INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the original text directly from the copy submitted. Thus, some dissertation copies are in typewriter face, while others may be from a computer printer.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyrighted material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is available as one exposure on a standard 35 mm slide or as a 17″ × 23″ black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. 35 mm slides or 6″ × 9″ black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Accessing the World's Information since 1938

300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

Order Number 8821093

Non-linear processing of signals and images

Kasparis, Takis C., Ph.D.

City University of New York, 1988

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

NON-LINEAR PROCESSING OF SIGNALS AND IMAGES

by

TAKIS KASPARIS

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1988

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

3/2/88
Date

George Eidelman
Chair of Examining Committee

3/2/88
Date

Jacques E. Benvenuste
Executive Officer

Professor S. Basu

Professor M. Conner

Professor L. Roytman

Professor N. Scheinberg

Professor R. Tolimieri

Dr. J. Keybl
Supervisory Committee

The City University of New York

Abstract

NON-LINEAR PROCESSING OF SIGNALS AND IMAGES

by

Takis Kasparis

Adviser: Professor George Eichmann

Lines and edges are probably the most important characteristics of an image. Many image processing techniques utilize lines and edges. The contributions of this work are in three major areas of image processing and they all involve lines or edges. The first area is that of image filtering using non-linear edge preserving filters. Among them, rank-filters with median filters as a special case are the most important. Median filters (MF) are used to remove impulsive noise from images while preserving edges. In this thesis, an extension of the MF, the Vector Median Filter (VMF) is introduced. As opposed to the MF, the VMF outputs for each window position a number of data elements. Just like the MF, the VMF filters impulses while preserving edges. Its principal advantage is superior computational speed, with performance similar to that of the MF. Deterministic and statistical properties will be examined, as well as two-dimensional extensions. A fast VMF algorithm is also presented. In addition, a novel

filtering scheme using MF's and linear transforms is introduced.

The second area of this work is in texture analysis and classification. Texture is one of the important image characteristics and is used to identify objects or regions of interest. Texture classification techniques are either statistical or structural. In this work, a new structural approach based on line detection is introduced. This classification is based on the relative orientation and location of the lines within the texture. With proper normalization, the classification is independent of geometrical transformations such as rotation, translation and/or scaling. Associative memory encoding is used as post-processing decision maker. Iterative associative memory encoding is introduced for additional accuracy.

The third area in which contributions are made is image segmentation. Image segmentation is a highly scene dependent and problem dependent decision making or pattern recognition process. Based on edge and line detection, an image segmentation technique for images containing polygonal shapes is introduced. The method is based on the fact that the sides of a polygon form a closed contour. Stray lines and noise are effectively suppressed.

ACKNOWLEDGEMENTS

I would like to thank my mentor, Professor George Eichmann, for his guidance and support. I would like to thank Professors Norman Scheinberg and Joe Barba for the use of their equipment and for providing me with all necessary information. Also, I would like to thank Ms. Abigail Cousins and Ms. Yvonne Blackman for their first class secretarial services. Finally, the grant supports of the City University of New York Faculty Research Award Program and the U.S. Air Force Office of Scientific Research are gratefully acknowledged.

List of Presentations and Publications From this Thesis

"Image Processing Using Vector Median Filters." Presented at the Annual Meeting of the Optical Society of America, Washington, D.C., October 1985.

"Vector Median Filters", Signal Processing, Vol. 13, No. 3, October, 1987.

"Texture Classification using the Hough Transform" SPIE Proceedings, Vol. 638, April 1986.

"Topologically Invariant Texture Descriptors", Computer Vision, Graphics and Image Processing (accepted for publication)

"Knowledge Based Image Segmentation" (with N. Marinovic) SPIE Proceedings, Vol. 726, Oct. 1986.

"A Fast Two-Dimensional Vector Median Filter Algorithm" Presented at the Annual Meeting of the Optical Society of America, Rochester, N.Y., October 1987.

"Pattern Classification Using an Associative Memory" Presented at the Annual Meeting of the Optical Society of America, Rochester, N.Y., October 1987.

TABLE OF CONTENTS

LIST OF TABLES

List of Illustrations

1. INTRODUCTION

Lines, and more generally edges, play a very important role in images. Primarily high frequency content is the key factor that defines detail in images, as well as other fundamental concepts such as shape, texture and regions. Much work in different areas of image processing is based on information extracted from edges and lines. Several image enhancement methods increase image detail by enhancing edges and lines. Many image segmentation and classification problems are approached using edges and lines. The problem of edge detection has been continuously under investigation and many linear and non-linear edge detectors are available with different advantages and disadvantages.

Noise removal from images is another very important problem in image processing. A very desirable feature of noise removing filters is to preserve edges. If a noise smoothing filter also smooths the edges, then the filtered image seems to be blurred, which is very undesirable. Linear filters that operate mainly on the frequency domain fail to perform well in this direction. Even though their behavior is well understood, their performance on images is unsatisfactory. For example, a low-pass averaging filter is a good noise suppressor, but it also smooths edges and blurs

the image. The basic reason for this poor performance is that both noise and edges have high frequency components. Since a low-pass filter cannot distinguish the signal and noise components, it will suppress all the high frequencies, including the signal. On the other hand, non-linear filters that operate on the spatial domain have superior performance over their linear counterparts. However, because of their non-linear nature, they are very difficult to describe, and their behavior is not well understood. Superposition does not hold any more and the properties of the filter could vary from signal to signal. Statistical analysis is also very difficult.

In the class of nonlinear filters, rank filters are of particular interest. A one dimensional rank filter slides a window along a data array. At each window position, the window elements are sorted according to their numerical value into a list. The rank filter output is that element that falls at a predetermined position within the list. If the filter selects an element at either end of the list, then it corresponds to either a MIN or a MAX rank filter. [1] Another popular rank filter, the median filter [2] (MF), assumes an odd number of data window elements and selects the 50th percentile element in the data window. Median filters preserve an edge in a signal while they

filter out impulses whose duration is less than (N-1)/2, where N is the length of the window. This type of filtering is not possible with a linear filter such as a low pass filter which filters both signal and noise components. As a result, MFs are useful in impulse noise elimination applications.

Rabiner, Sambur and Schmidt [3] have applied median filtering to speech processing. Velleman [4] has investigated the sinusoidal response of MFs. Median filtering techniques can be extended to multi-dimensional signals. Here, the window has both a size and shape. With the window centered at a particular pixel in the multi-dimensional image, the elements within the window are sorted and the median value is used to replace the center element. Huang [5] has developed a fast two dimensional MF that is based on a histogram calculation. Pratt [6], and also Narenda [7] who also examined the real-time implementations of this scheme, have used successive one dimensional MFs, filtering first the horizontal and then the vertical lines, to smooth a two dimensional image. Tukey [8], who is credited to be the first to suggest median filtering as a signal processing scheme, also suggested the following smoothing procedure: do repeated median filtering on a signal until the original

signal becomes a root signal. A root signal remains invariant under median filtering. Root signals have been used in bandwidth compression coding [9].

Along with the median filter, a number of median-type filters have been suggested [10-12]. The aim of these filters is either to reduce the computational complexity without too much performance degradation, or to be more effective on certain types of noise. An analog version of the discrete MF has also been recently introduced [13]. Recently, using VLSI technology, small window size MFs were fabricated on a single chip [14, 15, 16]. A survey of the MF properties is available [2]. Additional references on MF properties are [17-20].

In this work, a new median-type filter, a vector median filter (VMF), is introduced. The advantage of the VMF is that, with visual performance comparable to that of the MF, it results in reduced processing time. Because it is a multi-parameter filter, with the MF as its special case, it offers a wider choice of filtering possibilities. After introducing this new filter, its deterministic and statistical properties are examined and compared to MF properties. Results of computer simulations will also be presented.

As was mentioned earlier, edges and lines are very important in shape definition and texture. Texture is one of the most basic properties of the visible surface and it is used to identify objects or regions of interest in images. The visible image texture may be due to changes on the surface of an object, may be segments of lines as on a brick wall, or could simply be due to a collection of objects as in an aerial photo. Texture features are useful for both image analysis and classification, as well as in scene segmentation and identification, such as in an image understanding system applied to robot vision, industrial inspection, photo analysis etc. [21, 22]

The problem of texture classification has been widely studied. Texture classification techniques are basically statistical, structural or both in a combined fashion.

In this work, a new structural line detection approach is presented. The rationale for this approach is that regular textures consist of an arrangement primarily of line structures appearing periodically in the texture. Furthermore, it has been postulated that a human eye classifies structures based on line detection, i.e. the human eye is primarily a line detector.[23,24] In our approach line detection is accomplished using the Hough technique.

The Hough transform (HT) maps line segments into a point in the transform domain. In the HT domain, the line length can also be obtained. The method consists of calculating the texture HT and extracting features used for classification. Texture features that can be obtained from the Hough domain are principal directions of lines in the texture, periodicity and line separation in each direction etc. With proper normalization, classification is not affected by geometrical transformations such as rotation, translation or scaling.

