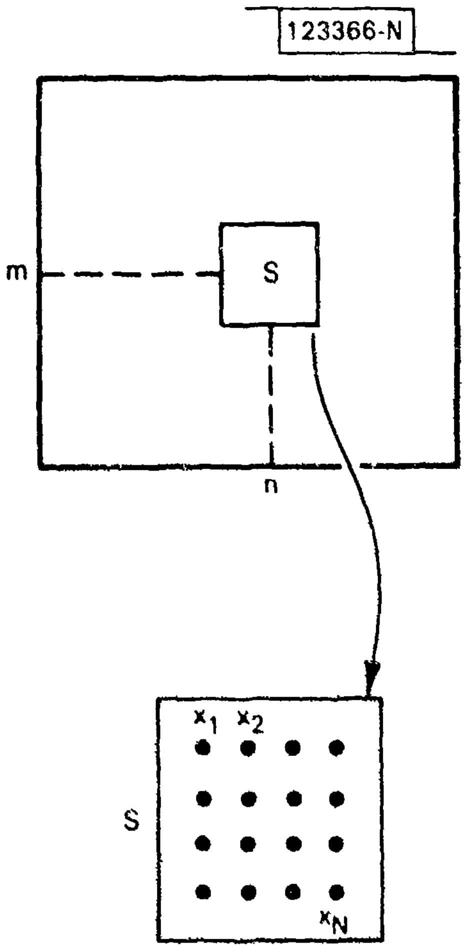


Fig. 2-2. Image and small region S.

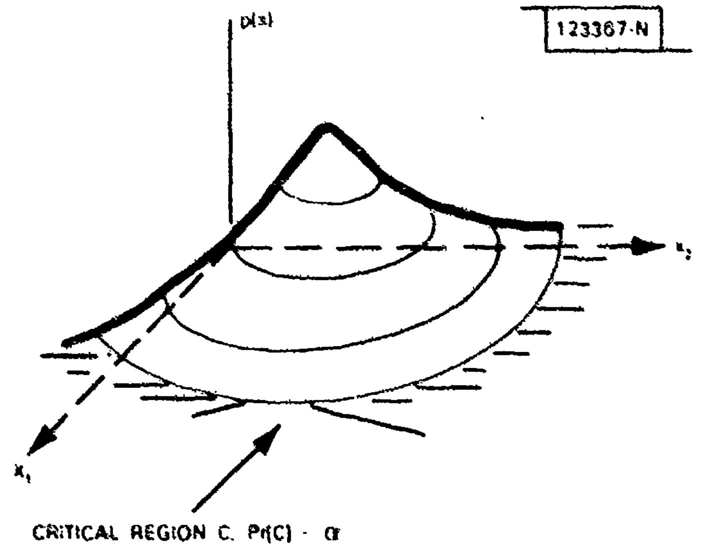


Fig. 2-3. Gaussian probability density with critical region C.

A critical region C can be defined by $p(\underline{x})<\lambda$. The relation between λ and the level of significance α is

$$\begin{aligned} &\int p(\underline{x})d\underline{x} = \alpha \\ &p(\underline{x})<\lambda \end{aligned} \tag{3}$$

Taking the logarithm of $p(\underline{x})<\lambda$, our significance test becomes:

$$(\underline{x}-\underline{m})^T K^{-1}(\underline{x}-\underline{m}) > -\ell n(2\pi)^N|K|-2\ell n\lambda \tag{4}$$

Suppose that we wish to maintain a constant level of significance regardless of the background statistics (i.e., regardless of the covariance K). This is equivalent to enforcing a constant false alarm rate. To do this we will find the particular form that λ must take as we vary K. We investigate the 1-D case; the N-D case is more complicated and is given in the appendix.

Let the density function of the zero-mean random variable x take the form

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-x^2/2\sigma^2\right] \tag{5}$$

Then the boundaries, $\pm x_0$, of our critical region are given by $p(\pm x_0) = \lambda$. From Equation 5 it follows that

$$x_o = (-2\sigma^2 \ell n(\sqrt{2\pi}\sigma\lambda))^{1/2} \tag{6}$$

The level of significance in this case is given by

$$2\int_{x_o}^{\infty} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-x^2/2\sigma^2\right]dx = \alpha \tag{7}$$

Now, let $u=x/\sigma$, so that

$$2 \int_{x_o/\sigma}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\,[-u^2/2]du = \alpha \qquad (8)$$

For a constant α, we must have (from Equations 6 and 8):

$$x_o/\sigma = (-2\ell n(\sqrt{2\pi}\,\sigma\lambda))^{1/2} = C \qquad (9)$$

where C is a constant. Therefore, λ must vary as

$$\lambda = \frac{C}{\sqrt{2\pi}\,\sigma} \qquad (10)$$

That is, λ must vary <u>inversely</u> as the standard deviation to guarantee a level of significance that does not vary with the background statistics.

Evaluating Equation 4 for the 1-D case where $K=|K|=\sigma^2$,

$$\begin{aligned}(x-m)^2/\sigma^2 &= -\ell n(2\pi\sigma^2)-2\ell n\lambda \\ &> -\ell n(2\pi\sigma^2 C^2/2\pi\sigma^2) \\ &> -\ell n C^2 = C' \qquad (11)\end{aligned}$$

where C' is a constant threshold. Thus, in the 1-D case, our significance test involves comparing the squared random variable (with mean removed and appropriately normalized) to a constant threshold.

More generally, we can show that for a constant level of significance, in the N-D case (see appendix), λ must vary as

$$\lambda = C/(2\pi)^{N/2}\,|K|^{1/2} \qquad (12)$$

Thus, Equation 4 becomes:

$$(\underline{x}-\underline{m})^T K^{-1}(\underline{x}-\underline{m}) > C' \qquad (13)$$

2.2.2 Relationship of Significance Testing with Linear Prediction

We will now show that the significance test of the previous section can be expressed in terms of the error residuals by optimally predicting each sample of our small region S by a linear combination of its "past" samples ("past" is defined below).

Our connection can be made because the background covariance matrix K can be uniquely factored in terms of upper and lower triangular and a diagonal matrix [3]. Let

$$K = LDL^T \tag{14}$$

where L is lower triangular with ones along its diagonal, where D is a diagonal matrix, and where T denotes transpose. Substituting Equation 14 into 13, we have

$$\begin{aligned}(\underline{x}-\underline{m})^T K^{-1}(\underline{x}-\underline{m}) &= (\underline{x}-\underline{m})^T (LDL^T)^{-1}(\underline{x}-\underline{m}) \\ &= (\underline{x}-\underline{m})^T L^{-1^T} D^{-1} L^{-1}(\underline{x}-\underline{m}) \\ &= \underline{e}^T D^{-1} \underline{e} > C' \end{aligned} \tag{15a}$$

where

$$\underline{e} = L^{-1}(\underline{x}-\underline{m}) \tag{15b}$$

It is straightforward to show that since L is lower triangular with a unit diagonal, L^{-1} has the same property, and thus Equation 15b represents a causal transformation of the vector $\underline{x}$ [3]. Furthermore, it can be shown [3] that each row of L^{-1} represents the coefficients required in optimally predicting an element of $\underline{x}$, x_n from its previous values, i.e., from a linear combination of $x_{n-1}\ldots x_1$. Since the covariance matrix K is that of the background, the coefficients of L^{-1} correspond to optimal prediction of the background and not target areas. This prediction concept is illustrated in

Figure 2-4, where we are predicting the middle pixel of a 5 x 5 region S from 12 of its neighboring values. An element e_k of $\underline{e}$ in Equation 15b is given by

$$e_k = (-\underline{a}_k^T, 1)\left(\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_k \end{bmatrix} - \begin{bmatrix} m_1 \\ m_2 \\ . \\ . \\ . \\ m_k \end{bmatrix}\right) \tag{16}$$

where $\underline{a}_k$ is the vector of coefficients for optimally predicting x_k from the values $x_1, \ldots, x_{k-1}$. The diagonal elements of D are given by $\sigma_k^2 = \text{var}(e_k)$. Note that the e_k's are uncorrelated, i.e., they form a white process [3] when the pixels being predicted (and doing the prediction) are background pixels.

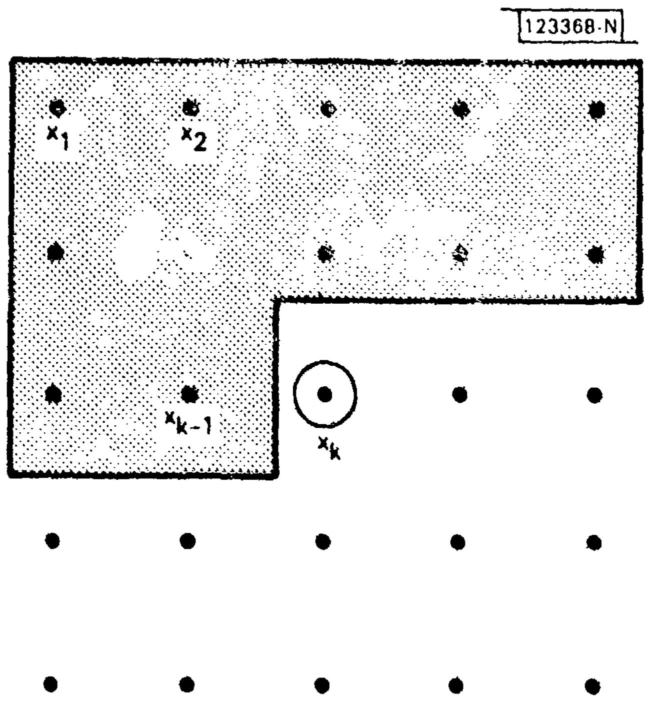


Fig. 2-4. Prediction of x_k from its previous values $x_1, \ldots x_{k-1}$.

As illustrated in Figure 2-4, the mask required in predicting each element x_k of S is a growing nonsymmetric half-plane mask [4]. Suppose that the background happens to be governed by such a pixel relationship, i.e., we shall assume that the background random field follows an autoregressive process of the form:

$$x(n,m) = \sum_{(j,k)>(0,0)} \sum a(j,k)x(n-j,m-k) + w(n,m) \qquad (17)$$

where $w(n,m)$ is white Gaussian noise and where $(k_1,\ell_1) < (k_2,\ell_2)$ denotes that (k_1,ℓ_1) is in the "past" of (k_2,ℓ_2). For any point (s,t), we define the <u>past</u> to be the set of points [5]:

$$\{(k,\ell)\,|\,k=s,\ \ell<t;\ k<s,\ -\infty< \ell<\infty\} \qquad (18)$$

With a 90° rotation of coordinates, the values of S used in predicting x_k can then be those elements in S that are in the now formally defined past of x_k. For our purposes, we shall refer to the model of Equation 17 as a causal model, i.e., each $x(n,m)$ is a function of its past. Note that this notion of causality is tied to the shape of the nonsymmetric prediction mask of Figure 2-4.

Let us suppose that the background follows an autoregressive model of finite order. For example, let the background follow a first-quadrant causal autoregressive model. Then, except for certain boundary elements, each element of e_k in Equation 16 consists of the error in predicting each pixel of S from its first quadrant neighbors (in a 90° rotated coordinate system). This is illustrated in Figure 2-5 where, except for the L-shaped boundary, the fixed-order prediction error equals e_k. That is, the remaining coefficients of the growing nonsymmetric half-plane mask of Figure 2-4 are zero and thus do not affect the prediction. Generally, the background random field can be modeled by a causal autoregressive process of sufficiently large order [4], and this model corresponds to the optimal linear prediction from past values of the 2-D random field.

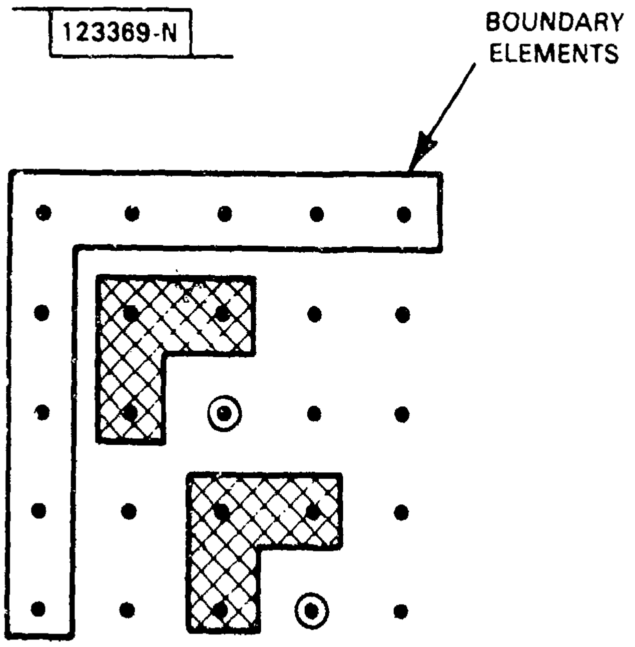


Fig. 2-5. Representation of first-quadrant fixed-order predictor.

Returning to our significance test, from Equation 15a,

$$\underline{e}^T \underline{D}^{-1} \underline{e} = \sum_{k=1}^{N} \frac{e_k^2}{\sigma_k^2} > \text{constant} \tag{19}$$

where σ_k^2 is the prediction error variance associated with predicting a background value x_k.

Although e_k^2 and σ_k^2 arise from a growing predictor using only the samples within region S, they can be approximated by the residuals and variances associated with a fixed-order linear predictor where samples both inside and outside S are used in the prediction. In this formulation, since the background is assumed stationary, σ_k^2 in Equation 19 is constant over the summation index. Thus, an approximate significance test can be written as

$$\frac{1}{\sigma^2} \sum_{k=1}^{N} \tilde{e}_k^{\,2} > \text{constant} \tag{20}$$

where $\tilde{e}_k$ is the prediction error based on a fixed-order predictor and where $\sigma^2 = \sigma_k^2$.

2.3 The Nonstationary Problem

The above detection algorithm depends on knowledge of stationary background statistics. The significance test described in the previous section uses a stationary Gaussian assumption where the covariance matrix K is the same at each spatial coordinate (n,m). In reality, however, the background statistics of an image are changing and the matrix K in Equation 2 is generally non-Toeplitz and differs at each (n,m). In this section we will investigate the significance test when the data are nonstationary. We shall see that the significance test changes little from what we found previously.

Let us consider the small region S extracted from a 1-D sequence x(n) illustrated in Figure 2-6. We investigate the 1-D case for simplicity--the 2-D case follows similarly. The time-dependence of the statistics of the sequence is represented through the time-varying covariance matrix K(n). The significance test associated with the region S is then given by

$$p(\underline{x}) = \frac{1}{(2\pi)^{N/2}\,|K(n)|^{1/2}} \exp\left[-\frac{1}{2}(\underline{x} - \underline{m}(n))^T K(n)(\underline{x} - \underline{m}(n))\right] > \lambda \tag{21}$$

where we have further indicated the time-dependence of the data through the time-varying mean vector $\underline{m}(n)$.

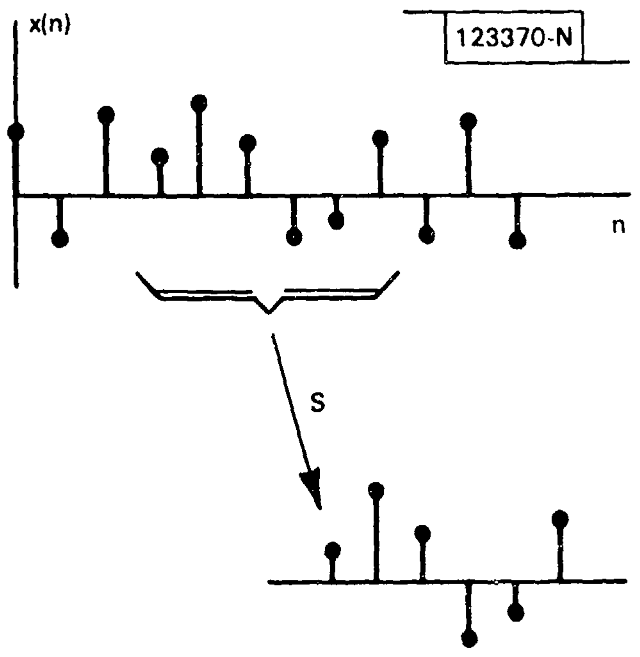


Fig. 2-6. One-dimensional signal with small region S.

Taking the logarithm of Equation 21, we have

$$(\underline{x}-\underline{m}(n))^T K^{-1}(n) (\underline{x}-\underline{m}(n)) > -\ln[(2\pi)^N|K(n)|] - 2\ln \lambda \tag{22}$$

We want to maintain a constant level of significance by appropriately varying λ. We can show that a constant level of significance can be maintained such that the significance test becomes:

$$(\underline{x} - \underline{m}(n))^T K^{-1}(n) (\underline{x} - \underline{m}(n)) > \text{constant} \tag{23}$$

Since K(n) is symmetric (but note, not Toeplitz), we can again perform an LDL^T decomposition, which now becomes a function of the time variable n:

$$K(n) = L(n)D(n)L^T(n) \tag{24}$$

The significance test becomes

$$(\underline{x}-\underline{m}(n))^T L^{-1}(n) D^{-1}(n) (L^{-1}(n))^T (\underline{x}-\underline{m}(n))$$

$$= \underline{e}^T(n) D^{-1}(n) \underline{e}(n) > \text{constant} \tag{25a}$$

where

$$\underline{e}(n) = L^{-1}(n)(\underline{x}-\underline{m}(n)) \tag{25b}$$

As before, it is possible to show that the transformation of Equation 25b corresponds to successive orders of linear prediction where the diagonal elements of D(n) are the prediction error variances. However, now the prediction coefficients embedded within $\underline{e}(n)$ are time-varying as well as changing with the growing order. This implies that even when the background can be modeled by a nonstationary autoregressive process of finite order, the prediction coefficients associated with our small region S are generally always changing.

Let us investigate this more carefully. For example, consider a Pth-order time-varying predictor, with prediction error of the form:

$$e(n) = x(n) - \sum_{k=1}^{P} a(n;k)x(n-k) \tag{26}$$

Then using the projection theorem, we can solve for the a(n;k)'s (at each time sample), which minimize $E[e^2(n)]$:

$$E[e(n)x(n-\ell)] = 0 \qquad \ell = 1,2,\ldots P \tag{27}$$

Substituting Equation 26 into 27,

$$E[x(n)x(n-\ell)] = \sum_{k=1}^{P} a(n;k)E[x(n-k)x(n-\ell)] \tag{28}$$

or

$$r(n;k=0,\ell) = \sum_{k=1}^{P} a(n;k)r(n;k,\ell) \tag{29a}$$

where

$$r(n;k,\ell) \overset{\Delta}{=} E[x(n-k)x(n-\ell)] \tag{29b}$$

Note that $r(n_o;k,\ell)$ represents the correlation of values of $x(n)$ for $n<n_o$ since k and ℓ are constrained to be non-negative. Thus, $r(n_o;k,\ell)$ reflects only the past of the time-varying correlation structure of $x(n)$.

Noting that the prediction error variance is given as

$$\sigma^2(n) = r(n;0,0) - \sum_{k=1}^{P} a(n;k)r(n;k,0) \tag{30}$$

we can combine Equations 29 and 30 to obtain the time-varying normal equations:

$$R(n)\underline{a}(n) = \underline{\sigma}^2(n) \tag{31a}$$

$$|\xleftarrow{\quad} P + 1 \xrightarrow{\quad\quad}|$$

where

$$R(n) = \begin{bmatrix} & \cdot & \\ & \cdot & \\ & \cdot & \\ \ldots & r(n;k,\ell) & \ldots \\ & \cdot & \\ & \cdot & \\ & \cdot & \end{bmatrix} \quad \Big\updownarrow P + 1 \qquad (31b)$$

and

$$\underline{a}(n) = \begin{bmatrix} 1 \\ -a(n;1) \\ -a(n;2) \\ \cdot \\ \cdot \\ \cdot \\ -a(n;P) \end{bmatrix} \qquad \underline{\sigma}^2(n) = \begin{bmatrix} \sigma^2(n) \\ 0 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \qquad (31c)$$

Refer back to the small region S that is illustrated in Figure 2-6 for the 1-D case. We want to show that $\underline{e}(n)$ in Equation 25 does in fact correspond to the prediction error from successively growing time-varying predictors. We first write the sequence of normal equations associated with predicting each element of S (we assume our region S runs over the interval $0 \le n \le N$):

$$R(0)\hat{\underline{a}}(0) = \underline{\sigma}^2(0)$$

$$R(1)\hat{\underline{a}}(1) = \underline{\sigma}^2(1)$$

$$\vdots$$

$$R(N)\hat{\underline{a}}(N) = \underline{\sigma}^2(N) \tag{32a}$$

where "^" refers to a growing predictor, and the index n refers not only to the location of the sample being predicted, but also to the order of the predictor. However, if the process of concern follows a Pth-order autoregressive model, $\hat{\underline{a}}(n)$ for $n>P$ will be of order P; i.e., $\hat{a}(n;u)=0$ for $k>P$.