Lines and edges can be utilized for image segmentation and shape recognition. The class of images containing polygonal shapes is of particular interest. Since polygonal shapes are formed by straight line segments, edge and line detection is a primary preprocessing step. The HT is a good line detector, but it fails to provide all the geometrical information about a line segment, such as ending points. Based on edge and line detection, a method is proposed for polygonal shape image segmentation. The rationale of the method is that since polygons are closed contours, line segments on the contour of one polygon should have common points. Grouping of all the common points reveals the vertices of the polygon, and reconstruction is possible. Masks can be generated that allow masking-off the polygon from the original image, and effectively the image is

segmented into regions containing only one polygonal shape. Noise and stray lines which are also considered as noise are successfully suppressed. Furthermore, since for each polygon information about the contour and the vertices is known, knowledge about its shape can be obtained. The preprocessing steps are edge-detection followed by line detection using the HT. Information extracted from the HT domain is utilized in the edge image for vertex detection. Mask generation and segmentation is then possible. The method can be generalized for other shapes with different preprocessing steps.

The organization of this thesis is as follows: In sect. 2 background on previous work will be presented. In sect. 3 the Hough transform is introduced. Since the HT is not one of the commonly used transforms, a separate section is devoted to the definition and its properties. The remaining sections are contributions of this work to the related areas. More specifically, sect. 4 includes contributions in areas of median-type filtering. That includes 1-D and 2-D vector median filters, a fast algorithm for 2-D VMF filtering, and signal restoration using MF's and linear transform. Section 5 includes contributions in areas of texture classification. In that section an algorithm for texture classification which is based on the HT is

presented. In section 6 a new application of associative memories as a post-processor of classification algorithms is introduced. The new scheme of iterative associative memory encoding is introduced as sect. 6.a. Sect. 7 is on a new polygonal shape image segmentation algorithm which is HT based. Finally, in sect. 8 all the experimental results will be presented, and sect. 9 is an overall summary of the work.

2. BACKGROUND

In this section background on previous work will be presented. This will cover background on median filters, textures and texture classification and on image segmentation, beginning with the median filter.

The median filter [2] is defined as follows: let x be an input sequence and y be the output sequence. Then, the output MF_{2n+1} at position i is

$$y_i = Med_n\ [x_{i-n}, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{i+n}]$$

where 2n+1 is the filter window size. In this definition, the filter output y_i depends on both the past and on the future input sequence values. In order to be able to take into account end effects, n elements are appended to both ends of the original data sequence. The values of the appended elements are the same as the first element value to the left and last element value to the right of the original data sequence, respectively. Under these conditions, the first and last elements of the original data sequence will not change under median filtering.

For a window size of N=2n+1 we define the following regions: a constant neighborhood is a region of at least n+1 consecutive points having the same value. Monotonic regions are regions where the signal is either increasing or decreasing. An impulse, also called a spike, is at least one but no more than n non-zero consecutive points of the same value, superimposed on a constant neighborhood. If an impulse is contained within a window, since it is narrower than n, the median value of the window cannot be an impulse element, and therefore the impulse will be eliminated from the output. If a pulse is wider than n+1 points, then this pulse will be preserved. Since a step can be considered as a wide pulse, it will also be preserved. Frequently, in more general signals and especially images, "roof type" edges are present. These type of edges are not preserved by MFs. For example an MF would symmetrically clip the peaks of a symmetric triangular wave.

In a similar fashion, the sinusoidal characteristics of the output of an MF due to a sinusoidal input are not always preserved. It has been shown [2, 4] that the small window size MF sinusoidal response has rather large sidelobes. To reduce these undesirable sidelobes, the use of a cascade of even window size MFs has been suggested. The output of an even window size MF is the average of the two center

elements in the window.[2] However, this filter nc longer preserves step inputs. Another way to decrease sidelobes is to take the average of two different window size MFs.[2] For example, the filter $\frac{1}{2}(MF_3 + MF_5)$ has shallower nulls than those of either MF_3 or MF_5. In the following paragraphs background on textures and classification is presented.

Texture may be considered either as a pattern of different spatial intensity arrangements, or as a basic periodically or quasi-periodically repeated local pattern. This definition is applicable to line patterns, such as ruled-line arrays, tiling patterns etc. Texture also relates to the spatial size of the tonal primitives of an image. A larger size tonal primitive shows a coarse texture, while a smaller size tonal primitive indicates a fine texture. The texture autocorrelation function, basically its spatial frequency content, determines the size of tonal primitives. An alternative approach is to view texture not in terms of spatial frequency content but in its edgeness per unit area. [25] Coarse textures have a small number while fine textures have large number of edgeness per unit area. Rosenfeld and Thurston [26] used the average value of the texture gradient as a local image property. Sutton and Hall [27] extended this idea by considering the

gradient to be a function of the distance between the pixels. This last gradient method is directly related to the texture autocorrelation function.

The problem of texture classification has been studied for many years and a number of approaches have been developed. Analysis and classification are based on texture features which are derived using either statistical or structural methods, or both in a combined manner. Statistical approaches view texture regions as a sample of a two-dimensional stochastic process. This process is describable by its statistical parameters. This model-based formulation is well suited for natural textures consisting of segments of multigray level images such as grass, sand, wool etc. Structural approaches are based on the view that textures are made up of primitives which appear in nearly regular repetitive spatial arrangements. To structurally specify a texture, the primitives and the placement rules [28] must be described.

Some of the statistical approaches include autocorrelation functions, [25] textural edgeness, [26] spatial gray level co-occurrence probabilities [29] and gray level run lengths. [30] The Fourier power spectrum of a texture gives essentially its statistical information, [31] although

it has been used to determine some structural descriptions, such as its periodicity and its directionality. [32]

Structural approaches are based on the view that within the texture there is a regular repetition cell. To describe texture, one needs to prescribe the resolution cell and the various placement rules with which the texture is formed. This approach is especially suited for regular, periodic textures. Zucker [29, 33] suggested that natural textures be viewed as a distortion of an ideal regular structure. Carlucci [34] described a texture model using line segments and/or polygons as primitives in which the placement rules are given syntactically in a graph-like language. Lu and Fu [35] presented a tree grammar syntactic approach for texture. The basic difference, in various structural approaches, is the choice of primitives. These can be either pixels, [35] gray level peaks, [36] line segments, [34] or tiles. [37] Statistical methods, with their primitives such as edgeness per unit area or run lengths, can be combined with structural approach. In other structural approaches, the primitives used are the average edge separation in different orientations [38] and the repetition of edges in different orientations, by the calculation of edge repetition arrays. [39]

A survey of texture analysis methods can be found in [28,40,41]. Several examples of natural textures can be found in Brodatz's photographic album of natural textures. [42] The remaining paragraphs are a background on image segmentation.

The image segmentation problem can be formulated as follows: Given an image defined on a sampling lattice X, find a partitioning of X into maximal regions such that a certain property that defines the segmentation is true when evaluated on each region. This segmented image is then used as input for higher level image understanding tasks.

Segmentation of images into regions is an important step in image analysis. A region is a group of pixels sharing some consistent characteristics. Regions in the image correspond to surfaces in a scene. Along with lines, regions are the atomic elements from which descriptions of the image are constructed.

Segmentation of the image into regions and subsequent analysis can be done in several ways: from simple 'pixel-by-pixel' classification methods using gray levels and local feature values, to more complex 'split and merge' techniques using statistical homogeneity tests. The goals

of region segmentation can widely differ. One may be interested in all the distinct regions in the image or one may be interested in extracting objects specified by a user. Knowledge about the class of images to be processed and the tasks to be performed plays an important role. Approaches which try to incorporate such knowledge have been attempted are either: structural, linguistic, or semantic.

Basic region segmentation techniques through the use of signal level knowledge are divided into three classes:

1) Using local spatial criteria for region merging,
2) Using global spectral information for region splitting,
3) Using both a combined manner.

Region growing historically is among the first of region segmentation approaches. It basically consists of first splitting the image into small homogeneous pieces, and then merging them step by step to build up regions based on local spatial analysis. Statistical measures are used between the distributions of intensities in the regions. Pavlidis [43] defined region segmentation as a problem of approximating the 2-D image by a set of predefined functions. Also Pavlidis and Horowitz [44] developed the

'split and merge' principle, which is to merge adjacent regions having similar approximations and to split those regions that have large error norms. Region segmentation by global spectral distributions is based on the use of histograms or distributions as a tool for detecting the possible existence of regions in the image. The third approach tries to incorporate both spatial and spectral information by adding postprocessing to the histogram based segmentation. A generalization in this line is the use of relaxation labeling [45].

Region segmentation by the use of signal level knowledge is useful for describing 2-D image features in term of regions. However, for the interpretation of images, we must assign semantic names to regions. For this purpose, semantic knowledge is needed. We can cascade a process of region segmentation at the signal level and a process of labeling semantic names to regions. Graphs were used to represent relational description of objects, and constrained tables on object-object and object-property relationship. Physical-level knowledge is necessary to obtain the scene domain cues either from the picture domain cues or from the image. At present, little is known about the physical-level of knowledge. What has been accomplished mainly concerns

the image formation process under fairly limited and idealized assumptions. Surveys on image segmentation are available [45,46,47], where additional references can also be found.

3. THE HOUGH TRANSFORM

The Hough transform [48] (HT) is a method for detecting curves that can be described by a number of parameters, such as lines, [49] circles, [50] parabolas [51, 52] etc. The HT is used to compute the locus, in parameter space, of the set of curves passing through a point in the image plane. In the special case where line detection is desired, since two parameters are sufficient to specify a straight line, the parameter space is two-dimensional. Hough [48] chose to use the slope and the intercept as the two parameters. Because these parameters are unbounded, the application of the technique is complicated. Through the so-called normal parametrization, a bounded parameter set is obtained. As illustrated in Fig. 1, a straight line (L) is specified by the angle formed by the x-axis and the normal to L, and its distance p measured from the origin to the point where the normal and L intercept. The equation of the line L is

$$p = x\cos\theta + y\sin\theta \qquad (3\text{-}1)$$

Restricting θ to the interval $[0, \pi]$ leads to unique normal parameters $[p, \theta]$, i.e. every line in the [x,y]

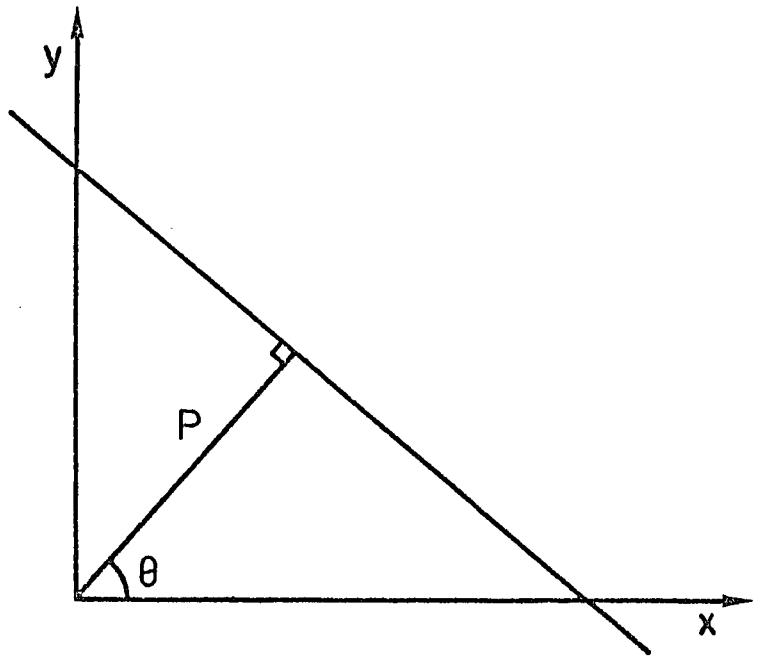

Fig. 1 The normal parameters of a straight line.

plane corresponds to a unique point in the [p, θ] plane. From Eq. (3-1), we note that a point in the [x,y] plane corresponds to a sinusoidal curve in the [p, θ] plane. Suppose now that we have a set of points $[x_i,y_i]$ lying on a straight line. For every one of these points there corresponds a sinusoidal curve in the [p, θ] plane, specified by

$$p = x_i\cos\theta + y_i\sin\theta$$

We can show that all these curves corresponding to points of a straight line have a common point of intersection, say $[p_0, \theta_0]$ in the transform plane. This point defines the line that passes through all these points. In summary, we have the following HT properties:

Property 1. A point in the image plane corresponds to a sinusoidal curve in the $[p, \theta]$ transform plane.

Property 2: A point in the $[p, \theta]$ transform plane corresponds to a straight line in the image plane.

Property 3: Points on the same curve in the $[p, \theta]$ transform plane correspond to lines through the same point in the image plane.

Property 4: Points on the same line in the image plane correspond to curves through the same point in the $[p, \theta]$ transform plane.

We next apply these results to line detection. First, both p and θ are quantized to N_p and N_θ levels, respectively, with levels that depend on the desired

resolution. The required computation increases linearly with N_θ or with an increasing number of image points, but it does not depend on N_p. A larger N_p increases the storage requirements, but not the amount of computation. For N_θ levels in θ and N_p levels in p, there will be N_θ x N_p quadruled grid points in the [p, θ] plane. In the case of binary images, we assume that a zero corresponds to dark background while unity corresponds to bright points in the image. The quantized [p, θ] region is treated as a two-dimensional (2D) array of accumulators, initially set to zero. The transform procedure is to scan all non-zero pixels in the image. At every non-zero pixel, at position (x_i, y_i), for every quantization value of θ, we compute p and for every pair [p, θ] we increment the corresponding accumulator. Thus, a given cell in the 2D accumulator array eventually records the total number of curves passing through it. When all image pixels are scanned, the value in each $[p_i, \theta_i]$ cell yields the number of pixels (within quantization error) which lie on the line $[p_i, \theta_i]$. Therefore, large peaks in the accumulator correspond to long lines in the image. If there are n non-zero pixels in the image, then for each pixel N_θ calculations are required and overall nxN_θ calculations are needed to complete the transformation. Clearly when n is large, compared to an exhaustive search that requires considering the lines

between all $\frac{1}{2}(n(n-1))$ pairs of image pixels, this procedure is more efficient.

There are limitations to this approach. First, the final results are sensitive to both N_θ and N_p. While a finer quantization increases resolution, it also exposes the problem of clustering entries corresponding to nearly colinear points. Furthermore, as was mentioned earlier, a finer quantization in N_θ will also increase the computation time. Secondly, this technique finds colinear points without regard to their contiguity. Thus, a peak value in the HT domain may represent either a continuous or a number of smaller discontinuous segments on the same line. Furthermore, an image line segment can be distorted by unrelated image pixels. Finally, in an image containing many lines of different orientations, there will be "crosstalk" among these lines in the HT plane. Consider, for example, the case of an image where there are twenty lines parallel to the y axis. Then, depending on the line to line separation in the [p, θ] domain, there will be twenty peak values in the $\theta = 0°$ axis for different p values. However, in searching for lines parallel to the x axis, we will always find twenty colinear points. Thus, in the [p, θ] domain, we will find smaller peaks in the direction of $\theta = 90°$, and for all values of p. This problem is even more exaggerated in

the case of a natural scene where the lines are not just one but many pixels wide.

In the previous discussion, we have considered binary, i.e. only two, gray level images. A natural scene is usually digitized to more than two gray levels and modification is necessary in order to adopt the Hough technique. There are two possible ways to accomplish it. The first method is to threshold the image so that only two gray levels will exist. However, information is lost during thresholding. In the case of a regular texture, after thresholding, while some detail is lost, the basic texture structure is still present. The second method is to relate the HT to the Radon transform (RT). [53,54] The RT is a well-known integral transform from the theory of computed tomography. The forward RT is defined as

$$F(p, \theta) = \int_{-\infty}^{\infty}\int f(x,y)\, \delta (p - x\cos\theta - y\sin\theta)\, dy dx \qquad (2)$$

where $f(x,y)$ is the input, $F(p,\theta)$ is the transform and $\delta(.)$ is the unit Dirac impulse function. $F(p,\theta)$ is equivalent to the integration of $f(x,y)$ along a straight line whose normal parameters are $[p,\theta]$. Recently, it has been pointed out

that, for binary images, the forward RT is equivalent to the HT. [55] We extend this concept by applying the RT to the non-binary image and by approximating the integral of the image along a line. The simplest approximation is to add along a given line all the pixel intensities. Thus, when we calculate the transform by updating the 2D register, when we get to the point to update a cell, we do not just increment by one, but we add to it the intensity of the pixel under consideration. In this sense, brighter lines correspond to brighter peaks in the RT domain.

More advanced HT implementations are found in [56]. Optical implementation of the HT, both coherent and incoherent versions, is also possible. [57,58] Recently, an optical HT that works at real-time video rates has been reported. [59]

4. VECTOR MEDIAN FILTERS

An MF uses an odd window size and thus a center element can always be defined. For an even window size MF, however, not one but two center elements can co-exist. For an even number of window elements, then, there is a problem in defining the output element. It has been suggested [2] to use as the MF output the average of the two center elements. An alternative is to define a vector rank filter (VRF) as the filter that simultaneously outputs a number of elements, i.e. an output data vector. As in the case of a scalar rank filter (with single element output data), we can define the MIN and the MAX vector rank filters [1] as those filters where the output vector is at either end of the sorted data window. For the vector median filter (VMF), the output values are at the center of the sorted data window.

Formally, the VMF_{NxM} is defined as follows: let a data window of size N slide along a data array. At any position, the N window elements are sorted according to their numerical value. At this position, the VMF_{NxM} output is a set of M elements, where M is less than N, situated at the center of the window, i.e. an equal number of sorted elements exist on either side of the output window between the output window ends and the data window ends. For such

an output to exist, both N and M must be restricted to be either both odd or even numbers of elements. The data window then moves M units over and the procedure is repeated.