Note that R(0), R(1)... are matrices growing in size, but that R(n) is a lower right-hand corner submatrix of R(N+1)--in spite of the fact that these matrices are non-Toeplitz. Consequently, we can express the sequence of normal Equations 32a as

$$\begin{bmatrix} \overset{\rightarrow}{\underline{a}}(0) & & & & 0 \\ & \overset{\rightarrow}{\underline{a}}(1) & & & \\ & & \overset{\rightarrow}{\underline{a}}(2) & & \\ & & & \ddots & \\ & & & & \overset{\rightarrow}{\underline{a}}(N) \end{bmatrix} R(N) \begin{bmatrix} \overset{\rightarrow}{\underline{a}}(N) & & & \\ & \overset{\rightarrow}{\underline{a}}(N-1) & & \\ & & \overset{\rightarrow}{\underline{a}}(N-2) & \\ 0 & & & \\ & & & \overset{\rightarrow}{\underline{a}}(0) \end{bmatrix} = \begin{bmatrix} \sigma^2(0) & & 0 \\ & \sigma^2(1) & \\ 0 & & \\ & & \sigma^2(N) \end{bmatrix} \tag{32b}$$

where $\overset{\rightarrow}{\underline{a}}(n)$ represents the reversal on $\hat{\underline{a}}(n)$.

Finally, note that (when x(n) is zero-mean) R(n) = K(n) in Equation 32b, so that

$$L^{-1} = \begin{bmatrix} \overset{\rightarrow}{\underline{a}}(0) & & \\ & \overset{\rightarrow}{\underline{a}}(1) & 0 \\ & & \overset{\rightarrow}{\underline{a}}(N) \end{bmatrix} \tag{33a}$$

and

$$D = \begin{bmatrix} \sigma^2(0) & & & & & \\ & \sigma^2(1) & & & 0 & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ 0 & & & & & \sigma^2(N) \end{bmatrix} \tag{33b}$$

Thus, $\underline{e}(n)$ does indeed correspond to successive orders of time-varying (growing) predictors, provided that the process being predicted has zero mean.

In the stationary case, we can approximate $\underline{e}(n)$ by time-varying finite-order predictors. As before, when the background process follows an autoregressive process of this finite order, only values of $\underline{e}(n)$ near the boundary of the region S will be inexact. Our approximate significance test based on Equation 25 becomes:

$$\begin{aligned} \underline{e}^T(n)D^{-1}(n)\underline{e}(n) &= \tilde{\underline{e}}^T(n)\tilde{D}^{-1}(n)\tilde{\underline{e}}(n) \\ &= \sum_{k=1}^{N} \tilde{e}^2(k)/\tilde{\sigma}^2(k) \\ &> \text{constant} \end{aligned} \tag{34}$$

where $\tilde{e}^e(k)$ and $\tilde{\sigma}^2(k)$ correspond to fixed-order, but time-varying predictors (note that the index k represents our time index over the region S).

Our approximation here may also have certain computational advantages. For example, we do not need to recompute predictor errors (corresponding to different orders) for overlapping regions S.

2.4 Adaptive Estimation and Prediction

The significance test of the previous section depends on knowledge of the background statistics. We need to know or estimate the coefficients of the assumed time space-varying causal autoregressive model. However, in attempting this estimation, we encounter the infamous uncertainty principle. That is, to obtain a reliable estimate of R(n) in Equation 31 (which is needed to estimate a(n,k)), we require stationarity over a "sufficiently large" window size. On the other hand, we assume statistics are generally changing everywhere in space.

To side-step this problem, we assume that the data are stationary over the estimation window, $w_e(n,m)$. The location of the 2-D estimation window that slides over our data will be designated by the center index (n_0,m_0) as illustrated in Figure 2-7. The model parameters associated with this window are defined as (n_0,m_0): $a(n_0,m_0;j,k)$.

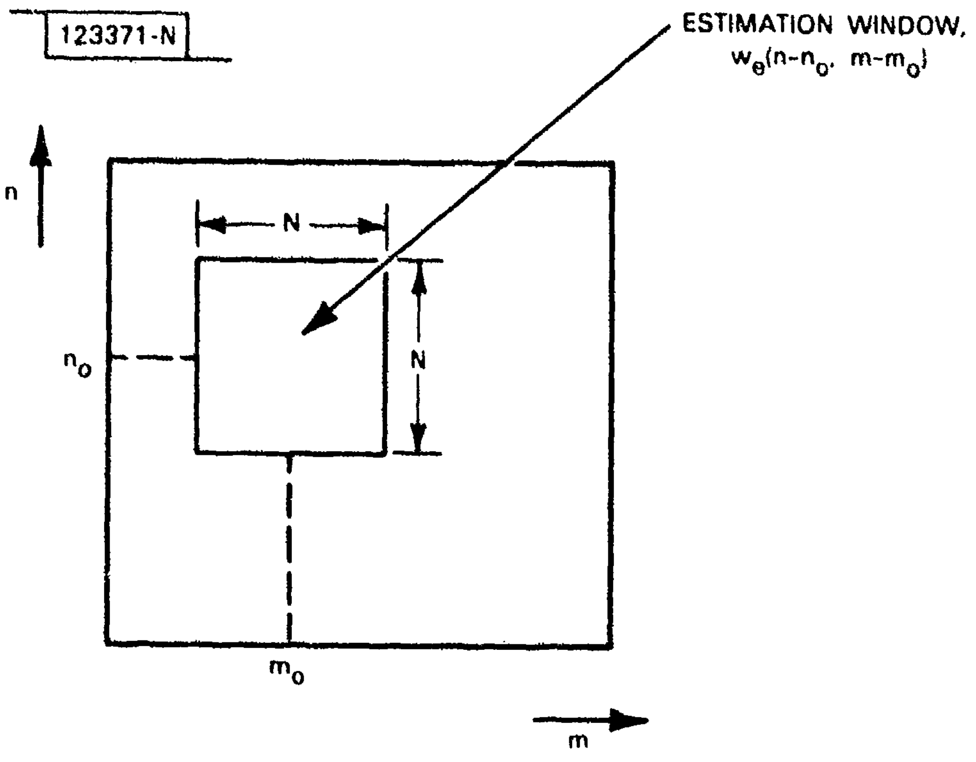


Fig. 2-7. Representation of estimation window.

The estimation procedure we shall use is the covariance method of linear prediction. The prediction error associated with the estimation window at spatial coordinates (n_0,m_0) is the error $e(n_0,m_0)$ in predicting the value $x(n_0,m_0)$ from its causal neighbors. Finally, the prediction error variance $\sigma_2(n_0,m_0)$ is given by the average squared prediction error under the estimation window at (n_0,m_0).

For each pixel location (n_0,m_0), we wish to estimate the set of model parameters $a(n_0,m_0;k,\ell)$ that vary in space. To do this we assume that $x(n,m)$ is a Gaussian 2-D random field stationary over each $w_e(n-n_0,m-m_0)$, and follows the model in Equation 17 under each estimation window. Therefore, for this section, we drop the space dependence and work with the model given by Equation 17.

We shall assume that the prediction coefficients $a(j,k)$ fall within a (PxQ) first-quadrant plane mask. For simplicity, we limit our derivations to this class of prediction masks, although it is clearly applicable to more general mask shapes, such as the nonsymmetric half-plane mask. The objective is to estimate from $x(n,m)$ the model parameters $a(j,k)$ for $j = 0,1...,P$ and $k = 0,1...,Q$, with $j = k \neq 0$. Further, assume we have $x(n,m)$ for $(n,m) \varepsilon [-P+n_1,n_2]x[-Q+m_1,m_2]$ (see Figure 2-8). We then define the error $e(n,m)$ over the region I, given by $I = [n_1,n_2]x[m_1,m_2]$, as

$$e(n,m) = x(n,m) - \sum_{\substack{j=0 \\ (j,k)\neq(0,0)}}^{P} \sum_{k=0}^{Q} a(j,k)x(n-j,m-k) \quad (n,m)\varepsilon I \qquad (35)$$

Our goal becomes to minimize the sum of the squared errors given by

$$E[n_0,m_0] = \sum_{n=n_1}^{n_2} \sum_{m=m_1}^{m_2} e^2(n,m) \qquad (36)$$

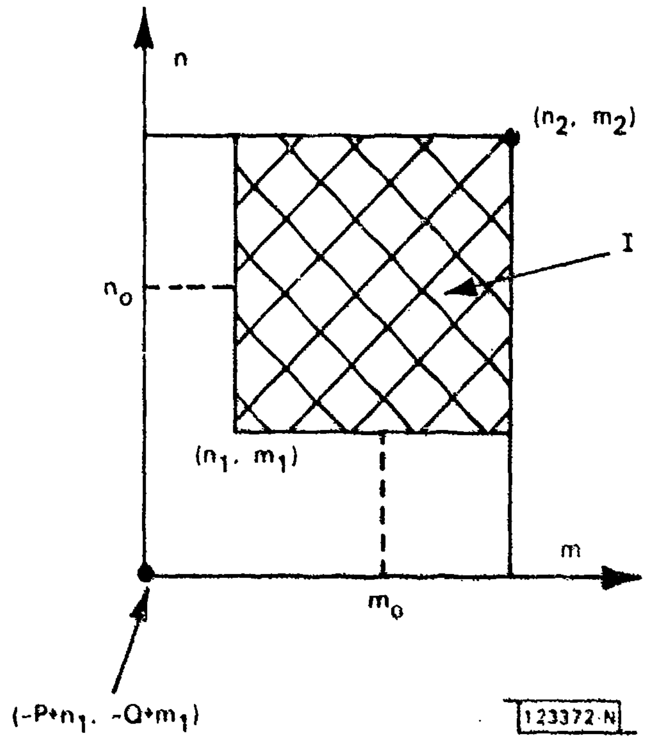


Fig. 2-8. Known data blocks used in 2-D least squares.

The approach we take is to transform the 2-D problem to a 1-D problem so that a 1-D least squares solution is applicable. Note, however, that we will still have solved the 2-D least squares problem. We wish to transform Equation 36 into a 1-D error expression. To accomplish this transformation, we define the vectors $a[n_0,m_0]$ and σ by

$$a[n_0,m_0] = \left[\begin{array}{c} a(0,1) \\ a(0,2) \\ \vdots \\ a(0,Q) \\ \hline a(1,0) \\ a(1,1) \\ \vdots \\ a(1,Q) \\ \hline \vdots \\ \hline a(P,0) \\ a(P,1) \\ \vdots \\ a(P,Q) \end{array}\right] \Bigg\updownarrow (PQ-1) \qquad \sigma = \left[\begin{array}{c} s(n_1,m_1) \\ s(n_1,m_1+1) \\ \vdots \\ s(n_1,m_2) \\ \hline s(n_1+1,m_1) \\ s(n_1+1,m_1+1) \\ \vdots \\ s(n_1+1,m_2) \\ \hline \vdots \\ \hline s(n_2,m_1) \\ s(n_2,m_1+1) \\ \vdots \\ s(n_2,m_2) \end{array}\right] \Bigg\updownarrow (PQ-1) \qquad (37)$$

$$S = \begin{array}{c} \xleftrightarrow{(PQ-1)} \\ \left[\begin{array}{c} \text{——} A_0 \text{——} \\ \text{——} A_1 \text{——} \\ \cdot \\ \cdot \\ \cdot \\ \text{——} A_{M-1} \text{——} \end{array}\right] \end{array} \updownarrow M^2 \tag{38a}$$

where

$$A_j = \begin{array}{c} \xleftrightarrow{\qquad\qquad\qquad\qquad FQ-1 \qquad\qquad\qquad\qquad} \\ \left[\begin{array}{l} [s(n_1+j-0,m_1-1)..s(n_1+j-0,m_1+Q)] \quad ...[s(n_j+j-P,m_1-0)..s(n_1+j-P,m_1-Q)] \\ \\ [s(n_1+j-0,m_i+1-1)..s(n_1+j-0,m_1+1+Q)]...[s(n_1+j-P,m_1+1-0)..s(n_1+j-P,m_1+1-Q)] \\ \\ [s(n_1+j-0,m_2-1)..s(n_1+j-0,m_2-Q)] \quad ...[s(n_1+j-P,m_2-0)..s(n_1+j-P,m_2-Q)] \end{array}\right] \end{array} \tag{38b}$$

and where we have assumed the data segment I to be of extent M x M. Note that σ is a vector consisting of the concatenation of the rows of $x(n,m)$ over S; $a[n_0,m_0]$ is a vector consisting of the concatenation of the rows of $a(j,k)$ for $(j,k)\varepsilon[0,P]x[0,Q]$ with $(j,k) \neq (0,0)$; and S is a matrix that consists of the concatenation of rows of various subsequences of the known $x(n,m)$ required in predicting each value of $x(n,m)$ over I.

Therefore, we can write Equation 36 as:

$$E[n_1,m_1] = \sum_{n=n_1}^{n_2} \sum_{m=m_1}^{m_2} e^2(n,m)$$

$$= (Sa[n_0,m_0] - \sigma)^T(Sa[n_0,m_0] - \sigma) \tag{39}$$

We then write the solution [1] to minimizing Equation 39 with respect to $a[n_0,m_0]$ as:

$$a[n_0,m_0] = R^{-1}S^T\sigma \tag{40a}$$

where

$$R = S^TS \tag{40b}$$

Note that the matrix R is of extent (PQ-1)x(PQ-1) and difficulty in its inversion is dependent on the model order, not on the size of the known block of data.

Since R is generally not Toepiitz, its inversion will require on the order of $(PQ)^3$ operations. The computation of R^{-1} can probably be reduced by considering its block-like structure resulting from the conversion of a 2-D problem to a 1-D problem. Thus, assuming $P,Q<<N$, the bulk of the computation is embedded within forming $R = S^TS$, which requires on the order of N^4 operations.

This estimation is then carried out at each pixel. An alternative to this direct estimation is to accomplish the estimation recursively. However, this may be a realistic alternative only when the estimation window size is less than the model order [1]; i.e., the matrix required to be inverted at each pixel is on the order of MxM. We are currently investigating methods to reduce this computation.

In either case, we obtain a parameter set at each pixel which represents an estimate of the model parameters of the changing background, required in our prediction procedure. Finally, it is straightforward to show from Equations 39 and 40 that the estimate of the prediction error variance given by the average squared prediction error under each estimation window can be expressed by

$$\tilde{\sigma}^2(n_0,m_0) = (\sigma^T\sigma - a[n_0,m_0]^T(S^TS)a[n_0,m_0])N^{-2} \tag{41}$$

2.5 The Detection Algorithm

We can now merge the results of the previous sections to form our target detection algorithm. From our coefficient estimates in Equation 40, we can compute the prediction error function $\tilde{e}^2(n,m)$ based on a fixed-order, but time-varying prediction model. Then with $\tilde{\sigma}^2(n_0,m_0)$ in Equation 41, we can compute the 2-D version of the approximate significance test of Equation 34:

$$\sum_{k,\ell \in S}\sum \tilde{e}^2(k,\ell)/\tilde{\sigma}^2(k,\ell) > \text{constant} \tag{42}$$

where we can think of the indices k and ℓ as running over different regions S.

Equivalently, we can consider generating the statistics in Equation 42 at each spatial location (n,m) of an image by convolving an N x N smoothing window $w_s(n,m)$ with the normalized prediction error to create a new smoothed function $E_s(n,m)$:

$$E_s(n,m) = a(n,m)**w_s(n,m) \tag{43a}$$

where

$$q(n,m) = \tilde{e}^2(n,m)/\tilde{\sigma}^2(n,m) \tag{43b}$$

In the estimation of the model parameters, the estimation window $w_e(n,m)$ should be small enough to preserve approximate stationary, but large enough to obtain a reliable estimate of the required correlation coefficients. The estimation window must also be large enough so that anomalies (i.e., targets) do not badly corrupt the correlation estimates.

The smoothing window should be small enough so that small-extent targets are not overwhelmed by background in the significance test. However, it should also be large enough so boundary effects in our finite-order model assumption do not play a significant role.

The overall detection algorithm based on the approximate significance test is illustrated in Figure 2-9. The first operation subtracts an estimate of the local mean of x(n,m) (recall that our significance test requires a zero-mean random field), which is computed by averaging x(n,m) under $W_e(n,m)$. Under the estimation window, a local covariance matrix R(n,m) as defined in Equation 40b is computed. R(n,m) is then used to find a(n,m) and $\sigma_2(n,m)$, which are required to compute the normalized prediction error, q(n,m). Finally, q(n,m) is convolved with the smoothing window $w_s(n,m)$ and compared to a threshold.

2.6 Examples

In this section, we present nine examples based on the detection algorithm developed in the previous sections. Throughout this section, the estimation window w_e (n,m) is of size 10×10 pixels, which we assume is sufficiently larger than the size of most targets. This assumption can be justified through our empirical observation that, in most cases, the CCF [14] of our processed images is relatively flat; i.e., the targets' presence appears not to adversely affect estimation of background statistics. We also assume that a 10×10 window is large enough to obtain a good estimate of the correlation coefficients required in estimating a[n,m] and $\sigma^2(n,m)$, but also small enough to maintain approximate stationarity. Of course, this assumption breaks down at region boundaries.

In the first examples, we consider computer-generated 1-D and 2-D signals determined by exciting all-pole filters with white noise. We then analyze progressively more complicated real images that were obtained from the RADC data base.

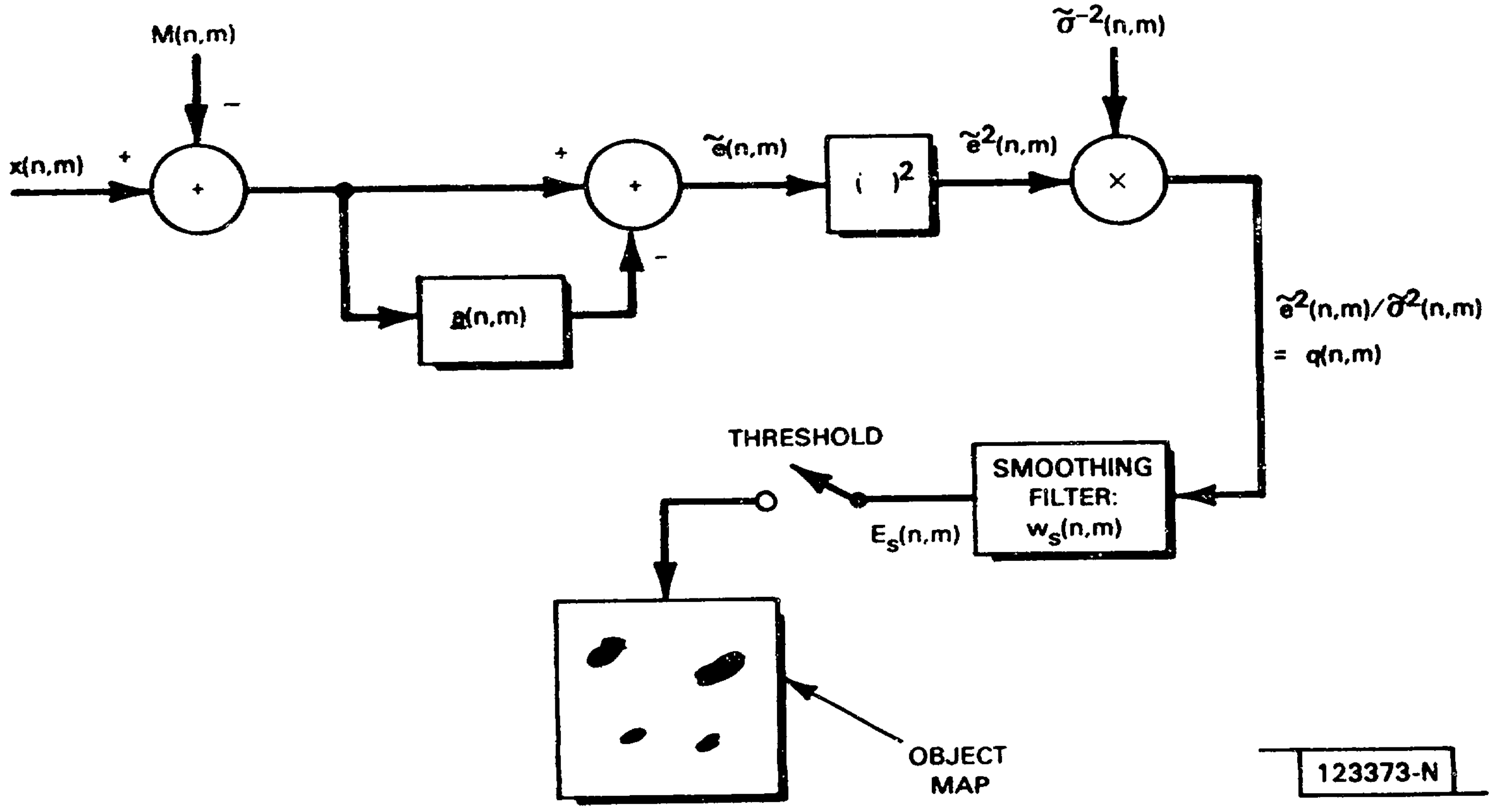


Fig. 2-9. Detection algorithm.

Example 1

Consider a sequence x(n) of the form:

$$x(n) = 0.95\ x(n - 1) + w(n) \tag{44}$$

where w(n) is zero-mean white noise. A sample function of x(n) is shown in Figure 2-10(a), and a 1-point "object" at n = 64 is shown in Figure 2-10(b). The single coefficient estimate was based on a 16-point estimation window. Figure 2-10(c) shows the squared prediction error $\tilde{e}_k^2$. The object is clearly detected.