We note that for an MF the median in the data window does not depend on the sort order, i.e. the same value is obtained if we sort in either an increasing or a decreasing order. However, this is not true for the VMF. It is easy to see that if we always sort, say, in an increasing order, a monotonically increasing signal is not affected by the filter, but a decreasing signal is distorted. Therefore, the sorting direction cannot be left arbitrary. To preserve both the increasing and the decreasing signals, the sort direction is determined by examining the signal trend in the data window. The rule we adopt is as follows: for each data element we assign a tag +1, -1, or 0, depending on whether the difference between neighboring sample amplitudes is positive, negative, or zero, respectively. Now, for the elements within the window, we compute the majority of the assigned tags. This can be accomplished by simply adding the element tags. If this result is positive (for a mostly increasing signal), then we sort the elements in an increasing order. Conversely, if the result is negative (for a mostly decreasing signal), then we sort the elements in a

decreasing order. Finally, if the sum is zero (for either a constant or an oscillatory signal), the sort direction can be left arbitrary.

Since the VMF has two design parameters N and M, as compared to the MF which has only one design parameter N, as well as a trend test, the VMF has a wider choice of filtering possibilities. For example, we could adaptively adjust M based on the results of the trend test. For a slowly varying signal, we can increase M to improve speed. For a constant signal for optimum speed, we can let N equal M and skip the sort. Also, fast MF sorting algorithms could be adopted for the VMF as well.

Based on its definition, the following deterministic VMF properties can be derived:

Property 1: Any monotonically increasing or decreasing signal will be preserved.

This can be seen from the fact that the elements falling within the data window are already sorted in the correct order and thus will not be disturbed. Since a step (or edge) signal is monotonically increasing or decreasing in a region around the edge, it will be preserved by a VMF.

Property 2: Any impulse narrower than ½(N-M) samples will be eliminated.

This is so because if an impulse falls in the window, after the sort it will be "pushed" to one end of that window. If the impulse is narrower than ½(N-M) samples, then it will fill up the window at a point where none of its elements will fall in the output vector. Since none of the impulse elements falls in the output vector, the impulse will be eliminated. By the same reasoning, any pulse wider than ½(N+M) will pass unaffected. Therefore, the VMF has the same two fundamental properties of the MF, namely, it preserves edges while it filters out sufficiently narrow spikes. As was mentioned earlier the MF will not preserve "roof type" edges such as peaks of triangulars. This is also true for the VMF. Additionally, depending on the starting point, the distortion introduced by a VMF could be asymmetrical even for a symmetrical "roof type" edge. However, for many classes of signals this could be a tolerable distortion.

It is known that if a signal is repeatedly MF-ed it will eventually convert the signal into a root signal [17]. An upper bound for the number of filter passes required to reach a root signal is given as ½[L-2], where L is the

length of the signal. However, this is not a very tight bound. It has been observed that for the same signal, the minimum number of passes required to reach a root signal decreases with increasing filter window size. Even though it has not yet been proved, it has been experimentally verified that the VMF has the same property, namely if a signal is repeatedly VMF-ed, it will eventually convert into a VMF root signal.

We also note that the VMF moves M-times faster along the data array than its corresponding MF counterpart. However, the VMF is slightly more complex to implement. Despite this additional computational complexity, the VMF results, especially for long data arrays, in a significantly shorter processing time. As with the MF, to account for end effects, $\frac{1}{2}(N-M)$ elements are appended to the beginning of the data array, each with a value equal to that of the first element. The number of elements appended at the end of the signal depends on L, where L is the length of the data array. However, in all cases, appending $\frac{1}{2}(N+M-2)$ elements at the end will always be sufficient to generate an output array. Also we note that if L is not a multiple of M, the output array will be longer than L. In this case, we simply keep the first L elements of the output.

Figures 2 and 3 illustrate the deterministic properties of the VMF. Fig. 2a shows the result of filtering a sinusoid of frequency of 6Hz with a VMF_{4x2} without the trend test and with increasingly ordered elements. We note that the increasing regions of the signal remain unaffected, but all the decreasing regions are distorted by the filter. Fig. 2b shows the same VMF, but with the trend test inserted. We can observe that most of the distortion is corrected. A small amount of distortion is still present at the signal peaks, and it could be due to "roof type" edge response using an unfavorable starting position.

Fig. 3a shows a square waveform where spikes of duration 2, 3 and 6 sampling points were added. Fig. 3b shows the result of a VMF_{6x2}. Since N=6 and M=2 then $\frac{1}{2}(N-M)=2$, and therefore all two point duration spikes were removed. Fig. 3c shows a similar result with a VMF_{10x2}. Now all four point duration spikes were removed. Finally, Fig. 3d displays the result of filtering by a VMF_{14x2}. Here, all of the spikes were removed. The same result is obtained when a VMF_{15x3} is used. It is worth noting that in all cases the signal edges were preserved.

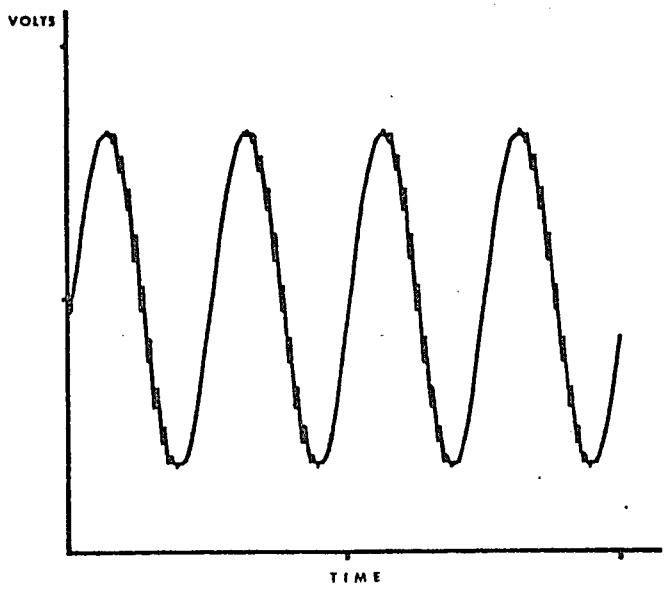

Fig. 2a. Output of a VMF_{4x2} without a trend test.

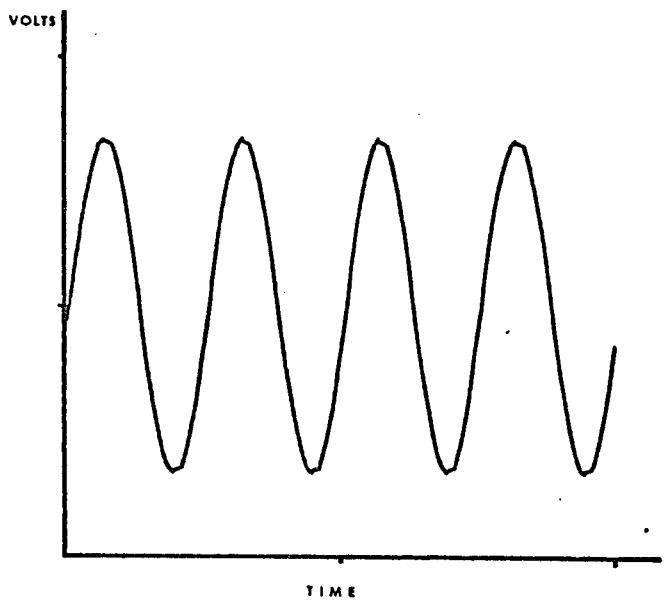

Fig. 2b. Output of a VMF_{4x2} with a trend test.

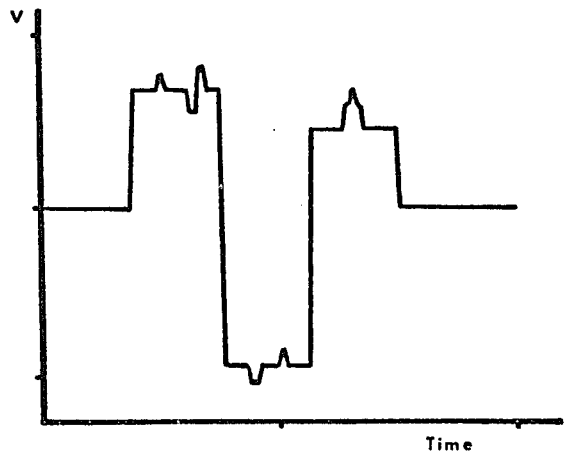

Fig. 3a. Signal with added impulsive noise.

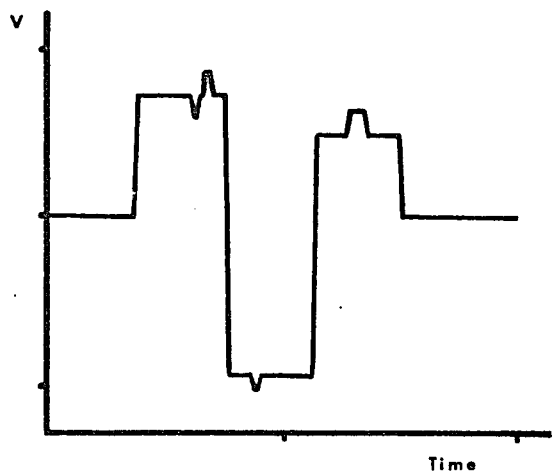

Fig. 3b. Filtering Fig. 2a with a VMF_{6x2}.