Consider a second sequence depicted in Figure 2-11(a) of the form in Equation 44 created with a different white-noise input. A four-point object has been implanted at locations n = 90, 91, 92, and 93. As before, the single coefficient estimate was based on a 16-point estimation window. The squared prediction error, illustrated in Figure 2-11(b), gives a clear indication of the object.

Example 2

Figure 2-12(a) depicts a 1-D slice of an aerial photograph with a one-point object implanted at n = 64. In particular, this 1-D slice was extracted from a section of the aerial photograph which consisted of a grove of trees. In this example, a two-parameter noncausal model was assumed:

$$x(n) = a_1\ x(n-1) + a_2\ x(n + 1) + w(n) \tag{45}$$

where w(n) is white noise. The squared prediction error, shown in Figure 2-12(b), clearly picks out the object. A second object and its corresponding squared prediction error are shown in Figures 2-13(a) and (b), respectively.

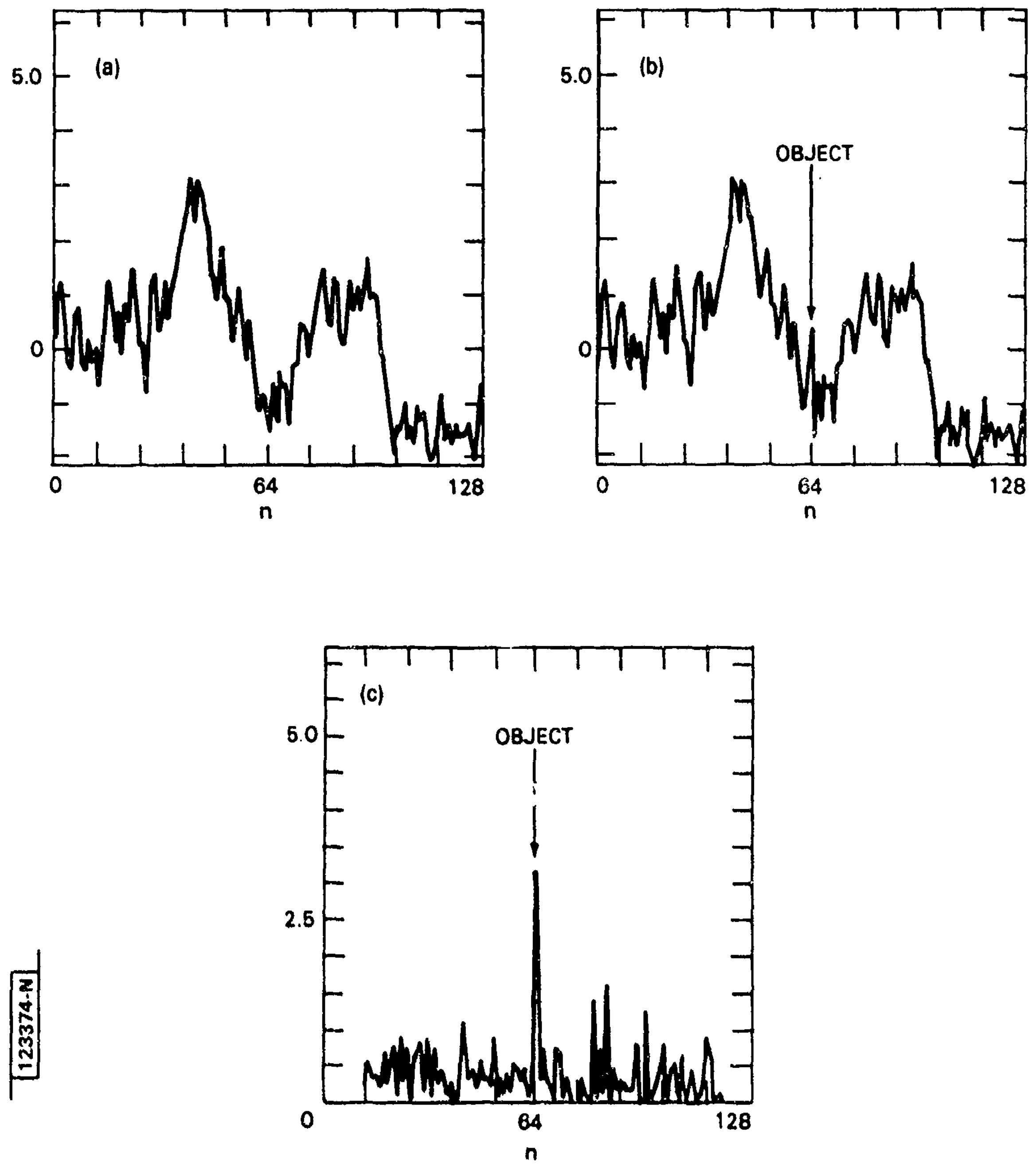


Fig. 2-10. Detection of 1-point object in Example 1. (a) 1-D random sequence, (b) random sequence with object, (c) squared prediction error.

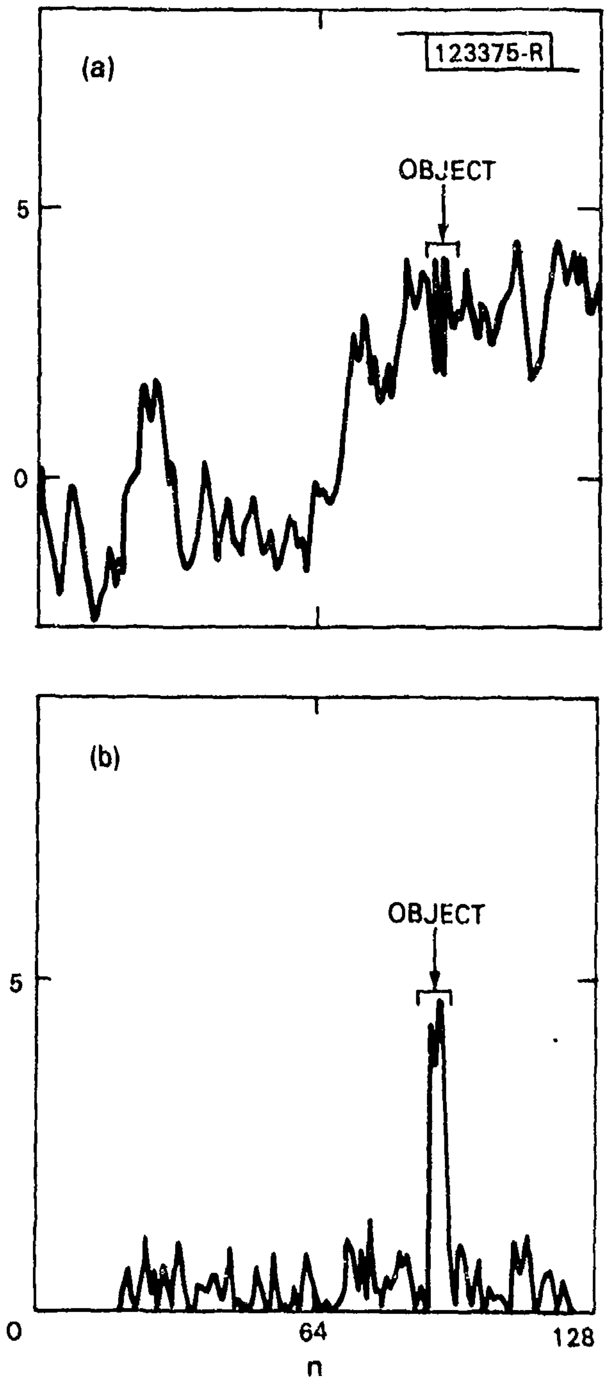


Fig. 2-11. Detection of 4-point object in Example 1. (a) 1-D random sequence with object, (b) squared prediction error.

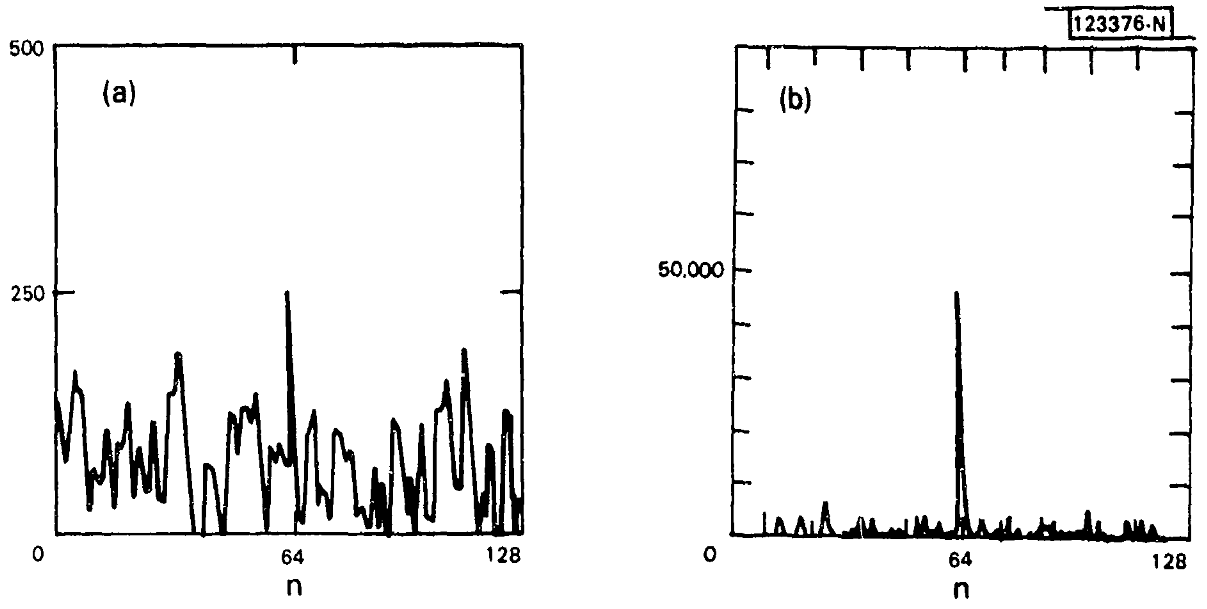


Fig. 2-12. Detection of 1-point object in Example 2. (a) slice of trees with object, (b) squared prediction error.

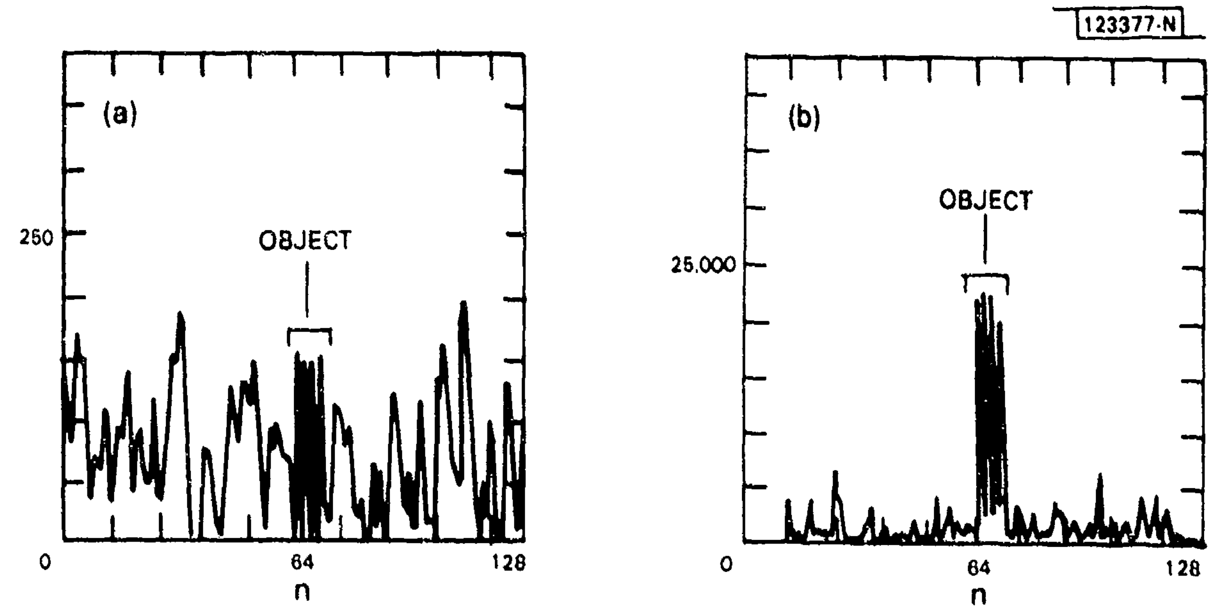


Fig. 2-13. Detection of 4-point object in Example 2. (a) slice of trees with object, (b) squared prediction error.

Example 3

Consider a 2-D sequence generated by the particular 2-D difference equation of the form:

$$x(n,m) = a(0,1)\, x(n-0,m-1) + a(1,0)\, x(n-1,m-0) + a(1,1)\, x(n-1,m-1) + w(n,m) \tag{46}$$

The background sequence (64×64 pixels in size) was generated with coefficients $a(0,1) = 0.1$, $a(1,0) = -0.9$ and $a(1,1) = 0.1$. Four objects were implanted within the image, all of a constant level, but with a variance about equal to that of the background. Moreover, the size and level of the anomalies were chosen to be visually difficult to detect from the background (see Figure 2-14(a)). The model assumed in the estimation procedure is given by the generating process (Equation 46).

The 3-D perspective and contour map of the squared prediction error $\tilde{e}^2(n,m)$ are given in Figures 2-15(a) and (b). All four objects are clearly detected, and even the two closely spaced objects are resolved. This same function, along with the smoothed $e(n,m)$ (a 3×3 smoothing window, $w_s(n,m)$, was applied in this example), are illustrated in Figures 2-14(b) and (c) after thresholding. Figure 2-14(d) shows the prediction error variance, and Figures 2-14(e) and (f) show the smoothed normalized prediction error--both appropriately thresholded.

Note that two different thresholds are applied to the smoothed normalized prediction error. The first resolves three of the four objects, the second resolves all four objects, but introduces false alarms. This is due to the inaccuracies of the estimate of the prediction error variance, which is illustrated in Figure 2-14(d). Ideally, since the background is stationary, the estimated prediction error variance should be flat. However, as seen in Figure 2-14(d), the estimate actually peaks in the region of objects--contrary to what we would hope to happen. We have encountered in this synthetic example, perhaps, what is a fundamental limitation in

123378 R

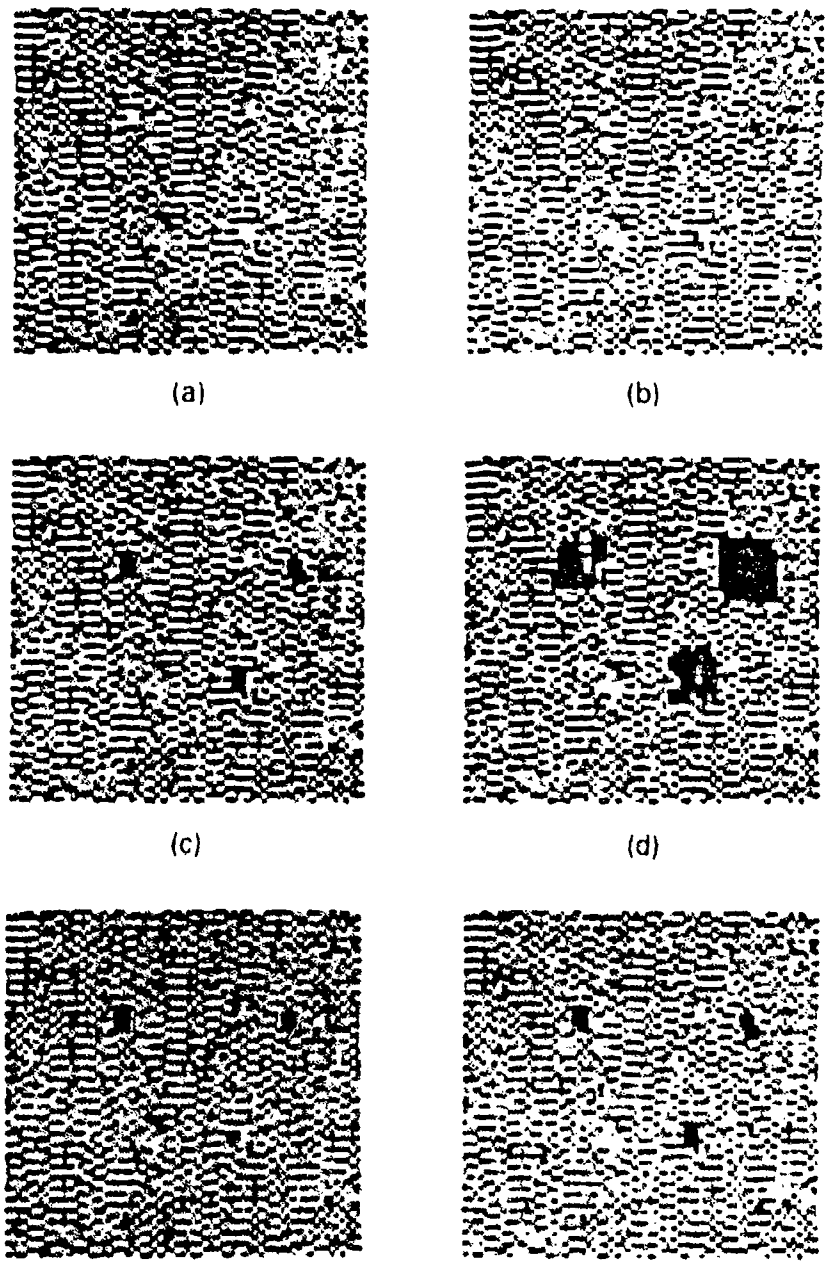


Fig. 2-14. Detection of objects in test image for Example 3. (a) test image with four objects, (b) prediction error, (c) smoothed prediction error, (d) prediction error variance, (e) smoothed normalized prediction error (high threshold), (f) smoothed normalized prediction error (low threshold).

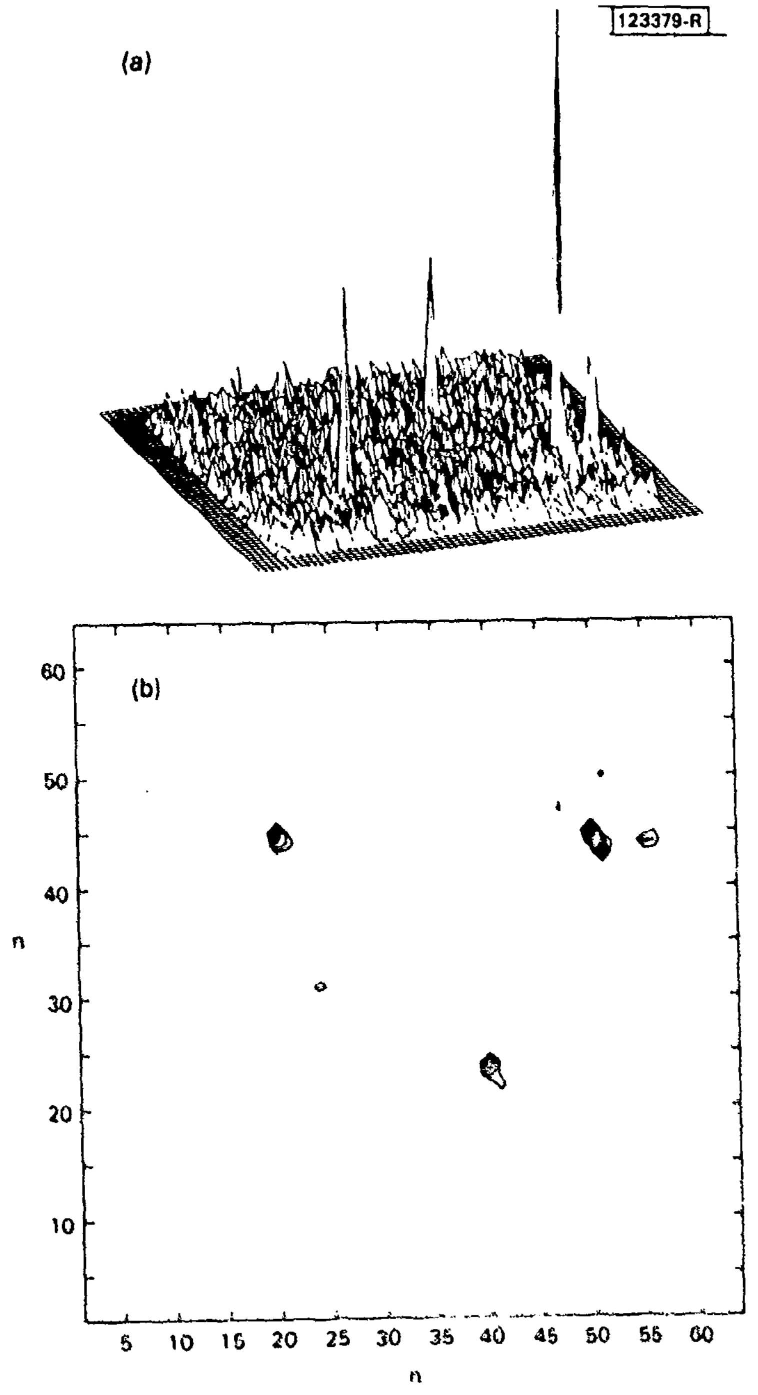


Fig. 2-15. Prediction error for Example 3. (a) 3-D perspective, (b) contour map.

measuring the background prediction error variance: the presence of objects can falsely increase the background residual variance. With a priori knowledge that the background prediction error variance is constant, we were able to improve detection.

Example 4

Figure 2-16(a) depicts a 64×64-pixel RADC image in which two 2×2 pixel synthetic objects (of constant level) have been implanted. This image was created by a 64-to-1 downsampling and smoothing of the original image. The assumed background model is the same three-parameter model used in the previous example in Equation 46. Figure 2-16(b) shows the prediction error; Figure 2-16(c), the prediction error variance; and Figures 2-16(d) and (e), the smoothed normalized prediction error (a 4×4 smoother, $w_s(n,m)$, was applied). The processed part of the image is given within the boxed area. Note that normalization of the prediction error in this case (unlike the previous example) has helped bring out the object from the more busy field background.

Example 5

Figure 2-17(a) depicts a 64×64 pixel RADC image in which two 3×3 pixel synthetic objects (of constant level) have been implanted. This field-tree image was created by a 64-to-1 downsampling and smoothing of the original image. In our first attempt to detect the two synthetic objects, the three-parameter model of Equation 46 was assumed. Although the object in field background was easily detected, the object in tree background was not detected, even with normalization by the prediction error variance.

Consequently, in our second attempt at detection, we assumed a twelve-parameter nonsymmetric half-plane autoregressive model [11]. This model is more general and thus more likely to accurately model the background [11]. Figure 2-17(b) shows the prediction error; Figure 2-17(c), the prediction error variance; and Figures 2-17(d) and (e), the smoothed normalized

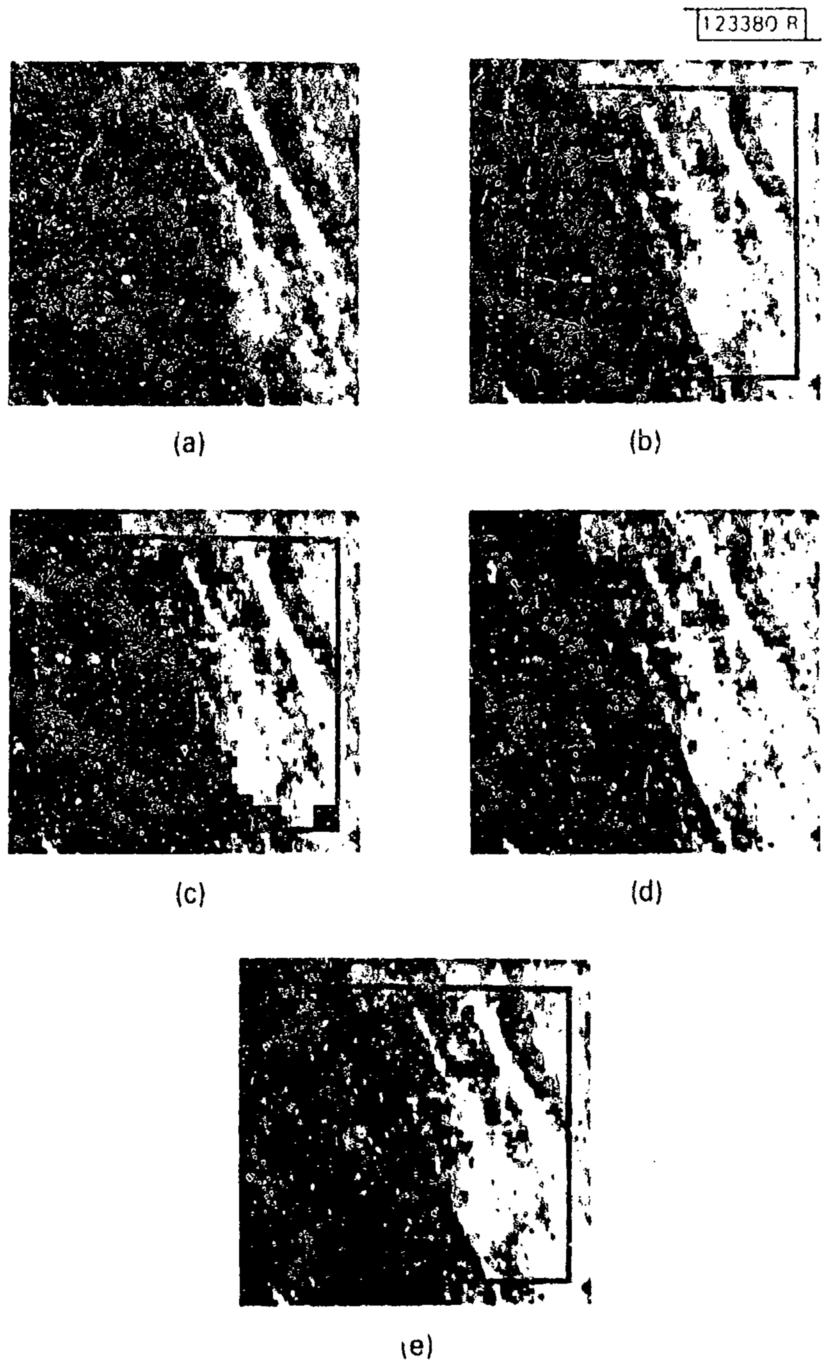


Fig. 2-16. Detection of objects in RADC image for Example 4. (a) image with two synthetic objects, (b) prediction error, (c) prediction error variance, (d) smoothed normalized prediction error (high threshold), (e) smoothed normalized prediction error (low threshold).

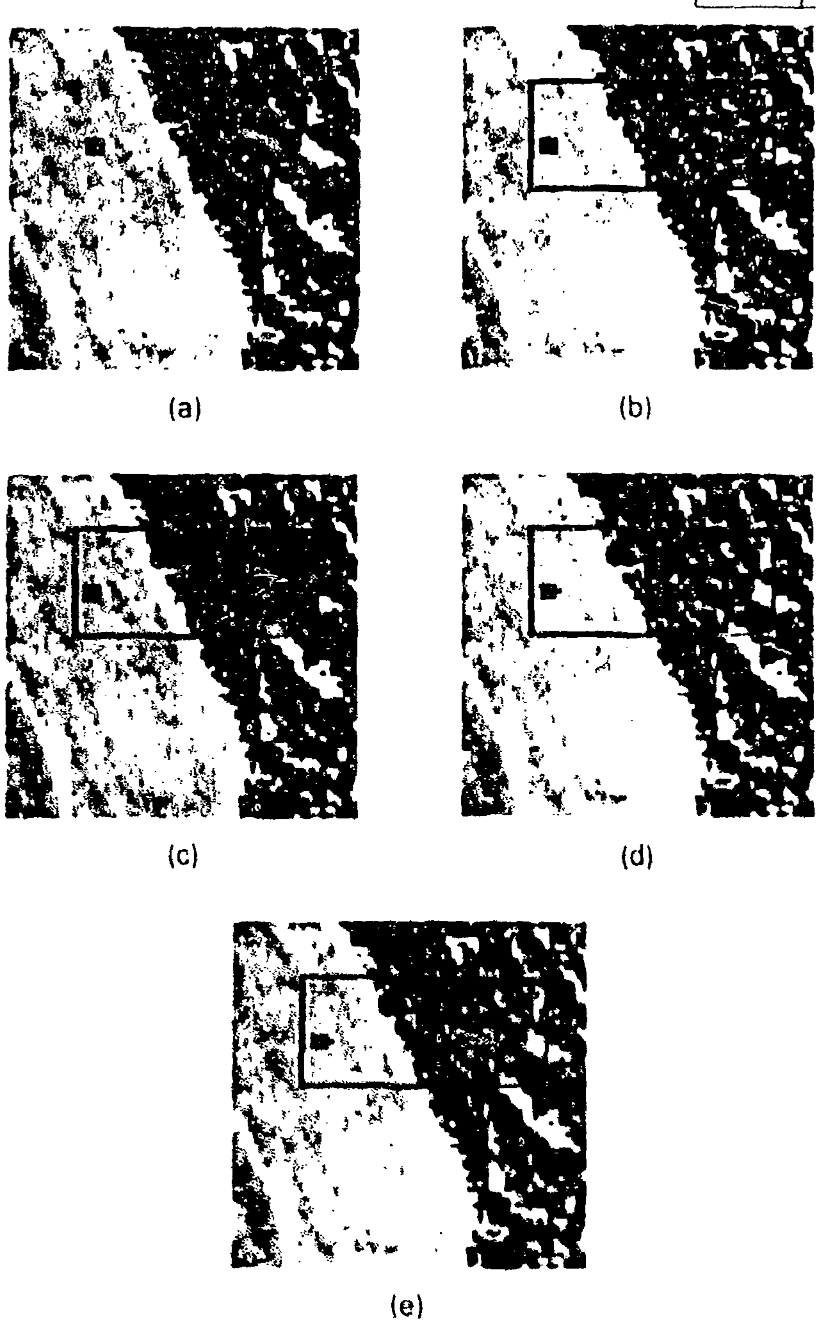


Fig. 2-17. Detection of objects in RADC image for Example 5. (a) image with two synthetic objects, (b) prediction error, (c) prediction error variance, (d) smoothed normalized prediction error (high threshold), (e) smoothed normalized prediction error (low threshold).

prediction error (a 4×4 $w_s(n,m)$ was applied). Because of the computational intensity with a twelve-parameter model, only the designated region was processed. Note that normalization of the prediction error has helped significantly in bringing out from the background the object embedded within the trees.

Example 6

The RADC image displayed in Figure 2-18(a) consists of 128×128 pixels and was created by a 16-to-1 downsampling and smoothing of the original image. The assumed background model is the same three-parameter model used in Example 3 in Equation 46. Figure 2-18(b) shows the smoothed normalized prediction error; Figure 2-18(c), the prediction error; and Figure 2-18(d), the smoothed prediction error--suitably thresholded. As in our synthetic example, the smoothed prediction error without normalization yields fewer detected objects (which may or may not be considered false alarms) than the smoothed normalized prediction error. This happens probably because the background variance appears reasonably constant throughout the image. The objects, however, can potentially introduce a false increase in the local variance, as illustrated in Figure 2-18(e), which shows a thresholded version of the prediction error variance.

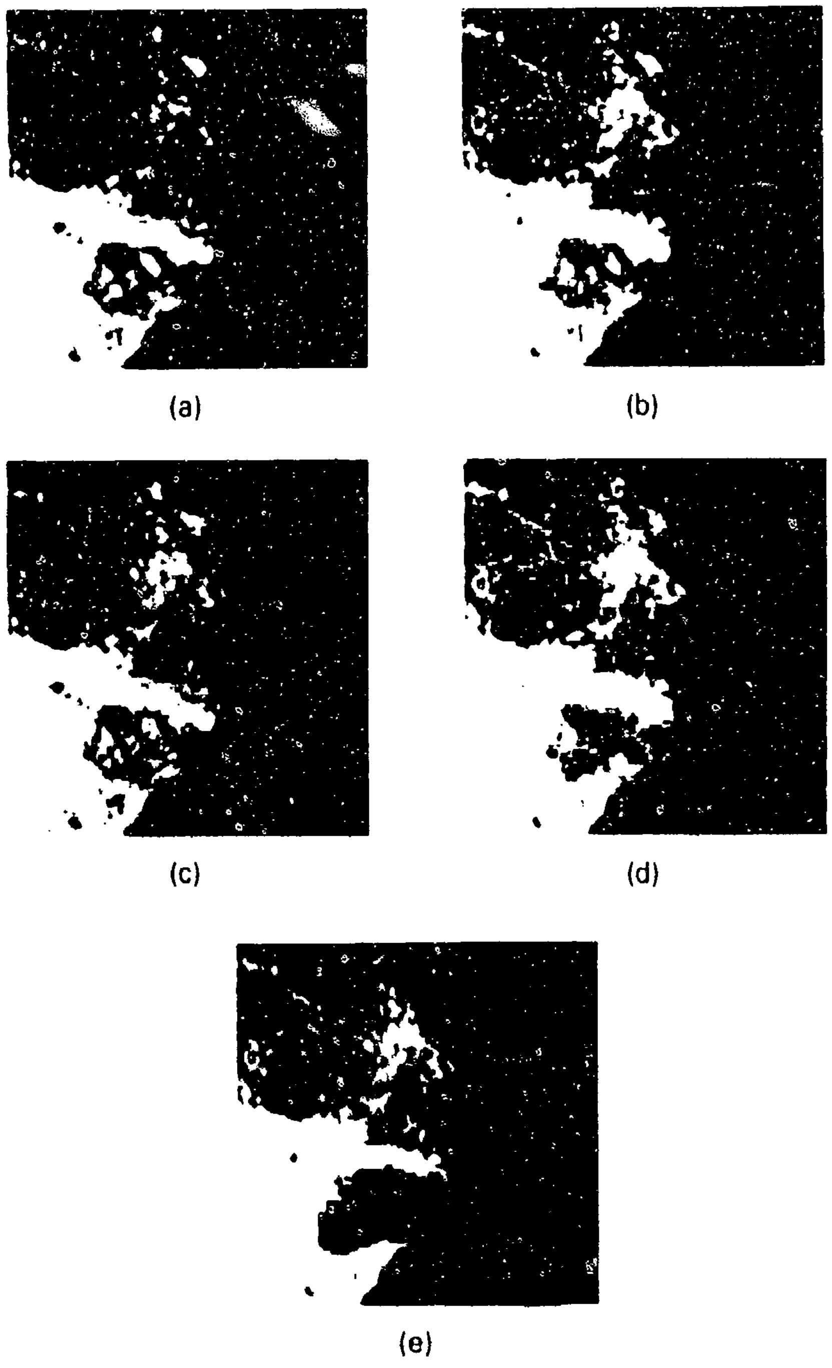
Example 7

The RADC image displayed in Figure 2-19(a) consists of 128×128 pixels and was created by a 16-to-1 downsampling and smoothing of the original image. As in the previous example, a three-parameter autoregressive model is assumed. Figures 2-19(b) to (e) make the same comparisons among the various residuals as made in Example 6.

Example 8

The RADC image displayed in Figure 2-20(a) consists of 128×128 pixels and was created by a 64-to-1 downsampling and smoothing of the original

123382-R



(a) (b) (c) (d) (e)

Fig. 2-18. Comparison of smoothed predictio. error and smoothed normalized prediction error for Example 6. (a) RADC image, (b) smoothed normalized prediction error, (c) prediction error, (d) smoothed prediction error, (e) prediction error variance.

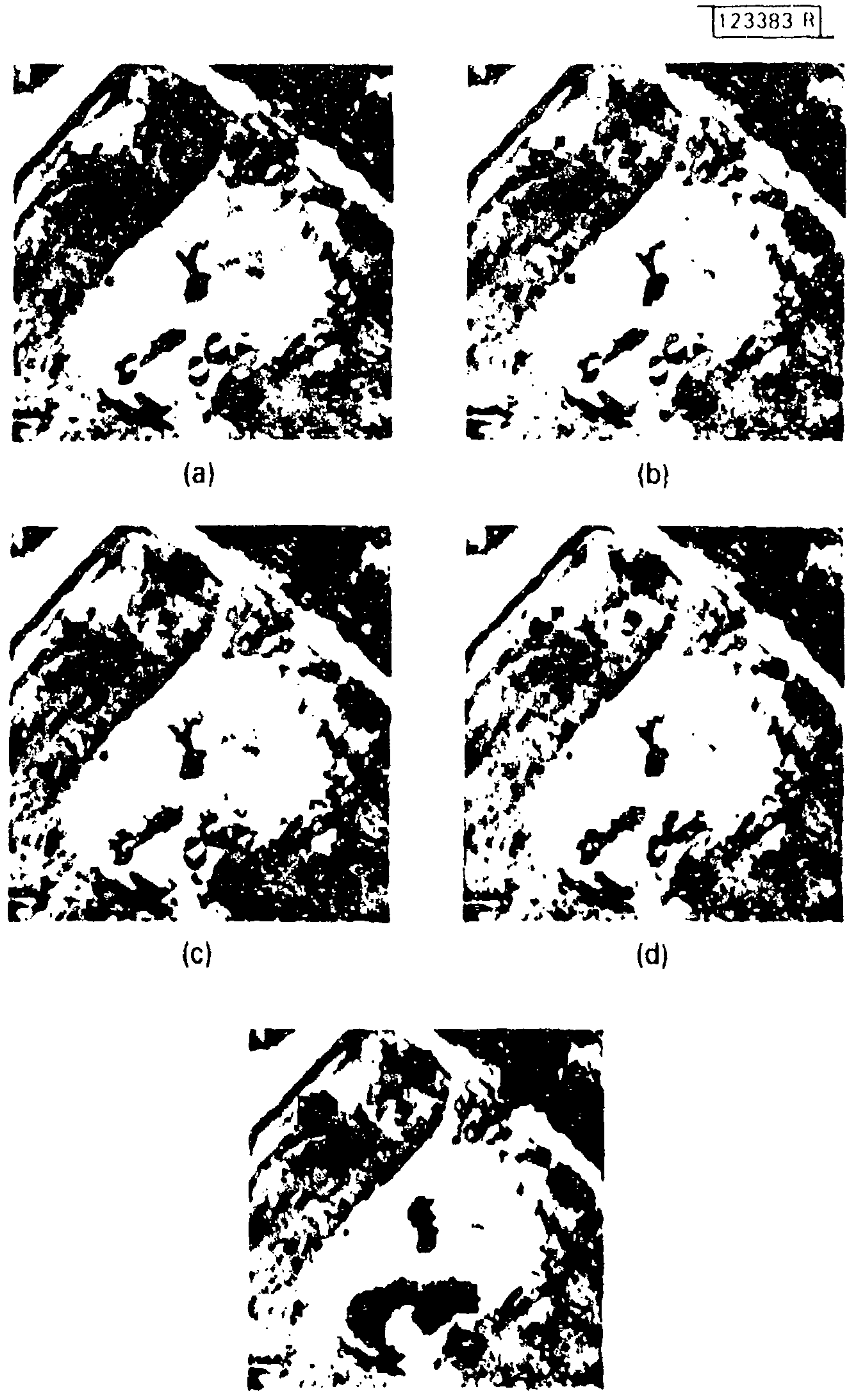


Fig. 2-19. Comparison of smoothed prediction error and smoothed normalized prediction error of Example 7. (a) RADC image, (b) smoothed normalized prediction error, (c) prediction error, (d) smoothed prediction error, (e) prediction error variance.

123384 R

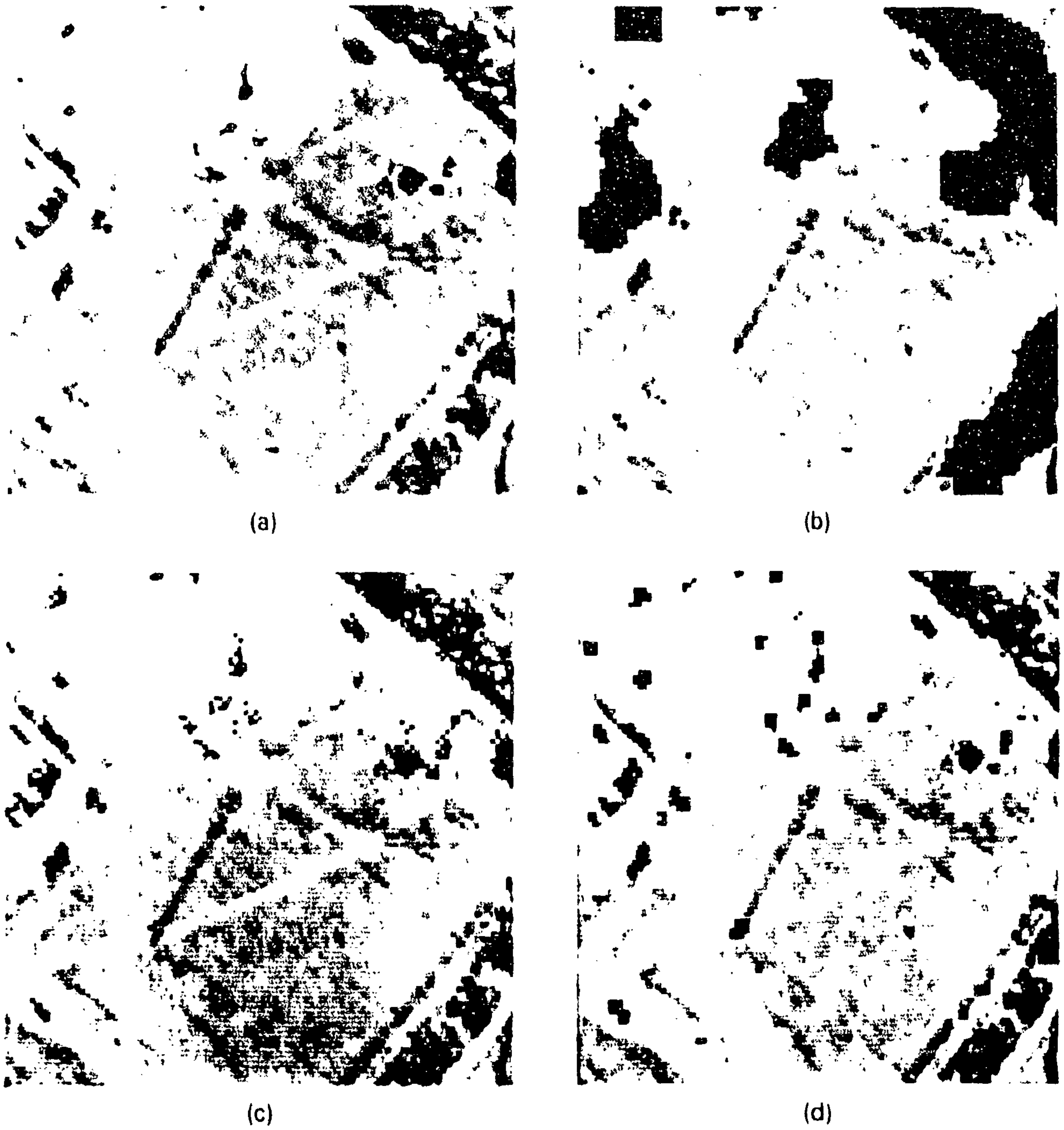


Fig. 2-20. Comparison of prediction error and smoothed normalized prediction error with first-quadrant mask for Example 8. (a) RADC image, (b) prediction error variance, (c) prediction error, (d) smoothed normalized prediction error.

image. In this example, we consider a first-quadrant causal, second-quadrant causal autoregressive model, and average of the two. This average represents an attempt to eliminate the directionality of the approximate significance test.