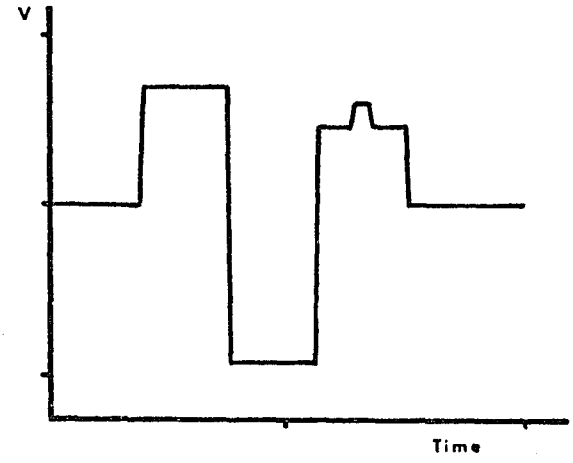

Fig. 3c. Filtering Fig. 2a with a VMF_{10x2}.

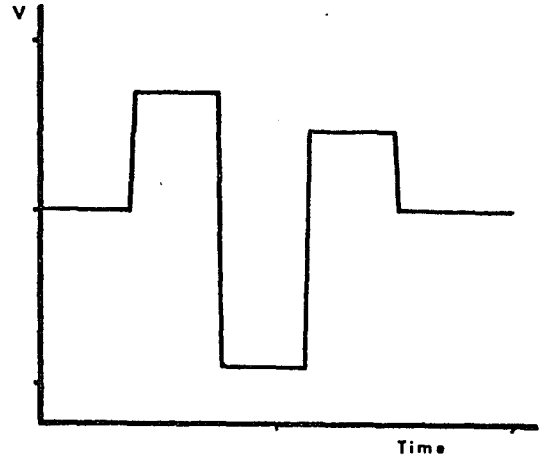

Fig. 3d. Filtering Fig. 2a with a VMF_{14x2}.

Next, the sinusoidal response of a VMF is investigated. The sinusoidal response of the MF has been investigated by Velleman. [4] Here, the parameter of interest is the gain, which is the ratio of the fundamental harmonic power output to the fundamental harmonic power input. Fig. 4a shows the gain, on a logarithmic scale, of an MF_3. The sinusoidal input is sampled at a rate of 128 samples per second. Fig. 4a shows that the MF_3 has a null of about -40 db at a frequency of about 43 Hz. Fig. 4b shows the gain of an MF_5. Here we note that there are three nulls. As a comparison, Figs. 4c and 4d show the sinusoidal response of a VMF_{4x2} and a VMF_{5x3}. Comparing Fig. 4a with Fig. 4d, we note that, as the window size increases, the VMF has a smoother response, concluding that for larger windows the VMF has, in general, a smoother sinusoidal response than the corresponding MF.

4.a VECTOR MEDIAN FILTERS - STATISTICAL PROPERTIES

Most of the work done on the statistical properties of the MF involves independent-identically distributed (i.i.d.) input data. Some work was also done for non-i.i.d data. [18] Here, we will assume that the input data is i.i.d. with probability distribution and density functions $F_x(x)$ and $f_x(x)$, respectively. Let y_i be the ith element of the VMF output at some position, where $1 \leq i \leq M$. Using a

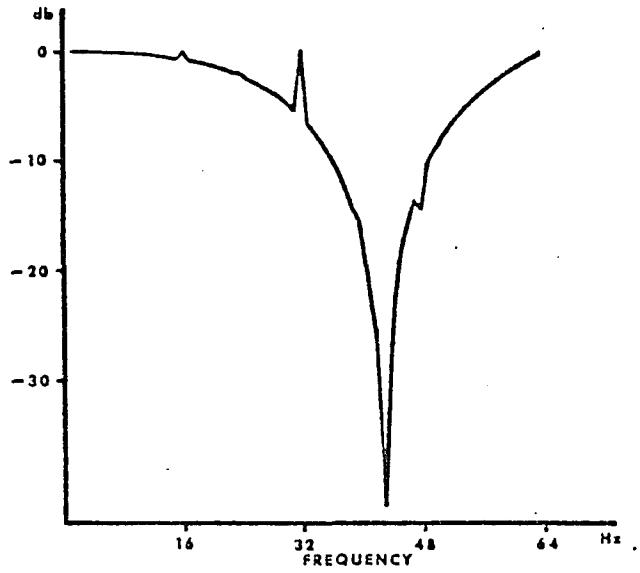

Fig. 4a. Gain with a MF_3.

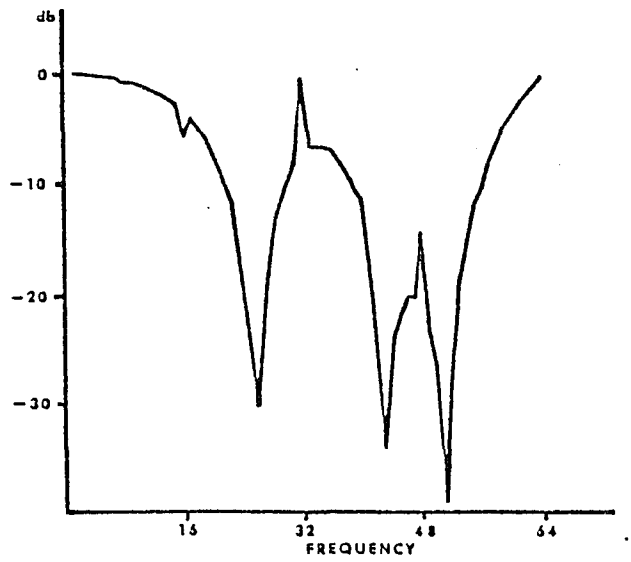

Fig. 4b. Gain of a MF_5.

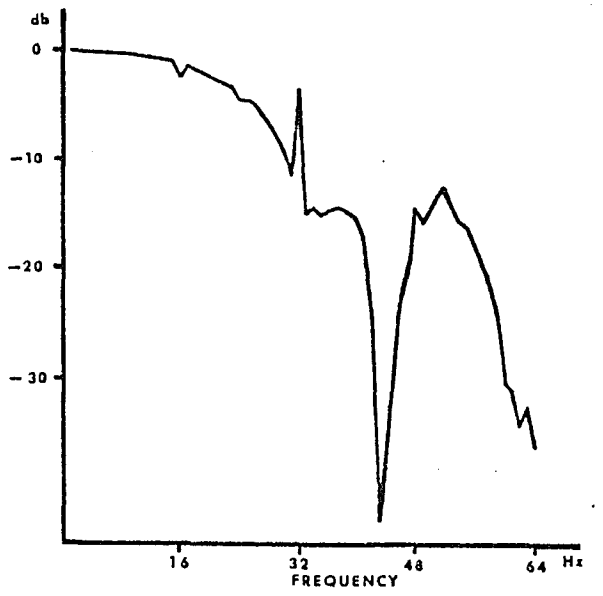

Fig. 4c. Gain of a VMF_{4x2}.

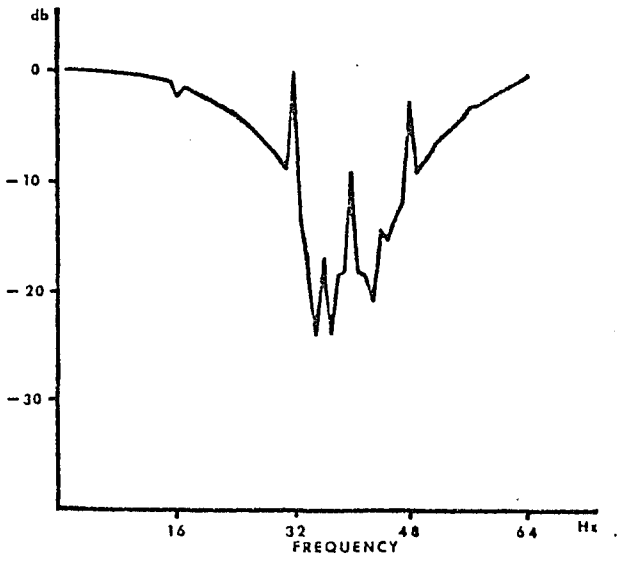

Fig. 4d. Gain of a VMF_{5x3}.

well-known order statistics [19] result, the probability density of y_i will be given by:

$$f_y(y_i) = \frac{N!}{P!\ Q!} [F_x(y_i)]^P \cdot [1 - F_x(y_i)]^Q \cdot f_x(y_i) \qquad (4\text{-}1)$$

where $P = \frac{1}{2}(N-M)+i-1$ and $Q = \frac{1}{2}(N-M)+M-i$

In the special case where M=i=1, Eq.(4-1) reduces to the output probability distribution of an MF with an output that is also i.i.d. However, for the VMF, even though the output elements are independent, they are not identically distributed. Furthermore, the density function of each output element is one of the M functions that appears periodically with period M in the output array. Also, the mean and the variance of each output element are not the same, and thus we must define the M-dimensional mean and variance vectors of means and variances, respectively.