Figures 2-20 and 2-21 illustrate the results with first-quadrant (three-parameter) and second-quadrant (three-parameter) prediction masks, respectively. Figure 2-22 summarizes our results by depicting the smoothed normalized prediction errors and their average. Note that the individual smoothed normalized prediction errors do well in detecting most objects, while the average appears to deteriorate the performance.

Two additional experiments that were performed with this data are shown in Figures 2-23 and 2-24. Figure 2-23 shows a different thresholded version of the CCF [14] corresponding to the prediction of Figure 2-20. The CCF bears little resemblance to our prediction errors. Moreover, due to the large estimation window (i.e., 10×10 pixels), this function is small everywhere--reflecting little sample-to-sample change in the coefficient estimates. Finally, in Figure 2-24, we depict the smoothed noncausal normalized prediction error. The noncausal prediction mask is an eight-point nearest neighbor mask. The results on this image and others (e.g., Example 5) are encouraging, but appear to do no better (and perhaps worse) than the causal prediction masks.

Example 9

Consider the 64×64 pixel RADC image in Figure 2-25, generated by down-sampling the original image by 16-to-1 with smoothing. This image is particularly interesting because of the presence of a radio tower in the lower right-hand corner of the image. Note that the top of the radio tower has been clearly detected.

It is interesting to observe that in Examples 7, 8, and 9 normalization of the prediction errors helped detection and reduced false alarms by reducing the background variance in busy regions such as the tree and brush areas.

123385 R

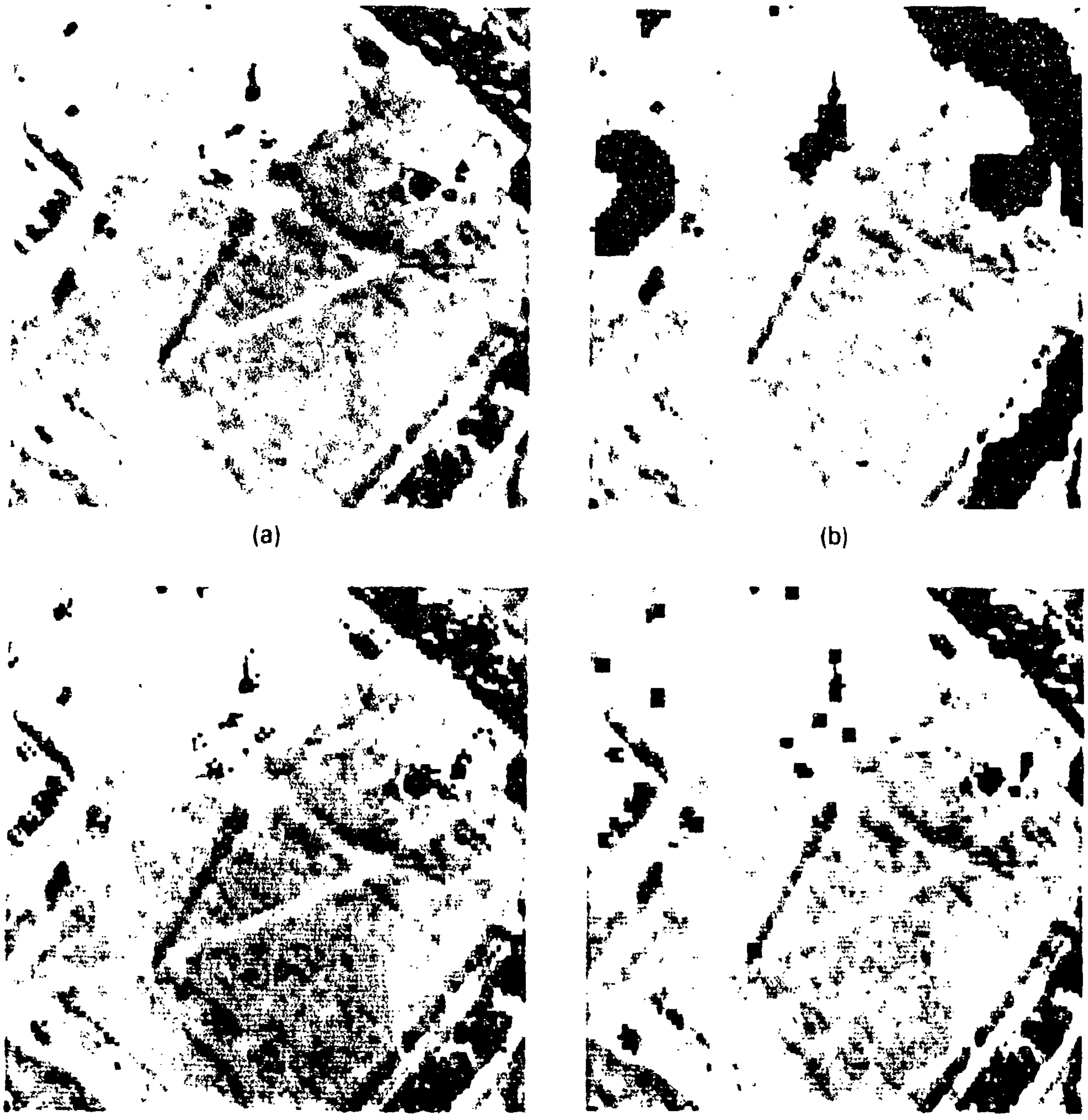


Fig. 2-21. Comparison of prediction error and smoothed normalized prediction error with second-quadrant mask for Example 8. (a) RADC image, (b) prediction error variance, (c) prediction error, (d) smoothed normalized prediction error.

PRECEDING PAGE BLANK-NOT FILMED

123386 R

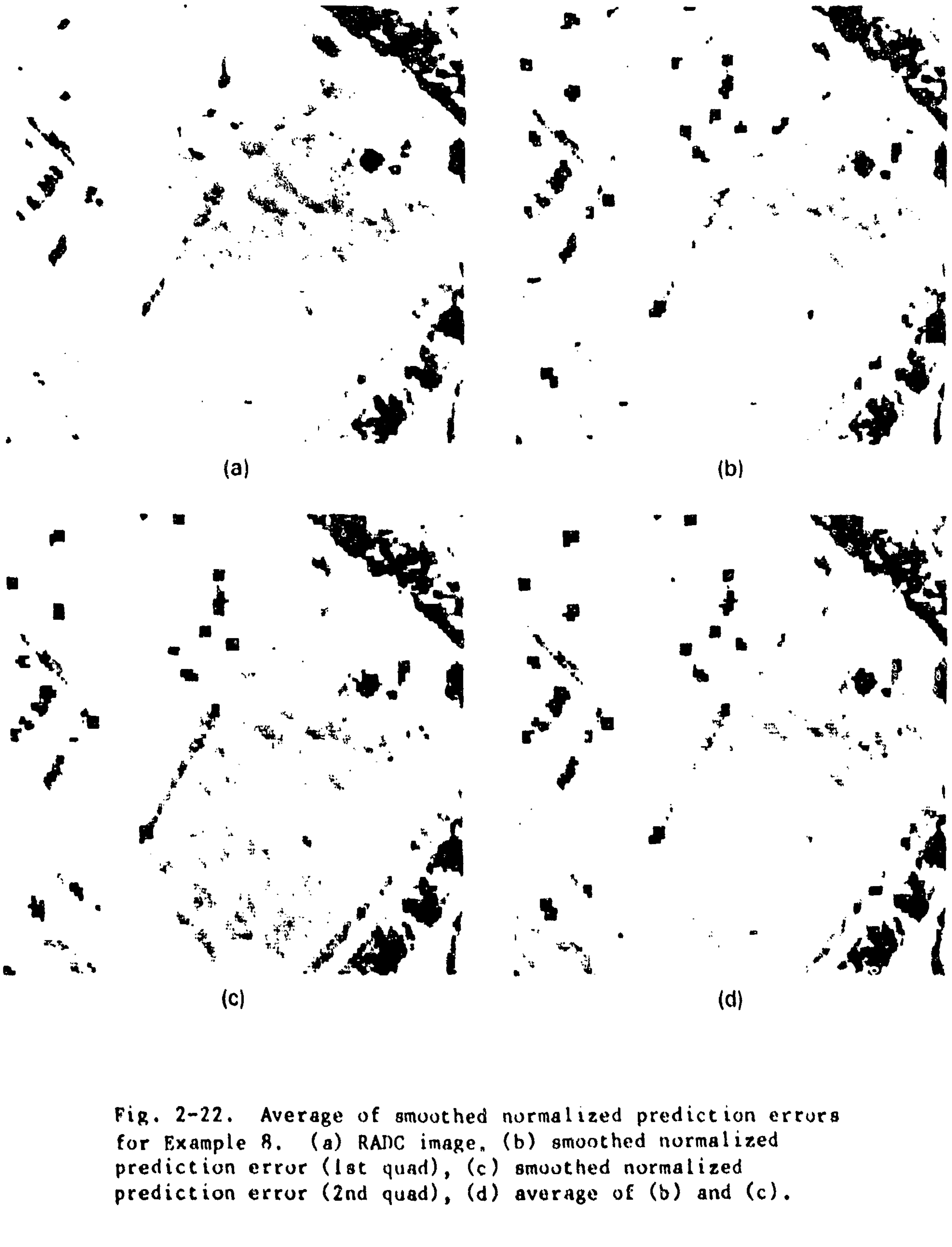


Fig. 2-22. Average of smoothed normalized prediction errors for Example 8. (a) RADC image, (b) smoothed normalized prediction error (1st quad), (c) smoothed normalized prediction error (2nd quad), (d) average of (b) and (c).

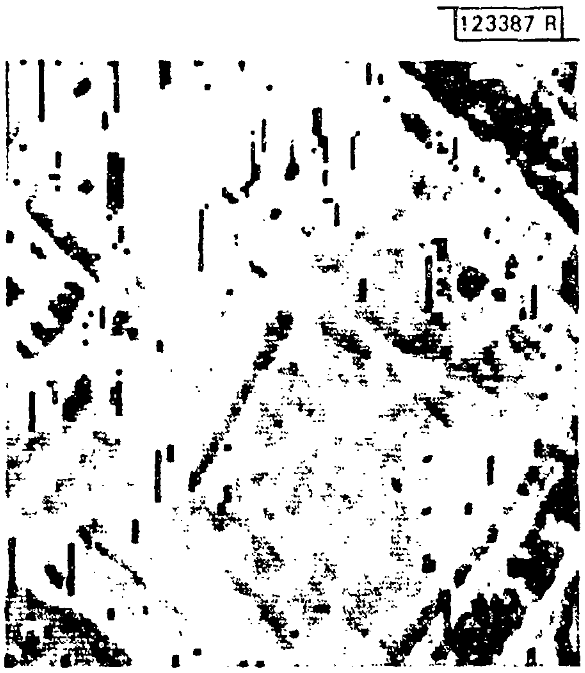


Fig. 2-23. CCF corresponding to Figure 2-20.

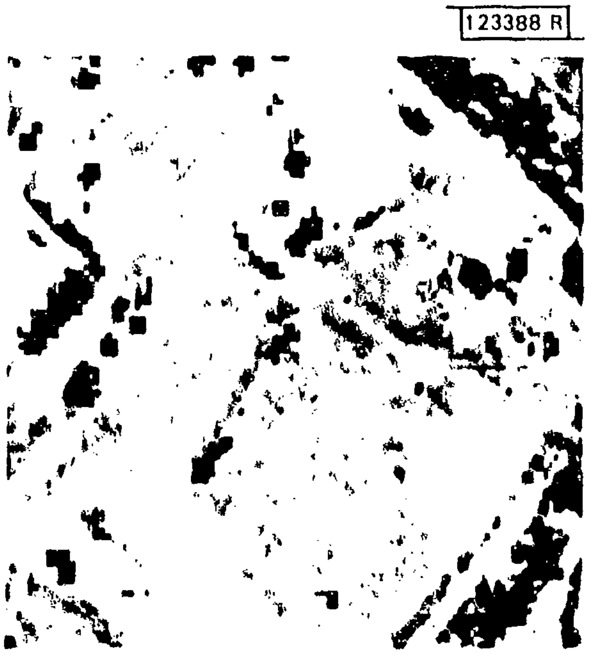
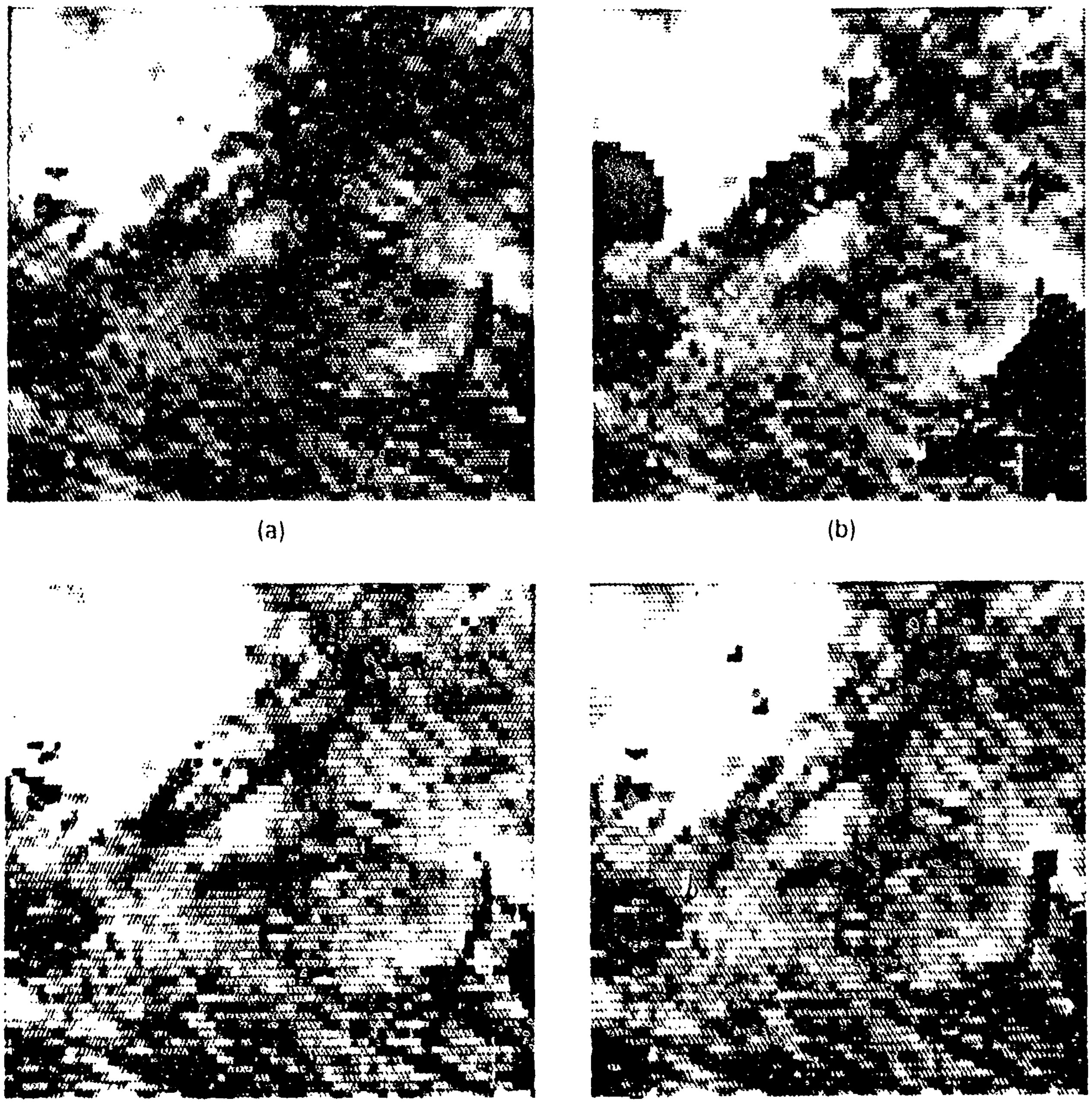


Fig. 2-24. Smoothed normalized noncausal prediction error for Example 8.

123389 R



(a) (b) (c) (d)

Fig. 2-25. Comparison of prediction error and smoothed normalized prediction error for Example 9. (a) RADC image, (b) prediction error variance, (c) prediction error, (d) smoothed normalized prediction error.

3. ADAPTIVE CONTRAST ENHANCEMENT PROGRAM

We have developed two algorithms for adaptive contrast enhancement of images degraded by cloud and/or shadow. The first algorithm is a simple gain in local contrast [7], and the second is more complex and sophisticated--the adaptive homomorphic algorithm [8]. A computer program system that implements these techniques was written in 'C' language under the UNIX operating system. The adaptive contrast enhancement system offers more operation modes than just the two techniques specified above. The following sections describe in detail all the options that the system offers to the user. The system consists of two programs: image_enhanc.c and scale.c; these are described. In addition, an actual example of how to use the program is given.

The system described was installed in RADC's AFES system and a version of it exists on our computer (VAX with UNIX as an operating system).

3.1 Description of image_enhanc.c

Program image_enhanc.c implements the adaptive contrast enhancement. The program offers three operation modes:

1. intensity domain
2. density domain (log) on a positive image
3. density domain (log) on a negative image

After selecting one of the three modes, the user can choose to process the resulting array $f(n_1,n_2)$ in two different ways:

1. point-by-point (done in the space domain)
2. block processing with 50 percent overlap (done in the frequency domain)

3.1.1 Point-by-Point Processing

As shown in Figure 3-1, the point-by-point processing separates the input array into two components: the local mean (f_L) and the local contrast (f_H).

The local mean is computed as the weighted average of the input array over a small 2-D window (W), i.e.,

$$f_L(n_1,n_2) = \frac{1}{U} \cdot \sum_{m_1=n_1-\text{hwsize}}^{{}_1+\text{hwsize}} \; \sum_{m_2=n_2-\text{hwsize}}^{n_2+\text{hwsize}} f(m_1,m_2)$$

$$\cdot\, W(m_1-n_1,m_2-n_2) \qquad (45a)$$

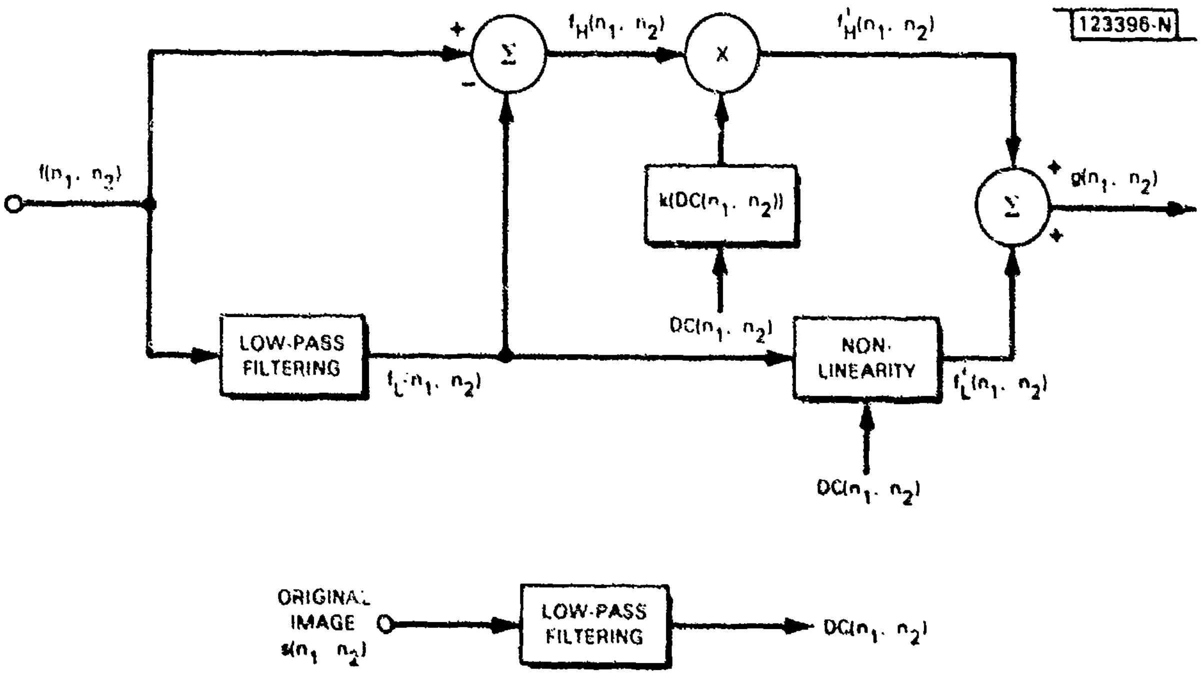


Fig. 3-1. Block diagram of point-by-point processing.

where

$$U = \sum_{i=-hwsize}^{hwsize} \sum_{j=-hwsize}^{hwsize} W(i,j) \qquad (45b)$$

The size of the filter W is $(2\ hwsize+1)^2$. W can be a rectangular window or a Gaussian window.

The local contrast is obtained by subtracting the local mean (f_L) from the value of the point (n_1,n_2), i.e.,

$$f_H(n_1,n_2) = f(n_1,n_2) - f_L(n_1,n_2) \qquad (46)$$

After separating the signal into its two components, the enhancement is achieved by modifying each component according to a function (which is defined by the user) of the local mean of the intensity values -- $DC(n_1,n_2)$, i.e.,

$$DC(n_1,n_2) = \frac{1}{U} \sum_{i=n_1-hwsize}^{n_1+hwsize} \sum_{j=n_2-hwsize}^{n_2+hwsize} s(i,j)\ W(i-n_1,j-n_2) \qquad (47)$$

where $s(i,j)$ is the value of the point (i,j) in the input image (intensity domain).

The local contrast is multiplied by a gain k [.], which is a function of $DC(n_1,n_2)$. If $k[DC(n_1,n_2)]>1$, the contrast is increased; if it is less than 1, the contrast is decreased. The function k is defined as piecewise linear. The new contrast $f_H(n_1,n_2)$ is defined as

$$f'_H\ (n_1,n_2) = f_H(n_1,n_2)\ k[DC(n_1,n_2)] \qquad (48)$$

The local mean goes through a nonlinearity; more specifically, the new mean $f'_L(n_1,n_2)$ is defined as

$$f'_L(n_1,n_2) = [f_L(n_1,n_2)-midrang] \cdot \ell[DC(n_1,n_2)] + midrang \qquad (49)$$

where midrang is the value of the midpoint in the range of values in the input array (in intensity processing it is 128 and in density processing it is $4 \approx 0.5 \log (255+0)$). The ℓ function ranges between zero and one and is defined as piecewise linear. Figure 3-2 shows the way the new local mean is generated.

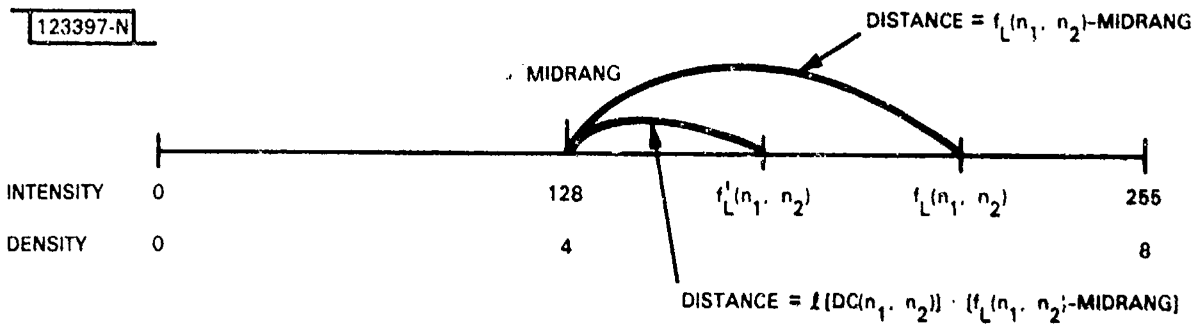

123397-N
DISTANCE = $f_L(n_1, n_2)$-MIDRANG
MIDRANG
INTENSITY
0
128
$f'_L(n_1, n_2)$
$f_L(n_1, n_2)$
255
DENSITY
0
4
8
DISTANCE = $\ell[DC(n_1, n_2)] \cdot [f_L(n_1, n_2)$-MIDRANG]

Fig. 3-2. Local mean processing.

We would like to change the local mean so it will be closer to the midrang. This is done for two reasons. The first is to compensate for the degradation (clouds → high DC, shadows → low DC). The second is to account for the increase in contrast that is achieved. The two new components (f'_L, f'_H) are combined to produce the processed array $g(n_1,n_2)$. The different parameters are described in the next section.

Parameters

The user has control over eight parameters in the point-by-point processing. The parameters are described below, and actual values are given in Table 3-1.

TABLE 3-1

PARAMETERS FOR POINT-BY-POINT PROCESSING

Parameter	Defaults*	Range of Normal Values
wtype	g	g or r
hwsize	5	≤15
var	6.	<2 hwsize+1
numpt	6	≤21
k[0],...,k[numpt-1]	2., 4., 5., 7., 8., 8.	12.≥k[i]≥1.
ℓ[0],...,ℓ[numpt-1]	0.2., 0.2, 0.2, 0.2, 0.2, 0.2	1.≥ℓ[i]≥0.
mn[0],..,mn[numpt-1]	0., 100., 150., 175., 255., 255.	xmax≥mn[i]≥0
xmax	255.	

*Defaults exist only for intensity processing of a cloudy image.

1. wtype - Shape of window to be used in computing the local mean.
   - g - Gaussian window
   - r - rectangular window
2. var - Thickness of Gaussian window if chosen (see Figure 3-3).

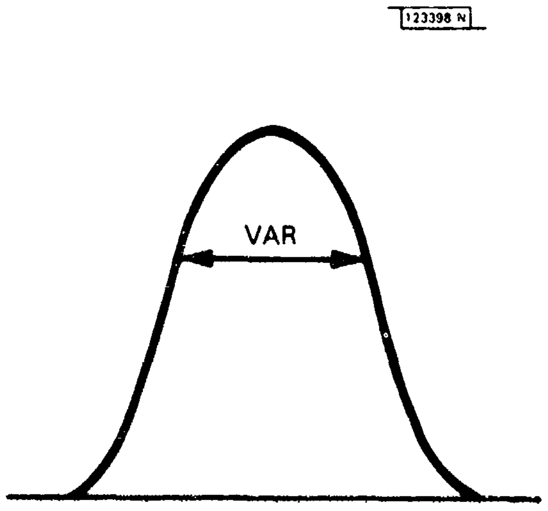


Fig. 3-3. Thickness of Gaussian window.

3. hwsize - $(2\ \text{hwsize}+1)^2$ is the extent of that window (max 31 × 31).
4. numpt - number of points in the piecewise linear functions (k,ℓ) (maximum 21 points).
5. k - the array that contains the values of the k function (max size 21).
6. ℓ - the array that contains the values of ℓ (max size 21).
7. mn - the array that contains the corresponding mean values of the k and ℓ functions, i.e.,

   Remark

   mn[0] is set to 0 (for the beginning of the range)

   mn[numpt-1] is set to the maximum gray level allowed (usually 255.)

8. xmax - maximum gray level in the input image (usually 255.) (see Figure 3-4).

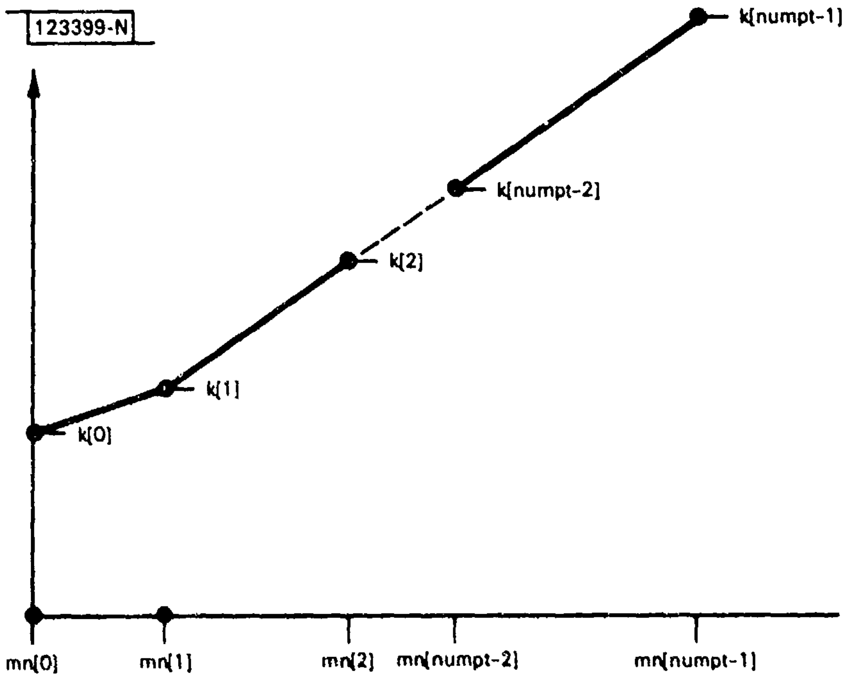


Fig. 3-4. The k function described as a piecewise linear function of the mean value mn.

3.1.2 Block-by-Block Processing

The block-by-block processing (Figure 3-5) uses a 2-D triangular window, with an overlap of 50% to window the data. Along one dimension, this looks like the sketch in Figure 3-6.

A high-pass filter is applied to each windowed section. The exact shape of the high-pass filter is determined according to the value DC_i, which is the average value (using a rectangular window) of the current input image (intensity domain) section. The exact shape of the filter is defined by three parameters (A, B, and C shown in Figure 3-7) which are given for DC = 0 and DC = max. allowed. For an intermediate value of DC_i, each of the three parameters is determined by the formula:

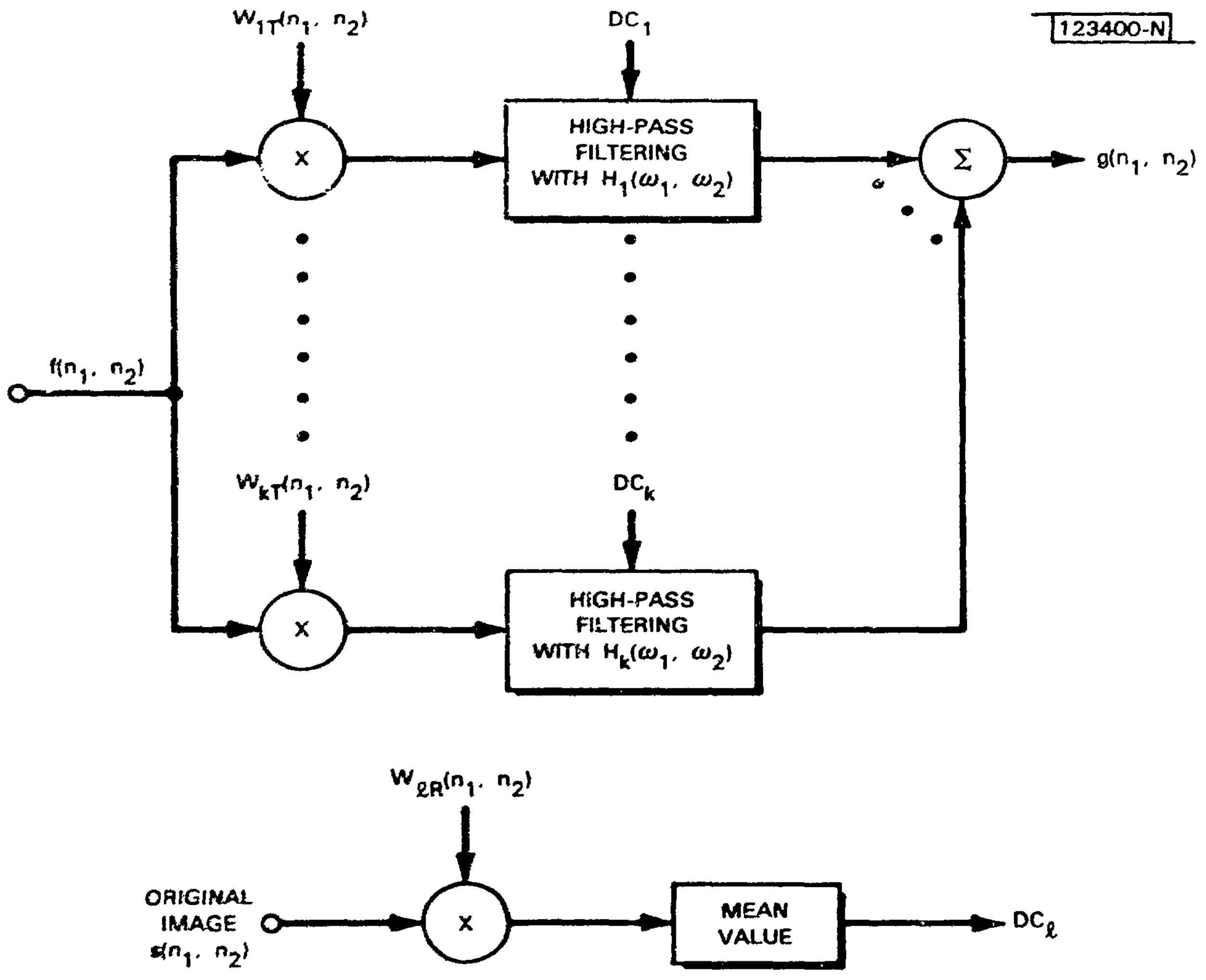


Fig. 3-5. Block-by-block processing. $W_{\ell T}$ is triangular window; $W_{\ell R}$ is rectangular window.

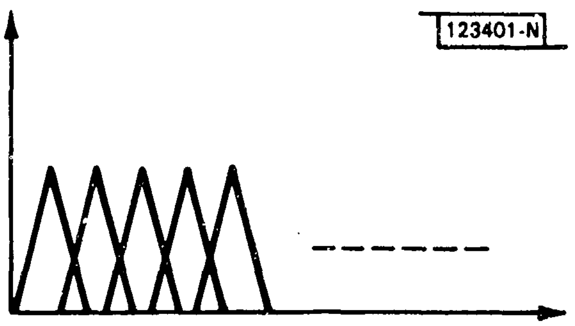


Fig. 3-6. Processing along one dimension.

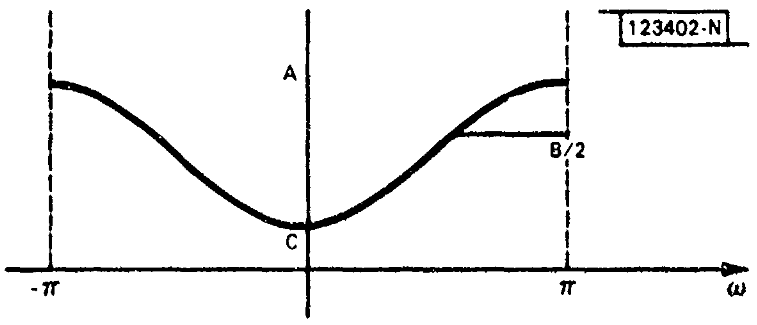


Fig. 3-7. Gaussian-shaped filter.

$$X_i = \frac{(X_{DC=max} - X_{DC=0})^2 \, DC_i^{\,2}}{(\text{max DC allowed})^2} + X_{DC=0} \text{ for } X = A, B, \text{ or } C \quad (50)$$

The filter is a Gaussian-shaped filter as shown in Figure 3-7. The following section describes in detail the various parameters of the processing. The parameters are listed in Table 3-2.

TABLE 3-2
PARAMETERS FOR BLOCK PROCESSING

Parameter	Default*	Range of normal values
hwsize	8	4,8,16
xmax	255.	
hfdc0	1.1	
hfdcmax	1.5	
ℓfdc0	0.53	
ℓfdcmax	1.2	
vardc0	5.	
vardcmax	20.	

*Defaults are designed for negative density processing of a cloudy image.

<u>Parameters</u>

1. hwsize - $(2\ \text{hwsize})^2$ is the size of the processed section and the filter that is applied (up to 32 × 32 for 512 × 512 images and up to 16 × 16 for 1024 × 1024 images). It should be a power of two.

2. xmax - Max gray level that exists in the current image (usually 255).
3. hfdc0 - Value of A (in Figure 3-7) for mean level = 0 (the lowest).
4. hfdcmax - Value of A for mean level = xmax.
5. ℓfdc0 - Value of C (in Figure 3-7) for mean level = 0.
6. ℓdcmax - Value of C for mean level = xmax.
7. vardc0 - Value of B (in Figure 3-7) for mean level = 0.
8. vardcmax - Value of B for mean level = xmax.

3.1.3 Resulting Output

After the point-by-point or block processing, the resulting output array is exponentiated if it is a density image. Inversion of negative mode images can be done in the program scale.c.

3.2 Description of scale.c

For some of the processing modes allowed in image_enhanc.c (such as all density processing and intensity block processing), the resulting output array does not lie in the required range for display (0-255). Thus, the output array must be scaled. Since the AFES auto_wndw.h software permits only one pass over the data and scaling must be done in two passes, this led to the need for the scale.c program to be run after image_enhanc.c for the specified modes of operation.

The program scales the output array of image_enhanc.c according to a number of parameters that are provided by image_enhanc.c in a file (see running instructions). The parameters are:

1. minv - minimum value in the array to be scaled.
2. maxv - maximum value in the array to be scaled.
3. pnt - point or block processing.
4. den - density or intensity mode.
5. nega - negative or positive mode.

According to these values, the array is scaled in the following manner:

1. For block processing of intensity mode or for any processing of positive density mode, scale the input array such that

```
minv --------> 0
maxv --------> 255
```

2. For any processing of negative density mode, scale such that

```
minv --------> 255
maxv --------> 0
```

3.3 Running Instructions

The following sections describe the interaction between the user and each one of the programs.

3.3.1 image_enhanc. (the load module of image_enhanc.c)

```
User:      image_enhanc    <image1.data>    <image2.data>
           where
           image1.data is the name of the input image.
           image2.data is the name of the output image.
Terminal:  do you want to process in INTENSITY or DENSITY domain?
           intensity - i, density - d
User:      i (or d)

If density domain    {

Terminal:  do you want to process the NEGATIVE or POSITIVE image?
User:      p (or n)
                 }
```

Terminal: do you want a POINT-by-POINT processing or a BLOCK processing?
block - b, point - p

User: b (or p)

<u>If POINT processing was selected:</u>

Terminal: do you want to change the defaults? The defaults exist only for intensity processing of a cloudy image.
yes - y, no - n

User: n (or y)

For density processing {

Terminal: enter filename for scaling parameters to be used by scale program

User: param.data

}

If the defaults were selected, the processing starts with no additional interaction with the user for this program. If the defaults were not selected, the dialogue continues.

Terminal: enter hwsize=half size of window to compute mean value - n (2n+1) (2n+1). n is limited up to 15 for a 512 × 512 image and up to 7 for a 1024 × 1024 image.

User: value of hwsize (integer).

Terminal: maximum gray level in the image (usually 255)?

User: value of xmax (float).

Terminal: the following inputs define the k and ℓ functions (as functions of the mean value). k - multiplies the local contrast >=1 and ℓ - is the nonlinearity that is applied to the local mean and is 0.=<ℓ<=1. If ℓ=1., then the original local mean is kept. If ℓ=0., then the new local mean is exactly the midrange of levels allowed.

NUM OF POINTS IN FUNCTIONS (up to 21)?

first point must be for mean value=0. and last must be for mean value=xmax.

User: value of numpt (integer)

Terminal: contrast amplifier=k[0], nonlinearity = ℓ[0] for mean value=0.

ser: values of the contrast amplifier and nonlinearity (both float) for mean value=0.

Terminal: mean value=mn[ ], contrast amplifier=k[ ],
nonlinearity = ℓ[ ] for j=... j=1,...
..,numpt-2

User: values of the jth mean value (float), jth contrast amplifier (float) and the jth nonlinearity (float).

Terminal: contrast amplifier=k[numpt-1], nonlinearity=ℓ[numpt-1] for mean value=xmax.

User: values of contrast amplifier and nonlinearity (both float) for mean value=xmax.

Terminal: Gaussian window - g or rectangular - r for calculating the local mean value?

User: g (or r)

If Gaussian window was selected {

Terminal: thickness (variance) of Gaussian window?

User: value of variance (float).

}

Processing starts with no additional interaction with the user.