As an example of such calculation, it is assumed that the input has an exponential or Laplacian probability density function (EPD). The EPD represents the noise due to laser intensity speckle. The performance of an MF on a laser speckle was examined by Frieden [20]. We will calculate the output statistics of a VMF due to EPD inputs and compare these results with those obtained with the MF. The

details can be found in Appendix A. As is shown there, the performance of the VMF on the laser speckle is slightly superior of that of the MF.

4.b TWO-DIMENSIONAL VECTOR MEDIAN FILTER

Median filtering can be extended to two dimensional signals, like images. The 2-D window has both a size and a shape. The 2-D MF operates as follows: with the window centered at a particular pixel, all the elements that fall within the 2-D window are transferred into a 1-D array, and sorted according to their numerical value. Then, the element that falls at the center of the 1-D array replaces the pixel at the center of the 2-D window. The window then moves to another pixel and the procedure is repeated. For a median value to exist, the total number of pixels in the 2-D window has to be odd. Furthermore, it has been proved that if the 2-D window is symmetric around the origin and also includes the origin, then the filter will preserve edges. Various forms of windows can be used, e.g. line segments, squares, circles, crosses, square frames, circle rings etc. All these windows fulfill the above edge preserving property, except for square frames and circle rings that do not include the origin. However, it has been observed that square frames and circle rings will change the edges

slightly. There is no theory to predict which window shape will perform better on different types of signals, and the best choice is an ad hoc process.

One disadvantage of 2-D MF's is that they are computationally slower than 1-D MF's. Several fast algorithms were developed to minimize this problem (see Sec. 4.c). Another approach is to use successive 1-D MF's, filtering first the rows of a 2-D signal and then filtering the columns of the result [6,7]. Such process was named a separable filter. Experiments showed that a separable MF behaves well enough so it can be used in place of a 2-D MF [7], the advantage being that its speed and implementation, both in hardware and software, make it more efficient. Nevertheless, the 2-D MF is in general considered superior to the separable filter.

The 1-D VMF can be used for 2-D filtering in the same way as an 1-D MF can be used for 2-D filtering, i.e. the separable VMF is a straight-forward extension of the separable MF. Additionally, the separable VMF enjoys the benefits of the VMF, i.e. multiparameter, superior speed, etc. Furthermore, since two filter passes are required (one for rows and one for columns), significant saving in processing time is possible.

However, even though the 2-D MF is a direct extension of the 1-D MF, this is not true for the 2-D VMF. The difficulty here is two-fold. One, how can a 2-D median vector can be defined, and two, how will this median vector be filled from 1-D ranking. The median vector could be defined to be a 2-D sub-window at the center of the filter window. For example, for a 5x5 square filter window, the median vector could be a 3x3 square sub-window centered within the 5x5 window. However, when the 25 elements in the window are sorted and the 9 center elements are taken out, still the problem of how these 9 elements will be arranged in the 3x3 median vector is not resolved. It is evident that an incorrect placement rule will result in pixel dispositioning within the image, and unacceptable distortion. Additionally, the trend test must be very carefully selected.

In a first attempt to circumvent the problem, a quasi-2D VMF is defined to be that filter for which the filter window is 2-D and the median vector is 1-D, i.e. one stripe centered within the filter window. The orientation of the stripe can be selected depending on the window shape. For example, for a cross window the stripe could be horizontal or vertical, where for a square window it could have any orientation. In general, there is no way to predict which

orientation will perform better, in the same way we cannot predict which window shape performs better. Different types of derivatives can be used as trend test. Directional derivatives in the same direction or perpendicular to the median vector, 2-D derivatives and gradients are possible choices. Again, the best choice is an ad hoc process.

In computer experiments the quasi-2D VMF performed very satisfactorily for the smaller median vector dimensions (up to four). Details will be presented in the experimental results section. Unfortunately, the performance of the 2-D VMF was found to be unacceptable in noise removing applications. For various placement rules and derivative tests, the performance was not satisfactory. However, even though the 2-D VMF has poor performance on noise removing applications, through investigation it might be found useful in other types of signal processing.

4.c FAST TWO-DIMENSIONAL VMF ALGORITHM

One major drawback of MF's is that computationally they are very time consuming compared with linear filters, since at every filter position, the elements within the window have to be sorted. This problem is even more pronounced in 2-D filtering because more elements are present in the

window. For example, even for the smallest 3x3 square window, nine elements have to be sorted at every position. For this reason, 2-D MF's tend to be very slow.

To minimize this problem, a number of fast MF algorithms have been devised, both for off-line and on-line computation [5, 60-67]. Some of the real time filtering algorithms are based on the binary representation of the samples and the median value at each position is calculated bit-by-bit [60, 61]. Other on-line algorithms are based on the availability of special dedicated hardware, [62, 63] where others are based on analog selection networks [64, 65]. As was mentioned earlier, single chip MF's were able to be fabricated using VLSI technology [14-16]. These chips utilize digital comparators in systolic algorithms and structures [14] where others are based on fast algorithms [15, 16]. Very recently, manufacturers announced a team of image processing chips, where one of them is dedicated for median filtering [68].

Among the off-line fast algorithms, the most important is based on histogram calculation. This algorithm utilizes the fact that moving from one position to the next, some elements leave and some new ones enter the window, where many others remain the same. Thus, a histogram of all the

gray-levels in the window is formed, and updated at every new position. The median value is found by searching the histogram. One fact that should be realized at this point is that the only difference between the 1-D and 2-D MF is in the way the elements to be sorted are collected, i.e. the window shape. The 1-D MF can be considered as a special case of the 2-D MF. The sorting process is identical for both filters. Therefore, the histogram based algorithm can be used for both 1-D and 2-D filtering.

A histogram based fast 2-D VMF algorithm is now presented. Since the VMF is basically an off-line filter, the histogram method was chosen to be the most appropriate. The algorithm is a modification of the fast MF algorithm adopted for VMF filtering.

Assume that N elements exist in the 1-D or 2-D window, and that the median vector V(k) is of dimension M. We note that if M=1 (MF), then the median value is of rank $R=\frac{1}{2}(N+1)$ in the window. If now $M\neq1$ then the rank of the first element in the median vector, i.e. V(1) will be of $R=\frac{1}{2}(N-M+2)$. Subsequently, at every position we search the histogram to find V(1). Once this element is found, beginning from V(1) we take the next M non-zero positions of the histogram to be the elements of V. Moving to the next

position, the histogram is updated and the procedure is repeated. The algorithm can be far more efficient if moving from one position to the next we save information about the position of the previous V(1) in the histogram. This information can be updated when updating the histogram. Therefore, at a new position instead of starting from the beginning of the histogram, we start from the position of the previous V(1) and we move up or down the histogram until we find the new V(1). Since in natural scenes the pixels do not change too much from position to position the new element V(1) will most likely be close to the position of the previous V(1).

The algorithm is summarized in the following steps:

Let: $h(i)$ = Gray-level histogram of elements in window.

$P=V(1)$ = Histogram position of $V(1)$

$R=\frac{1}{2}(N-M+2)$ = Rank of $V(1)$

$S_p = \sum_{i=0}^{P} h(i)$ Histogram sum up to position P

Step 1 : Set $P=0$, $S_p=0$. Form the first histogram $h(i)$.
Form the first trend test.

Step 2 : Start from histogram position P with S_p.
If $S_p > R$ ($S_p < R$), move down (up) the histogram so that $S_{p-1} < R \leq S_p$.

Set V(1)=P, L=1

If S_p=R, then go to step 4.

Step 3 : Let L = S_p - R + 1

If 1 < L < M, then V(1) = V(2) = ...V(L)=P. Go to step 4.

If L $\geq$ M, then V(1) = V(2) = ...V(M) = P. Go to step 5.

Step 4 : Move up in the histogram and take the next (M-L) non-zero positions of h(i) to be subsequent elements of V. If at any position h(k)=B, then count position k, B times.

Step 5 : Check trend test and load V in output array.

Step 6 : Update histogram for incoming and leaving elements. If any incoming (leaving) element x_j is $x_j \leq P$, then increment (decrement) S_p. Update trend test.

Step 7 : Stop if the end of the line is reached. Otherwise go to step 2.

Note that in steps 2 and 3 provision has been taken for multiple values in the window. For example if all the

elements in the window have the same value (constant region), then the histogram will be zero everywhere except at the position of that gray level. In this case S_p=N and L > M (see step 3). In this case all M elements of V will have the same value P. We also notice that the elements in the median vector are sorted in an increasing order. However, depending on the trend test the median vector could be loaded in the output array in reverse order. The trend test which is the sum of the sign tags of all the elements in the window is updated at the same time as the histogram.

In the more general case where rank filtering is desirable, the algorithm can be used by simply changing the value of R. Thus, this algorithm can more generally be used as a fast rank filtering algorithm.