<u>If BLOCK processing was selected:</u>

Terminal: do you want to change the defaults? The defaults exist only for a negative density processing of a cloudy image.
yes - y, no - n

User: n (or y)

```
Terminal:  enter filename for scaling parameters to be used by scale program
User:      param. data
```

If the defaults were selected, the processing starts with no additional interaction with the user for this program. If the defaults were not selected, the dialogue continues.

```
Terminal:  maximum gray level in the image (usually 255.)?
User:      value of xmax (float)
Terminal:  mapping variables HFDCO, HFDCMAX - the values of the high-pass
           filter at the largest frequency for mean values 0. and xmax
           (maximum gray level).
User:      values of hfdc0 and hfdcmax (float).
Terminal:  mapping variables LFDCO, LFDCMAX - the values of the high-pass
           filter at W=(0,0) for mean values 0. and xmax.
User:      values of ℓfdc0 and ℓfdcmax (float).
Terminal:  mapping variables VARDCO, VARDCMAX - the thickness of the
           Gaussian high-pass filter for mean values 0. and xmax.
User:      values of vardc0 and vardcmax (float).
Terminal:  enter hwsize=half window size for the adaptive filtering
           (8,16,32..) - a number that is divisible by the row and column
           length of the image. hwsize is limited up to 16 for a 512 × 512
           image and up to 8 for a 1024 × 1024 image.
User:      value of hwsize (integer).
           End of interaction.
```

Specific example:

```
User:      image_enhanc    cloud.data proccloud.data
Terminal:  do you want to process in INTENSITY OR DENSITY domain?
           intensity - i, density - d
User:      i
Terminal:  do you want a POINT-by-POINT processing or BLOCK processing?
           block - b, point - p
User:      p
```

```
Terminal:  do you want to change the defaults? The defaults exist only for
           intensity processing of a cloudy image.
           yes - y, no - n
User:      y
Terminal:  enter hwsize=half size of window to compute mean value - n
           (2n+1)(2n+1).  n is limited up to 15 for 512 × 512 image and up to
           7 for a 1024 × 1024 image.
User:      5
Terminal:  maximum gray level in the image (usually 255.)?
User:      255.
Terminal:  the following inputs define the k and ℓ functions (as functions of
           the mean value).  k = multiplies the local contrast >=1. and ℓ -
           is the nonlinearity that is applied to the local mean and is
           0.=<ℓ<=1..  If ℓ=1., then the original local mean is kept.  If
           ℓ=0., then the new local mean is exactly the midrang of levels
           allowed.

           NUM OF POINTS IN FUNCTIONS (up to 21)?

           first point must be for mean value=0. and last must be for mean
           value=xmax.
User:      3
Terminal:  contrast amplifier=k[0], nonlinearity=ℓ[0] for mean value=0.
User:      2.  0.7
Terminal:  mean value=mn[ ], contrast amplifier=k[ ], nonlinearity=ℓ[ ] for
           i = 1
User:      150.   4.   0.5
Terminal:  contrast amplifier=k[numpt-1], nonlinearity=ℓ[numpt-1] for mean
           value=xmax.
User:      6.   0.2
Terminal:  Gaussian window - g or rectangular - r for calculating the local
           mean value?
```

```
User:      g
Terminal:  thickness (variance) of Gaussian window?
User:      6.
```

Processing starts with no additional interaction with the user.

3.3.2 Scale (the load module of scale.c)

```
User:      scale <image2.data> <image3.data> <param.data>
```

where:

image2.data is the name of the ouput file of image_enhanc.
image3.data is the name of the resulting processed image.
param.data is the name of the file that contains information for the scale program. The filename was given as a parameter in image_enhanc program.

3.4 Installation Instructions

The two programs were dumped on tape using:

```
tar c image_enhanc.c    scale.c
```

To read these files from tape, use

```
tar x image_enhanc.c    scale.c
```

<u>To compile and load:</u>

```
         image_enhanc.c                                  stdio.h
   CC         or          + add options for using        math.h
           scale.c                                       auto_wndw.h
```

Special running instructions: The program image_enhanc.c requires a 64K byte memory just for data arrays; i.e., the special setting of 64K bytes for data and 64K bytes for program should be used.

3.5 Glossary of the Program Variable Names

hfdc0	- value of the high-pass filter at (π,π) for zero mean.
hfdcmax	- value of the high-pass filter at (π,π) for mean value=xmax.
hwsize	- half of the window size.
k	- contrast gain function.
ℓ	- nonlinearity applied to the mean value.
ℓfdc0	- value of the high-pass filter at (0,0) for zero mean.
ℓfdcmax	- value of the high-pass filter at (0,0) for mean value=xmax.
mn	- array of mean values.
numpt	- number of points in k,ℓ and mn functions.
var	- variance of Gaussian window.
vardc0	- thickness of Gaussian-shaped high-pass filter for zero mean.
vardcmax	- thickness of Gaussian-shaped high-pass filter for mean value-xmax.
wtype	- shape of window.
xmax	- maximum value in the image.

4. MULTI-PROCESSOR ARCHITECTURES FOR IMAGE PROCESSING

During the last half of FY82, we have been exploring several issues related to the development of a processor architecture suitable for image processing problems. Typical image processing problems involve data sets composed of several hundred thousand pixels (picture elements) and large amounts of computation (several hundred processor instructions per pixel). Depending upon the application, the computation may have to be carried out in real-time at TV frame rates (30 per second) or fast enough to permit comfortable interaction for a human operator (1 second elapsed time). It is clear that an image processing architecture must be capable of supporting rapid computation on large data sets.

In image processing, as opposed to other multi-dimensional signal processing applications, most operations tend to be local; that is, the processing of widely separated parts of the image is independent. Global operations, such as the 2-D Fourier transform of an entire image where each output value depends on all of the input pixel values, are rarely used in typical image processing operations. For this reason, it is possible to consider the implementation of most processing operations by a set of processors working in parallel on different parts of the image with a minimal amount of communication. To be truly useful, however, such a multi-processor architecture must be capable of handling the few (but important) exceptions that may occur in any particular application.

A very high-level diagram of the multi-processor architecture we plan to pursue is shown in Figure 4-1. It consists of 16 nodal processors connected by a communications network. The number of processors is somewhat arbitrary from an architectural point-of-view but was sized based on preliminary computational requirements and was chosen to be a power of two in order to permit the use of a butterfly communication network. The system input/output (I/O) port is shown as being part of the communication network, but it may be necessary to alter this to provide the required I/O bandwidth. A thorough study of the system I/O remains to be done.

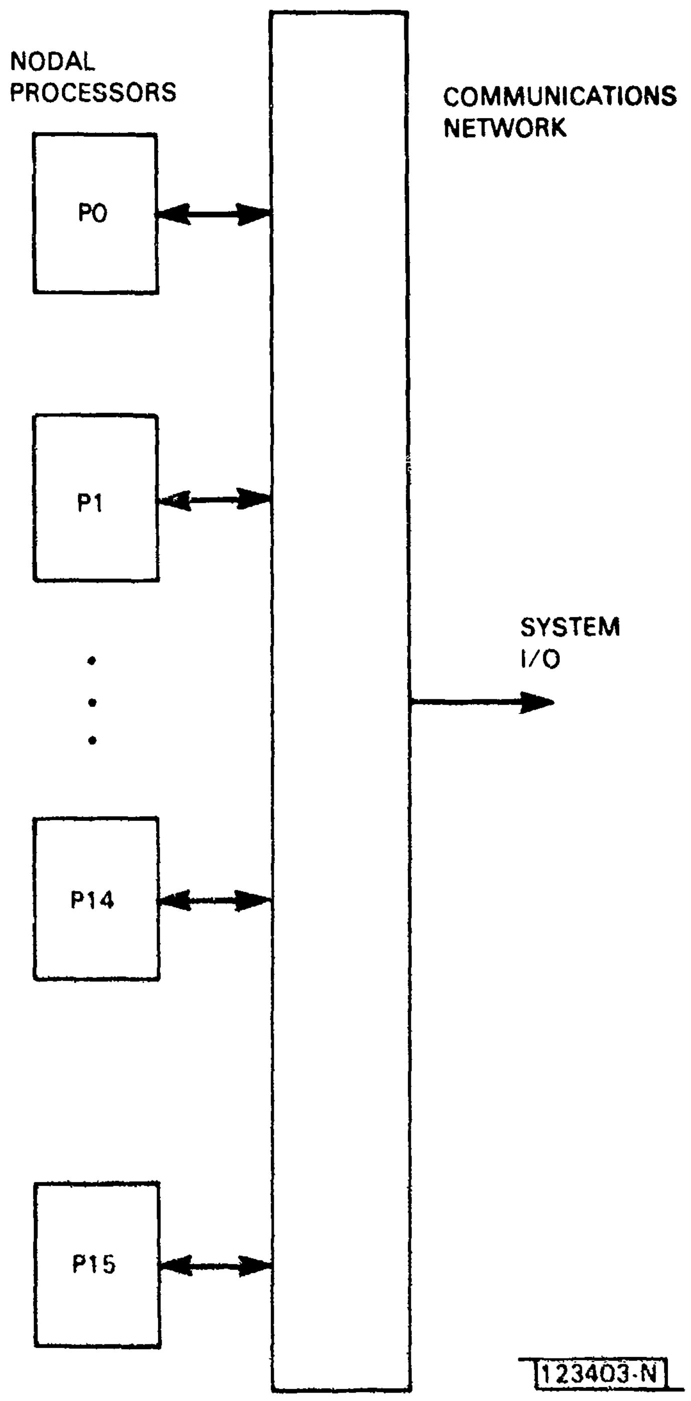


Fig. 4-1. A high-level block diagram for a multi-processor system.

There are several possibilities for the communications network. The most obvious choice is a high-speed data bus. Typical busses used in conventional computers have bandwidths on the order of 1-2M bytes/second, but this is not fast enough to satisfy the bandwidth requirements of a 16-processor system. It is conceivable that an advanced bus based on RF coaxial cable or optical fibers could be developed to satisfy the bandwidth needs which we estimate to be roughly 100-200M bytes/second.

In FY82 we have begun investigating the butterfly network structure shown in Figure 4-2 for use as the interprocessor communication network. The network is shown unfolded so that processor outputs are located on the left and processor inputs are located on the right. This network allows communication between any two nodal processors and permits many sets of conversations to take place in parallel. In another project, a four-processor system was constructed using Motorola MC68000 microprocessors and a similar butterfly network.

Several important system-level questions remain to be addressed. We have already mentioned the question of system-level I/O to an outside host computer or data source. The significant issue of how to make the programming (and controlling) of a multi-processor system appear straightforward also needs to be examined in detail. A multi-processor system that is fast but difficult to program will not be very useful to the image processing community.

In the following sections several architectural features for a nodal processor will be discussed. These features are worth developing and refining because they show promise of helping achieve the dual goals of high computational throughput and ease-of-programming. The architectural study is, however, far from complete, and subsequent ideas and developments may alter or eliminate some of the architectural features discussed below.

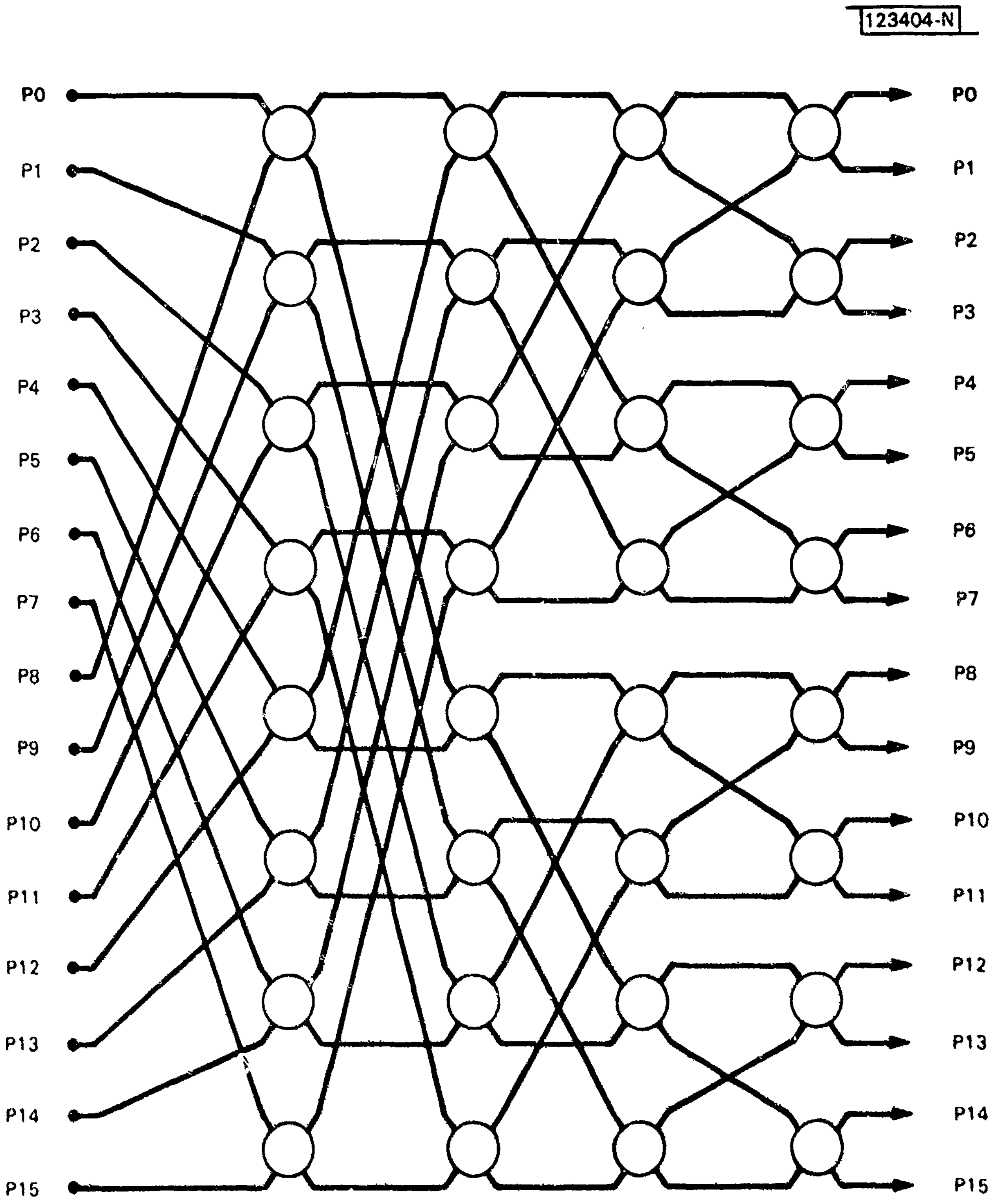


Fig. 4-2. A 16-processor butterfly network. Each circle represents a 2-input, 2-output switch that can pass signals straight or crossed.

4.1 Ground Rules for Developing a Nodal Processor Architecture

Most of our work in this area during the last quarter of FY82 has been directed toward the architecture of a nodal processor. As a ground rule, we have tried to separate architectural issues from implementational issues. Some architectural principles may ultimately have to be compromised in the interest of efficient and timely hardware implementation. However, our objective thus far has been to develop the best architecture possible without any preconceptions about implementation details. Furthermore, the technology to support the underlying implementation of a particular architecture is constantly evolving. Many computer manufacturers use the same architecture (for software compatibility) but constantly upgrade the implementation to deliver systems with improved performance/cost characteristics. With the advent of VLSI and VHSIC technology, implementation techniques will undoubtedly change once again. The development of an image processing architecture may influence future chip sets, thus permitting the implementation of architectural features that may have to be compromised with today's technology. Therefore, we have decided to concentrate initially on developing the right architecture without implementational constraints.

Processors can be compared along many dimensions: speed, efficient use of memory, ease of programming, instruction set sophistication, word length, size, weight, power consumption, reliability, etc. For now, we have taken the attitude that speed and ease of programming are most important, that the instruction set is important as an influence on the speed (minimizing the number of instructions required for some computation) and ease of programming, and that "memory is cheap." It is permissible to use memory inefficiently if it permits faster execution or simplifies programming.

The easiest way (up to a point) to build a fast processor is to use the fastest available logic family for the implementation. For a fixed architecture, the basic machine cycle time will determine the speed of execution. There are mitigating concerns, of course. Fast logic, in general, implies more power consumption, more demanding board layout, lower-level integration, and less reliability. It will probably be necessary to compromise between raw logic speed and other measures such as integration level or power consumption.

4.2 Architectural Requirements for a Nodal Processor

Based on the premise that the nodal processors in a multi-processor image processing system will have to be fast computers in their own right that are capable of handling large arrays, we can begin to outline some of the architectural requirements for the nodal processors. Traditional high-speed array processor architectures have been very "horizontal," using techniques such as separate program and data memories; separate hardware for address computation and index register manipulation; and pipelined fetch, decode, and execution of instructions. The nodal processor architectures that we are now studying will use many of the same techniques, provided that they do not impair the ease with which the machine can be programmed.

Since the nodal processors will be handling large arrays of image data, array access must be very efficient. Similarly, it is important to provide a mechanism for controlling program loops that accrues very little overhead.

Many processor architectures are capable of dynamically allocating data memory upon each invocation of a subroutine, procedure, or function. This capability permits re-entrant subroutines, efficient interrupt handling, and shared instruction code (although this is probably not important for this application). In addition, dynamic memory allocation should result in a more efficient use of the data memory than static allocation would, since it permits the "time-sharing" of memory. Unless we uncover a sound argument against it, dynamically allocated memory will be an architectural requirement for the nodal processors.

To permit simple, straightforward programing, the nodal processor architecture should efficiently support a high-level language such as "C". The machine language itself should be relatively high level so that assembly language programing (machine mnemonics plus macro-instruction capability) is conducted at a high level. It is important to isolate the programmer as much as possible from the particular implementation of the nodal processor. The programmer should be concerned with specifying operations and data objects to be used as operands and not with the details of address computation or index register manipulation.

The machine language instruction set should be flexible so that it can be "tuned" for different applications. In particular, it should provide some simple array handling instructions such as clearing an array or adding two arrays. A set of a half-dozen or so such array instructions could relieve a programmer of much pedestrian code generation in an image processing application and allow him to concentrate on the important parts of the program. Machine language flexibility could be achieved by using micro-code to realize each machine-level instruction. New instructions could be accommodated by writing new micro-code, assuming that there are enough unused op-codes remaining in the instruction format. The fetching, decoding, and execution of the micro-instructions could be pipelined to increase the effective speed of the nodal processor.

The nodal processor architecture should be flexible enough to include special-purpose computational devices that may differ depending on the application. For example, a high-speed multiplier or multiplier/accumulator could be considered a "special-purpose" device, although for image processing and multi-dimensional signal processing applications it is a requirement. Other computational devices might include an FFT butterfly, a 16-point FFT, or a CORDIC rotation element. A special-purpose device may also be necessary to facilitate I/O with the outside world as well as the interconnection network. The important point is to allow sufficient flexibility to permit the nodal processors to be retrofitted with special-purpose components to help increase computational throughput for a particular application.

The issue of fixed-point versus floating-point arithmetic has yet to be decided. To some extent, it is more of an implementation issue than an architectural issue. Fixed-point arithmetic tends to be faster, but new single-chip LSI floating-point components may close the gap. Fixed-point accuracy and dynamic range can be improved by going to longer word lengths, with the concomitant increase in hardware, but then it becomes more difficult to use the single-chip 16-bit multipliers currently on the market. There are advantages and disadvantages to both data representations, and the ultimate solution may be the traditional one of supporting both. However, because of the horizontal architecture, it may be feasible to use fixed-point arithmetic for addressing, counters, loop control, and indexing and use floating-point arithmetic strictly for data computation. This would enforce the natural separation in the programmer's mind between data to be processed and variables used for data access and program control.

4.3 Three-Address Instructions

In many cases, execution of a machine instruction will take two source operands and perform an operation on them to produce a single result to be stored at a particular destination. Thus, three addresses need to be specified in a typical instruction. Some computer architectures use one of the source addresses as the destination address, while more primitive architectures use an accumulator register as one source as well as the destination. However, programs written for these architectures usually require a significant number of LOAD, STORE, or MOVE instructions that simply transfer data without doing any computation. The three-address architecture should alleviate some of this overhead, resulting in an inherently faster nodal processor. The use of three-address instructions is also consistent with the philosophy of a horizontal architecture, permitting the addresses of the sources and destination to be computed in parallel.

There are occasions, however, when the destination address is identical to one of the source addresses. Some code compaction will result if two-address instructions are included in the instruction set. On other occasions, the result of an operation needs to be stored only temporarily because it will be an operand for the next instruction. Thus, it may be prudent to provide a stack of temporary storage registers to be used as easily accessed operand sources and destinations. (Actually, a "temporary variables" stack is an implementational rather than an architectural issue. The three-address instructions do not need a stack to be useful, particularly if access to data memory can be made as fast as access to the stack.)

4.4 Array Instructions

For image processing as well as other array processing applications, it seems prudent to provide the nodal processor architecture with a small but powerful set of instructions for performing array operations. These operations would include element-by-element addition, subtraction, multiplication, and division of two arrays, and other operations such as element-by-element maximums and minimums may also be worth including. Other candidates are inner products, sum-of-elements, sum-of-elements-squared, absolute value, and 2-D convolution.

These array instructions would be implemented in micro-code to utilize the maximum speed advantage of the architecture. It should also be possible to implement other array instructions in micro-code for particular applications.

4.5 Data Addressing

An underlying assumption of the nodal processor architecture is that it must support a fairly large data memory (256K bytes-1M byte) for image processing applications. For direct addressing, this implies addresses at least 20 bits in length, and perhaps 24 bits or even 32 bits for future expansion of the data memory. With three-address instructions, the number of bits needed for specifying source and destination locations thus ranges from 60 to 96 bits, resulting in very wide instruction words. To keep the instruction width down, some computer architectures make use of address registers. These registers, which contain the addresses of the desired operands, are few enough so that they can be specified with a small number of bits in an instruction word. Operands are thus accessed indirectly. However, address registers must be saved and restored during context switches like subroutine calls and returns and interrupt servicing, and this may imply a high level of overhead.

Because it fits in nicely with the notion of dynamic memory allocation, we have been investigating a stack-oriented data memory. Data in the stack can be accessed relative to the stack pointer (which points to the top of the stack), a frame or environment pointer (which points to a memory location determined by the current program context; see Section 4.9), or a global pointer (which for simplicity may be taken as the bottom of memory). It may also be useful to provide other pointers into the stack to facilitate data access. This question is still open.

With this structure, single data items (as opposed to arrays of data items) will be generally accessed by specifying in the instruction word a pointer register and a constant offset value to be added to the contents of the pointer register to compute the effective address.

4.6 Array Data Access

In a typical computer architecture, array elements are accessed by computing an effective address, which is the sum of a base address plus an offset or index. For two- or higher-dimensional arrays defined in a high-level language, it is necessary to explicitly compute the effective address by repeated multiplications and additions. When array elements are accessed sequentially (within a loop, for example), much of this address computation can be eliminated by simply incrementing the effective address by the proper amount.

From a programmer's point of view, however, the computation of an effective address from an array base address and index values is a nuisance to be handled by a compiler or directly by the hardware. In image processing applications, we expect that a great deal of computation will use two-dimensional arrays accessed sequentially within loops. Consequently, it is important that the architecture handle the implied address computations rapidly.

We have been investigating two concepts for accessing array elements. The simpler consists of providing the necessary arithmetic capability to compute effective addresses by adding the contents of two address registers while simultaneously incrementing one of the address registers by an amount contained in a third register. This allows the base address to be contained in one address register with the periodically updated offset to be contained in the second address register. This scheme is relatively straightforward, but it forces the machine language programmer to get involved in keeping track of effective addresses rather than simply specifying an array name and the values of its indices.

As an alternative, we have been exploring the concept of array access registers (AARs). In its simplest form, an AAR contains information about an array, such as its base address in memory, the number of dimensions, the maximum number of storage cells in each dimension, and the number of bytes of memory used to contain a single storage cell, which will allow hardware to convert a request for a particular element of a particular array into an effective memory address. Going one step further, we can actually incorporate index registers, as well as the effective address corresponding to the current values of the index registers, into the AARs. This permits a separate, noninterfering set of index registers for accessing each distinct array.

The index registers can be initialized or reset by machine-level instructions specifying the AAR and the new values of the indices. In addition, instructions can specify that an array index be incremented, decremented, or zeroed out after being used. Hardware will be responsible for computing a new effective address from the altered index (or indices) and storing it back in the AAR for future use. This mode should be very efficient for loops which perform the same basic operation on all elements of an array.

The AARs can be allocated dynamically, just as data memory is. This will permit efficient use of AARs by effectively "time-sharing" them as different subroutines become active. AARs will also permit arrays to be passed as subroutine arguments in an efficient manner. If AARs are allocated by a stack, however, it implies that array data are accessed by two levels of indirection, one with respect to a stack pointer (or frame pointer) to get the appropriate AAR, and the second to use the effective address provided by the AAR. We will have to examine the AAR concept carefully to see if the doubly indirect access leads to an unacceptably slow architecture.

In theory, scalar-valued variables could also be accessed using AARs. This would provide a consistent architecture in that scalar and array variables would be accessed in a similar manner, with the programmer treating each variable as a data object with certain attributes. However, this would mean that scalars as well as arrays would require two levels of indirection for accessing. It does not seem that an extra level of indirection for scalars will improve the access efficiency that is potentially available for arrays, because scalar variables do not have indices to be manipulated or effective addresses to be computed. Nevertheless, the architectural consistency which would result may overrule this conclusion upon closer examination.

4.7 Program Control

Currently, the nodal processor architecture contains several useful instructions for transferring program control. The instructions consist of BRANCH, JUMP, CALL, RETURN, BREAK, and NEXT. The instructions may be used in both conditional and unconditional formats.

A BRANCH instruction transfers control to an instruction located at some address relative to the BRANCH instruction; that is, it is a relative jump. Conversely, a JUMP instruction transfers control to an absolute address in the instruction memory. CALL transfers control to an absolute address as well, but it also keeps track of a number of things (discussed in Section 4.9) useful for transferring control to a subroutine. RETURN allows control to be transferred back from a subroutine using the absolute address stored by CALL. BREAK and NEXT are essentially BRANCH instructions useful in controlling program loops. (They will be discussed in that context in Section 4.8.)

The conditional versions of these control-transferring instructions make use of a set of condition codes. The condition codes consist of bits that indicate whether a result from an operation was positive, negative, zero, caused a carry or an overflow, etc. The condition codes are set by an operation resulting from an instruction that had its test flag set. Codes from subsequent instructions that had their test flags set are ORed with the existing condition codes. The codes are reset (cleared) by a conditional control-transferring instruction, unless that instruction indicates (via a flag) that the codes are not to be altered. Finally, the architecture should support instructions that can store the condition codes in data memory and can restore the condition codes from data memory.

4.8 Loop Control

The image processing applications for the nodal processor will doubtlessly lead to programs containing many, relativly short, nested loops. Consequently, it is important for the architecture to support program loops in a very efficient manner. These loops will have the characteristic that the number of times they are executed is independent of the data being processed. Thus, in a typical architecture, a register is initialized to contain the number of times that the instructions in the loop will be executed. Then the register is decremented at the bottom of the loop and tested to determine whether to branch back to the top of the loop. Every time the loop is executed, the decrement, test, and conditional branch instructions are also executed. For short loops, these instructions can represent a significant overhead compared to the useful computation carried out within the loop.

For most of the loops used in image processing software, a simple architectural feature can be used to support the overhead for loop control.

Since loops are perfectly nested, a stack of loop counters can be used to keep track of the current iteration. When a loop is initialized (by a single machine-language instruction), an initial value for the loop counter is pushed on the stack. At the end of the loop, another instruction causes the loop counter to be decremented and, if it is still positive, control to be transferred to the top of the loop. The top-of-loop address can be set up at loop initialization in a stack parallel to the loop counter stack. When the loop counter finally reaches zero, the loop is exited and the stacks holding the loop counter and the top-of-loop address are popped.

Under certain circumstances, it is desirable to exit prematurely from one iteration of the loop and to begin the next. The NEXT instruction, which may be conditionally executed, performs this function. It may be desirable to exit a loop entirely before all the iterations have been completed. The BREAK instruction, also conditionally executable, can be used for this purpose.

The NEXT and BREAK instructions are essentially conditional BRANCH instructions. The NEXT instruction branches to the end-of-loop instruction, which decrements the loop counter and tests it as usual. The BREAK instruction branches out of the loop, but it must also pop the loop-counter stack and the top-of-loop address stack. The offsets needed for calculating the next instruction address can be computed by a compiler or assembler. Although it is not necessary, there may be some speed advantage to storing the transfer addresses for BREAK and NEXT instructions on stacks parallel to the loop counter stack. This implementational issue will have to be examined in more detail.

Loops in which the number of iterations is data-dependent can also be implemented with the specialized loop instructions. An infinite loop could be set up by setting an "infinity" bit in the loop counter. The loop would be exited by using a conditional BREAK instruction.

Structured programming texts argue for testing at the top of the loop rather than the bottom as a defensive programming tactic. It may be prudent to incorporate other specialized loop control mechanisms in the architectural requirements to support both top-of-loop and bottom-of-loop testing.

4.9 Subroutine Linkage

Structured software tends to have many subroutines and consequently many subroutine calls. Thus, it is important to have an efficient subroutine linkage mechanism so that the overhead for short subroutines is not too large. One subroutine linkage mechanism we have been exploring is an elaboration of those used on stack-oriented computers such as the HP-3000. At present, it actually involves seven parallel stacks.

The seven stacks are called the instruction address stack (I-stack), the data stack (D-stack), the array access register stack (A-stack), the data frame stack (DF-stack), the AAR frame stack (AF-stack), the number-of-data-parameters stack (#DP-stack), and the number-of-AAR-parameters stack (#AP-stack). Figure 4-3 is an outline of how these stacks are manipulated during subroutine calls and returns.

The simplest stack to understand is the I-stack. When a subroutine is called, control is passed to the starting address of the subroutine and the return address is pushed onto the I-stack. When a RETURN instruction is executed in the subroutine, the return address is popped off the I-stack and into the program counter.

The D-stack, the DF-stack, and the #DP-stack are closely related. At any given level of subroutine nesting, the D-stack contains the information needed for the current subroutine context. The data stack pointer, which itself sits atop the DF-stack, points to the top of the D-stack and is updated when data are pushed on or popped off the D-stack. One of the addressing modes allows data to be accessed with a negative offset relative to the data stack pointer. Data on the D-stack may also be accessed relative to the data frame pointer, which itself resides just below the data stack on the DF-stack. The data frame pointer points to the top of the parameter list for the current subroutine.

To call a subroutine, space is first allocated on the D-stack to hold any values to be returned by the subroutine. This is accomplished simply by incrementing the data stack pointer. Next, the parameters specified by the

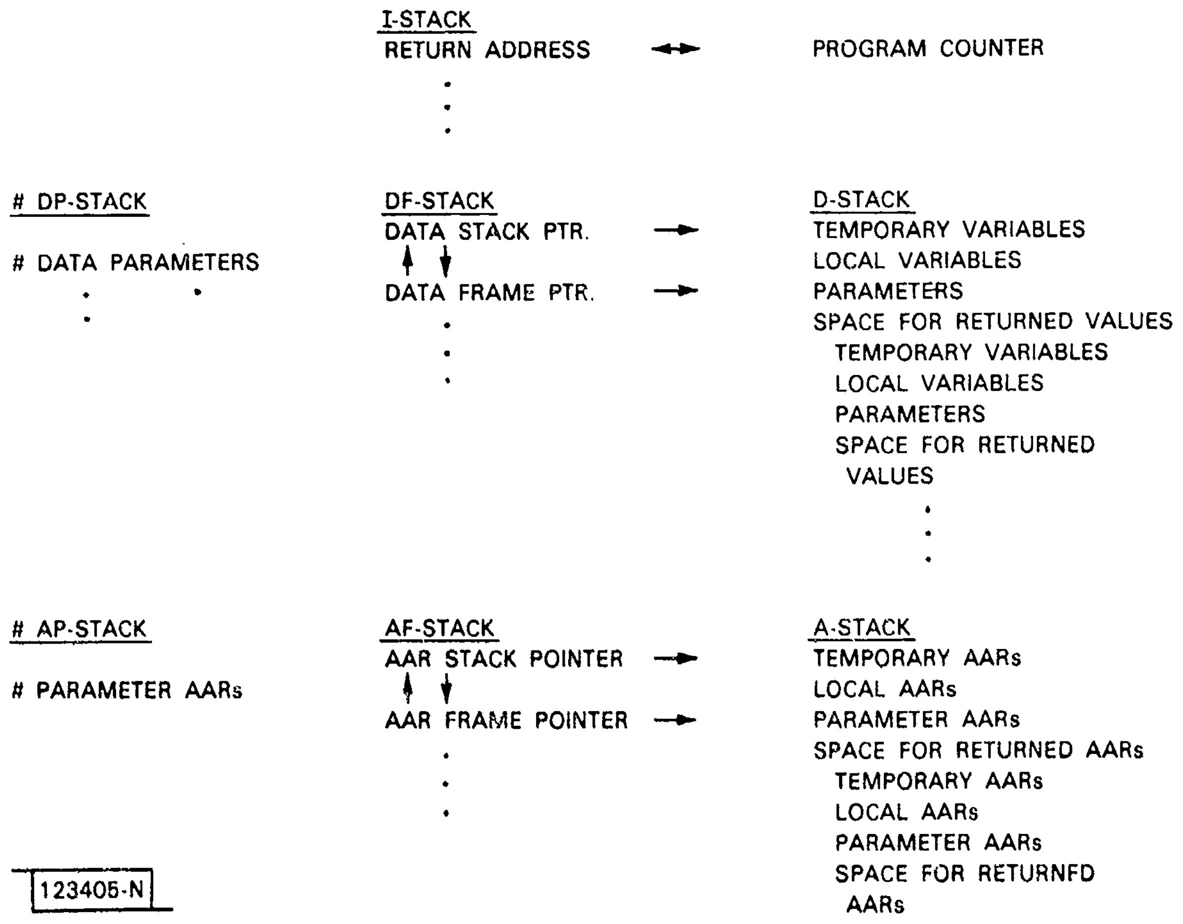


Fig. 4-3. Seven stacks used for subroutine linkage.

arguments of the CALL instruction are pushed onto the D-stack and the number of parameters is pushed onto the #DP-stack. When the CALL is executed, a duplicate copy of the data stack pointer is pushed onto the DF-stack, making the new data frame pointer equal to the old data stack pointer. After control is transferred to the subroutine, the data stack pointer is again incremented to allocate storage for dynamic variables that are local to the subroutine. Additional temporary storage may be allocated simply by pushing values onto the D-stack and updating the data stack pointer.

When a subroutine RETURN is executed, the DF-stack is popped restoring the old data frame pointer and the data stack pointer to the state that existed after the subroutine parameters were pushed on the D-stack but before the subroutine was actually called. The number of data parameters is then popped off the top of the #DP-stack and subtracted from the data stack pointer to effectively de-allocate the memory used to pass parameters to the subroutine.

The other three stacks--the A-stack, the AF-stack, and the #AP-stack--are handled in a fashion analogous to their data stack counterparts. Rather than holding data values, however, the A-stack contains the names of AARs, which in turn point to arrays of data values.

As we mentioned earlier, it is possible to use the AAR concept to access scalars as well as arrays. If this were done, the resulting architectural consistency would permit the reduction from seven stacks to four stacks. The separate D-stack, DF-stack, and #DF-stack would not be needed.

4.10 Input/Output

There are two aspects of nodal processor I/O: communication with the outside world and communication with other nodal processors in the multiprocessor system. Here we shall limit our discussion to the latter. In terms of architectural requirements, we want the nodal processor to handle the protocols for I/O as little as possible for two primary reasons. First, we do not want to slow the nodal processor with bookkeeping tasks related to the I/O and second, we want the flexibility to interface improved interprocessor connection networks easily as they are developed.

These requirements argue for a direct memory access (DMA) capability with a separate I/O processor to handle the necessary communications protocols. The nodal processor should simply be able to request that data be sent to another processor or be received from another processor. In addition, it should be possible to interrupt the nodal processor to inform it that data has been received from another processor. A great deal remains to be done in the specification of architectural features to support both interprocessor I/O as well as system I/O.

4.11 Summary of the Multi-Processor Architecture

The architectural specifications for a multi-processor system are far from complete. As described in the previous sections, we have begun looking at some of the architectural requirements for the nodal processors. In FY83 we plan to continue, with the immediate goal of specifying a nodal processor instruction set and architectural features to permit its efficient implementation. Major areas of concern include I/O, array access, loop control, and subroutine linkage. Ease of programming, flexibility of the instruction set, and the ability to incorporate special-purpose computational hardware are also important.

The 16 features listed below contribute to our goals of high computational throughput and ease of programming. Much more study needs to be done and further development may alter the desirability of these features. Nevertheless, at this stage, these are important specifications of a system architecture for image processing applications.

1. Multi-processor architecture consisting of 16 nodal processors communicating via an interprocessor communications network.
2. Butterfly-type communications network.

3. Modular interface between nodal processors and communications network.
4. "Horizontal" architecture for nodal processor.
5. Dynamic allocation of data memory using data stacks.
6. Thirty-two-bit-wide data words.
7. Flexible instruction set implemented in micro-code.
8. Pipelining of instruction fetch, decode, and execution.
9. Support for special purpose computational elements.
10. Three-address instructions.
11. Array handling instructions.
12. Stack-oriented data access with separate hardware for address computation.
13. AARs for rapid access of array elements.
14. Specialized instructions for loop control.
15. Stack-oriented subroutine linkage.
16. Separate I/O processor to support DMA communications.

REFERENCES

[1] Semiannual Technical Summary, Multi-Dimensional Signal Processing Research Program, Lincoln Laboratory, M.I.T. (31 March 1982), DTIC AD-A118186/6.

[2] A.W. Drake, "Fundamentals of Applied Probability Theory", McGraw-Hill Book Company.

[3] C.W. Therrien, "On the Relation Between Triangular Matrix Decomposition and Linear Prediction", to be published.

[4] M.P. Ekstrom, J.W. Woods, "2-D Spectral Factorization with Applications in Recursive Digital Filtering", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-24, No. 2, pp. 115-178 (April 1976).

[5] T. Marzetta, "The Minimum Energy-Delay Property of 2-D Minimum-Phase Filters", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-30, No. 4, pp. 658-659, (August 1982).

[6] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw-Hill, New York, 1965.

[7] "Adaptive Filtering for Image Enhancement", T. Peli and J.S. Lim, Optical Engineering, January/February 1982.

[8] T. Peli and T.F. Quatieri, "Exposing Objects Under Light Cloud Cover by Adaptive Homomorphic Filtering," Technical Report 587, Lincoln Laboratory, M.I.T. (6 January 1982), DTIC AD-A112338/9.

APPENDIX

DERIVATION OF CONSTANT FALSE-ALARM RATE DETECTION

In this appendix, we derive the general functional form of λ to maintain a constant level of significance in Equation 3, regardless of the background statistics (i.e., regardless of the covariance K). Without loss of generality, we shall assume a zero-mean process. Then from Equations 2, 3, and 15a, we have:

$$\alpha = \int_{p(\underline{x})<\lambda} \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}\underline{x}^T K^{-1}\underline{x}\ d\underline{x}\right]$$

$$= \int_{p(\underline{x})<\lambda} \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}(L^{-1}\underline{x})^T D^{-1}(L^{-1}\underline{x})\right] d\underline{x} \qquad \text{(A-1)}$$

Now, let

$$\underline{e} = L^{-1}\underline{x} \qquad \text{(A-2)}$$

so that, since L^{-1} has unit diagonals, using the method of Jacobians [6], we have,

$$d\underline{e} = d\underline{x} \qquad \text{(A-3)}$$

Thus, substituting Equations A-2 and A-3 into A-1, we obtain

$$\int_{p(L\underline{e})<\lambda} \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}\underline{e}^T D^{-1}\underline{e}\right] d\underline{e} = \alpha \qquad \text{(A-4)}$$

PRECEDING PAGE BLANK-NOT FILMED

Furthermore, we have (since $D^{-1/2}$ is diagonal):

$$\int_{p(L\underline{e})<\lambda} \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}(D^{-1/2}\underline{e})^T(D^{-1/2}\underline{e})\right]d\underline{e}$$

$$= \int_{p(LD^{1/2}\underline{\hat{e}})<\lambda} \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}\underline{\hat{e}}^T\underline{\hat{e}}\right]|D|^{1/2}d\underline{\hat{e}} = \alpha \qquad \text{(A-5)}$$

where we have used the substitutions:

$$\underline{\hat{e}} = D^{-1/2}\underline{e} \qquad \text{(A-6a)}$$

$$d\underline{\hat{e}} = |D|^{-1/2}\, d\underline{e} \qquad \text{(A-6b)}$$

Noting that $|K| = |D|$ [3], we have from Equation A-5,

$$\int_{p(LD^{1/2}\underline{\hat{e}})<\lambda} \frac{1}{(2\pi)^{N/2}} \exp\left[-\frac{1}{2}\underline{\hat{e}}^T\hat{e}\right]d\underline{\hat{e}} = \alpha \qquad \text{(A-6c)}$$

Note that the integrand in Equation A-6 does not depend on the statistics of $\underline{x}$. Furthermore, the boundary of our transformed critical region is, from Equation A-6, given by the equation:

$$p(LD^{1/2}\underline{\hat{e}}_o) = \lambda \qquad \text{(A-7)}$$

which, from Equations 2 and 15a, can be expressed as

$$\frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp\left[-\frac{1}{2}(LD^{1/2}\underline{\hat{e}}_o)^T L^{-1^T} D^{-1} L^{-1}(LD^{1/2}\underline{\hat{e}}_o)\right]$$

$$= \frac{1}{(2\pi)^{N/2}|K|^{1/2}} \exp -\frac{1}{2}\underline{\hat{e}}_o\underline{\hat{e}}_o = \lambda \qquad \text{(A-8a)}$$

or

$$-\frac{1}{2}\hat{\underset{\sim}{e}}_o^T\hat{\underset{\sim}{e}}_o = \ln[(2\pi)^{N/2}|K|^{1/2}\lambda] \qquad \text{(A-8b)}$$

Therefore, to maintain a constant level of significance, we must have:

$$\ln[(2\pi)^{N/2}|K|^{1/2}\lambda] = \text{constant} \qquad \text{(A-9a)}$$

or

$$\lambda = \frac{C}{(2\pi)^{N/2}|K|^{1/2}} \qquad \text{(A-9b)}$$

as we had proposed earlier in Equation 12.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-82-097	2. GOVT ACCESSION NO. AD-A126 333	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Multi-Dimensional Signal Processing Research Program		5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Summary 1 April — 30 September 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dan E. Dudgeon		8. CONTRACT OR GRANT NUMBER(s) F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element No. 62702F Project No. 4594
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center Griffiss AFB, NY 13440		12. REPORT DATE 30 September 1982
		13. NUMBER OF PAGES 112
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

target detection
adaptive contrast enhancement
image processing architectures

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This Semiannual Technical Summary covers the period 1 April through 30 September 1982. It describes the significant results of the Lincoln Laboratory Multi-Dimensional Signal Processing Research Program sponsored by the Rome Air Development Center, in the areas of image segmentation, classification, target detection, and adaptive contrast enhancement.

DD FORM 1473 1 Jan 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*