The performance of the algorithm was tested on the same images used for VMF performance evaluation. The figure of merit was the computer run time of two identical filters, one implemented with the standard quicksort method, and the other using the fast algorithm. The performance will be presented in the experimental results section.

4.d SIGNAL RESTORATION USING TRANSFORM AND MEDIAN FILTER

One interesting application of MF's is in signal restoration. In this new filtering technique, advantage is taken of the two fundamental properties of the MF, i.e. to preserve monotonic signals while eliminating narrow spikes. In this type of problem, a useful signal is corrupted by an unwanted signal. The interference could be unintentional, or it could be intentional jamming. To restore the desired signal, a suitable linear, reversible transform converts the wanted signal into a smooth transform, where the undesired interference has the form of spikes superimposed on the smooth transform of the useful signal. To recover the signal, a MF is applied in the transform domain. The filter window size is selected so that the spikes are removed, and the remaining transform is not too much degraded. The inverse transform will reveal a close approximation of the desired signal.

Even though the proposed solution to this problem may sound too fictitious because of the existence of such a transform, a very practical application is in removing unwanted sinusoids from signals. It is very well known that the Fourier transform of a sinusoid is a spike. Furthermore, the Fourier transform is linear and reversible

and can be very efficiently computed via the FFT algorithm. Additionally, a large class of one-dimensional signals, and especially the band-limited signals, have a sufficiently smooth spectrum. This suggests that the previously proposed filtering scheme can be applied in these problems. The problem is now formulated as follows: A signal is corrupted by sinusoidal interference of unknown frequency (jamming). Furthermore, the jamming frequency could change (frequency hopping). It is assumed that the useful signal has a sufficiently smooth spectrum. To remove the jamming signal, the Fourier transform of the noisy signal is median filtered. Since the noisy transform has the appearance of a smooth signal with superimposed spikes, the MF will remove the spikes without degrading too much the remaining transform. It is worth noting that the location of the spike (jamming frequency) is irrelevant. The inverse Fourier transform of the MF'ed spectrum will recover the desired signal. A block diagram of such a filter is shown in Fig. 5.

Experiments showed excellent performance on many commonly used signals, like squarewaves (digital signals). Details will be presented in the experimental results section. This filtering scheme has many practical applications, especially in communications and signal

transmission. Furthermore, hardware implementation is possible since both FFT [69, 70] and MF [14-16, 68] chips exist, allowing real-time processing. Additionally, this filtering scheme can be extended to other types of signals as well, by finding a suitable transform. For example, it is known that the Hough transform converts straight lines into spikes in the transform domain. Therefore, the HT along with the MF can be used to remove lines from images. However, the inverse HT is not easily feasible.

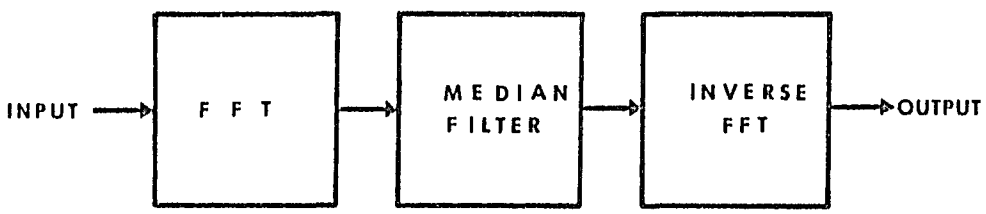

Fig. 5 Block diagram of a sinusoid removing filter.

5. TEXTURE CLASSIFICATION USING THE HT

Structural approaches to texture classification are based on regularity. A resolution cell is periodically repeated within the texture in accordance with some placement rules. This approach is very well suited for regular, periodic structures. A different point of view of periodic structures is to consider them as a regular arrangement of lines, or line segments, of different orientations. Consider, for example, an image of a brick wall. Such a structure can be viewed as a repetition of horizontal lines, with line segments regularly arranged in the vertical direction. However, if a rotated view of the same image is considered again, then horizontal and vertical lines might no longer exist. In this case, we can still view this structure as a regular repetition of solid lines of some orientation, with line segments perpendicular to the solid lines. This last view is independent of rotation, a very desirable property of a classifier.

In some structural approaches, the primitives used for classification are the average separation of edges at some certain orientations and the arrangement of edges, again in certain orientations. [38,39] However, the number of orientations to be considered is limited by the required

computation, and the classification is rotation sensitive. A model based, rotation invariant texture classification method is described in [71].

The classification method proposed in this paper does not consider line (or edge) orientation, but instead the angle at which they intersect. This angular feature does not depend on image rotation. A second classification primitive is the normalized separation of lines for the same orientation. This feature is independent of scale, since the normalized separation does not change with scale. Furthermore, this feature is also independent of linear image translation. A third classification primitive is the number of principal line orientations within a texture. In a regular texture, the lines or edges will not appear along random directions, but will be arranged regularly at some orientations. For example, in a brick well image, the lines appear along two principal directions spaced 90° apart.

These primitives are especially suited to describe regular textures that consist mainly of straight line elements. Lines, or more generally curves, are an important characteristic of textures. The HT is a very efficient technique to find desirable texture primitives, since all the information can be extracted from the $[p, \theta]$ domain.

Since the classification algorithm needs to be invariant under geometrical transformation, such as rotation, translation or scaling, therefore how the HT of an image is affected by these transformations needs to be examined. In this regard the following HT properties are of interest.

Assuming that an infinite periodic line structure exists, then the following HT properties hold:

Property 1: If two lines intersect at angle ϕ, where $0° < \phi \leq 90°$, then the corresponding points in the $[p,\theta]$ domain will be also located ϕ degrees apart.

Property 2: Linear translation of the image has the effect of shifting the HT in the p direction. The shift of the transform is not uniform, i.e. some peaks at some angles may shift less, or even not shift at all. This is evident from the fact that the normal parameters of lines parallel to the direction of the translation will not change, whereas the parameters of the lines perpendicular to the direction of the translation will be most affected.

Property 3: Linear translation does not affect the spacing between peaks at a given angle. This is so because, after the translation, while the location of lines with

respect to the origin may change, the relative line-to-line spacing does not change.

Property 4: Image rotation, by some angle ϕ, is equivalent to a circular shift of the HT by the same angle ϕ. This property is due to the fact that lines that are located at angle θ_o, after rotation will be located at angle $\theta_o + \phi$. The shift is circular because all points crossing the 180° axis will again appear at the 0° axis, i.e. the HT is "wrapped" around a cylinder, so that the θ = 0° axis is identical to θ = 180° axis.

Property 5: Scaling the texture in all directions (zooming) is equivalent to scaling the HT in the p-direction. This is so because, after the scaling, for all lines the p-parameters will scale, but not the θ -parameters.

These HT properties show the effects of geometrical transformations on the HT and also indicate that with proper normalization it is possible to compensate for all geometrical image transformations.

Zucker [33] suggested that natural textures can be considered as distortion of ideal structures. What would appear to be ideal are structures formed of thin straight

lines, periodically arranged in different directions. It was pointed out earlier that there is some "crosstalk" between lines at different directions. This is more pronounced in textures where there are wide lines present. In this case, the HT peaks corresponding to these lines will not be single points but rather regional peaks. Also, all these curves will have the effect of raising the values of other accumulators in the 2D register, and therefore the HT will have the appearance of wide peaks in a noisy background. However, in the ideal case, where the image lines are one pixel wide, then the curves corresponding to line pixels will pass through one point in the HT domain. In this case, the HT will have the appearance of large sharp peaks in some low-level noisy background. It is apparent that line detection now is far more accurate. From this discussion, it follows that classification accuracy can be increased by preprocessing the natural texture so that it will be as close as possible to an ideal one, or in other words, by extracting the "skeleton" of the texture. This can be accomplished by thinning the lines or by first applying an edge detector and then applying a gap filling operator to form solid lines. Different techniques are available for this purpose. [41,72]

In the following discussion, it will be assumed that the texture is a regular, periodic structure formed by thin straight lines. This image can be the result of preprocessing, or simply it is an ideal test pattern. The next classification step is to calculate the HT to a desired resolution. Texture HT primitives will then be extracted. The HT texture primitives are: a) the number of line orientations, b) the relative orientation angles, and c) the line spacing for each orientation. Several normalization procedures will be employed so that: a) classification will not depend on geometrical transformations and b) the dimensionality of the feature space is reduced. Furthermore, to reduce the space to 3-D, some statistical parameters will also be used.

The classification algorithm is composed of the following steps: First calculate the texture HT with the desired resolution. As mentioned earlier, finer quantization of the p-axis does not increase the computation, but only the storage requirements. Therefore the important decision is to select the quantization levels of the θ-axis, i.e. the directions at which we will search for lines. Next, locate and isolate peak values in the HT. Different approaches can be used at this step. One approach is to simply threshold the HT with a proper threshold parameter so that only the

peak values remain. However, simple thresholding exposes the problem that peaks below the threshold will not be "seen", whereas in some regions of the transform the background noise may be larger than the peaks in a different region. Therefore the proper threshold value is very critical, and must be very carefully selected. A better approach is to use a Median Filter (MF). The MF will eliminate local peaks in the HT. Hence, subtracting the output of an MF from the original transform will reveal the local peaks. The window of the MF should be kept sufficiently small so that the peaks could be eliminated without much affecting the background. Some post thresholding will eliminate any residual noise. Since the HT peaks are usually regional, it is very possible that neighboring angles will also have peak value. For example, if a texture has lines oriented at $\theta = 30°$ it is possible that peak values will be also found at, say, $\theta = 28°$ and $\theta = 32°$. Since these angles actually correspond to the same lines, these peak values should be merged with the central peaks. Then, for every value of θ where peaks are present, record that value of θ and the distance between consecutive peaks. The result of this step is a table listing those angles where peaks were found, and the distance from peak to peak. This is the information about line orientation and separation of lines at each direction. However, this

information depends on geometrical transformations and it must be normalized. Therefore, normalize all the recorded distances in the table to the maximum or the minimum one. Usually the maximum is the better choice, since it is less sensitive to distance errors. This normalization will take into account texture scaling. Next, calculate at every angle the average of the normalized distances. The average normalized separation at every orientation was found to be a good measure of the arrangement of the lines at that orientation. After this step all the information is in a table of two rows and as many columns as there are orientations in the texture. The first row is the value of angles where peaks were found, and the second row is the average normalized separation at those angles. Now, circularly shift the columns of this table so that the maximum value of the second row occurs at the first column. This way we select the orientation with the largest average separation as the reference orientation. Then, subtract the first entry of the first row from all the entries at the first row. Subtraction is modulo-180°. This will set the reference orientation at $\theta = 0°$ and take rotation into account. Next, divide all entries of the first row by 180°. This will normalize the angles to 180°. Also, divide all entries in the second row by the first entry of the same row. This will set the maximum average separation to unity.

After the previous steps, all texture information is condensed into a normalized table of two rows and as many columns as there are texture orientations. Note that the first column of the normalized table contains no information since it has the values of zero and one. We can consider the two rows of the table as two feature vectors, one revealing information about line directionality and one about line separation. For multi-directional textures, the dimension of these feature vectors will be large. We can reduce the dimension of the feature space by replacing the two vectors by their corresponding variance. This step reduces the accuracy of classification. Hence, calculate the variance of each row of the table at the previous step and form a feature vector $\underline{x}=[x_1,x_2,x_3]$ where

x_1 = number of orientations in the texture.

x_2 = variance of the normalized separations.

x_3 = variance of the normalized orientations.

This last feature vector will be used for classification.

The classification process consists of finding the feature vector of the texture to be classified using the previous algorithm. Then a similarity measure, or any clustering algorithm, can be used to classify the unknown

vector to one of the known classes established during the learning process. Note the importance of the x_1 element of the feature vector. Textures with different numbers of orientations cannot be classified incorrectly in the same class.

6. ASSOCIATIVE MEMORY ENCODING

As was mentioned above, after the texture features have been extracted by applying the proposed algorithm, and a feature vector is formed, a similarity measure or a clustering algorithm can be applied to classify the unknown vector into one of the known reference classes established during training. There are a large number of such algorithms in the open literature with different levels of complexity, and the choice of the one with the best performance is an ad hoc process. One new alternative is to use linear associative memory mapping (LAMM). This can be noticed from the fact that classifying an unknown vector into a known class is the equivalent of mapping that vector into a point (or small region) in space. Additionally, error tolerance properties of LAMM make this mapping least square sense optimum and it can correct some errors introduced by the feature extraction algorithm. The result is a very elegant and compact decision maker which can be incorporated as part of the feature extraction process. Furthermore, this method can be applied to classification of patterns other than regular textures.

Associative recall may in general be defined as a mapping in which a finite number of input vectors are

transformed into a given set of output vectors. In the case of incomplete or erroneous input vectors, it has been shown [73] that this mapping is least square sense optimal. It is this error tolerance that suggests its applicability to pattern (vector) restoration and classification. There are two kinds of associative recall: the autoassociative and the heteroassociative. In the former an incomplete vector is restored into a complete version of itself, while in the latter, an output vector is produced in response to an input vector. The mapping between the input-output pair of vectors is arbitrary and depends on the application requirements. Associative recall suggests the working mechanism of an error-correcting content-addressable memory. This means that, for all similar vectors, in the sense of some appropriate measure, the recall will be similar to the corresponding output vector.

The mapping process is described by the following transfer relation:

$$y_k = Mx_k \qquad k=1,...P \qquad (6\text{-}1)$$

where x_k's and y_k's are the input and output column vectors and M is the unknown associative memory matrix. LAMM is a problem of memorizing a set of responses to a set of input

signals. It can be formulated as a problem of finding the optimal solution, in the sense of least squares, for the matrix M. Given M, recognition is achieved by linearly transforming an unknown input according to Eq.(6-1) and therefore belongs to the scalar transform category. The matrix M can be determined through an iterative procedure. The mathematical details can be found in Appendix B.

The classification task with an associative memory is a two stage process. In the first, a learning (or training) phase, patterns representative of given classes are used to induce the proper output responses. According to the algorithm presented in Appendix B, first the matrices X and Y need to be generated. This is done by concatenating the feature vectors in a column-wise fashion, with the columns of X representing the set of training features, and the column of Y the desired response. In the second or recognition state, an unknown or degraded feature vector is applied to the input of an associative memory filter, and the produced output is similar, in the least square sense, to one of the trained responses.

For the purpose of texture classification, according to the proposed HT based algorithm, the input vectors could be the reference feature vectors of all the textures in a given

class. One realistic approach to generate the reference features for each class is to run the algorithm on several views of the same texture, where each new view is the result of geometrical transformations. Since the algorithm is invariant to such transformations, all the generated features should be very similar to each other, with some discrepancies to be expected. An average of all the generated features could be a single reference representative of that texture. However, more than one representative for some texture pattern could be used. The output vector due to this input is assigned to be a code vector, where the code represents that texture whose features are the input. Such code vectors could be orthonormal vectors, i.e. vectors with all elements zero, except one. It is obvious that the dimension of these vectors equals the total number of textures in a class. Other code vectors could be used.

In summary, for the training stage, the training input X is formed by combining all the reference feature vectors of the textures in the class, where the training output Y is their corresponding code vectors. Using these training inputs and outputs, the associative memory matrix M is generated. During classification, an unknown feature vector, which is generated using the proposed algorithm, is

passed through the LAMM filter. This is done by simply multiplying the unknown vector by the M matrix and examining the result. Assuming that orthonormal code vectors were used, the identity of the unknown texture is determined by finding the maximum element in the output and noting its position, and the position of this maximum is the desired information. In the event that more than one equal maximum values exist, then unsuccessful classification is declared.

One very important aspect of LAMM is the linear independence of the input vectors. For LAMM to be successfully used, the input vectors should have a high degree of linear independence. If an input vector is linearly dependent on the others, then its orthogonal projection produced by the Gram-Schmidt orthogonalization process will be zero, and this vector will not be included in the iterations that generate the M matrix, and therefore in the recall, when a degraded version of this vector is inputed to the LAMM filter, the system will fail to recognize it. The result will be an incorrect output with possible incorrect classification.

In the proposed texture classification algorithm, the output is a 3-dimensional feature vector. We recall that the feature space was reduced to 3-D by introducing some

statistical reduction. Since all the input feature vectors x_k are 3-dimensional, for m texture patterns, the input matrix X will be of dimensions 3 x m. It is now obvious that of the m columns of the X matrix, no more than three can be linearly independent, and therefore, only three texture patterns can be classified correctly, which is a major drawback.

There are two ways to overcome this problem. For small m, i.e. not too many textures in a class, we can increase the dimensionality of the feature space, so that it will be at least m. Even though this method does not guarantee the linear independence, for very different texture patterns, their feature vectors will be sufficiently independent However, for many different textures in a class, not so many features will be available and this method is not feasible. Another alternative is to form an extended input matrix by stacking the columns of the output below the columns of the input. The extended input X' will be of the form

$$X' = \begin{bmatrix} X \\ \cdots \\ Y \end{bmatrix}$$

and the dimensions of X' will be of (n+m)xm, where n is the number of features used for classification and m is the total number of texture patterns in the class. Furthermore, if orthonormal code vectors are used, i.e. the Y matrix is

an identity matrix, then all the columns of X' will be linearly independent. During recall, since the code part of the input is unknown, all those elements are set to zero. The associative memory filter is now called to reconstruct the missing part of the input. In this sense, this is an autoassociative recall.

There are two disadvantages to this scheme. One is that since the input has larger dimensions, the M matrix will also be of larger dimensions, i.e. more computation. More specifically, the M matrix will now be of dimensions mx(n+m) as compared to mxn, where m and n were specified earlier. A second disadvantage is that since now the associative memory filter is required to tolerate for two sources of errors, namely classification algorithm errors plus the error due to the missing code in the input vector, the classification process will be less accurate, i.e. more incorrect or unsuccessful classifications can be expected. This second disadvantage can be corrected by using iterative associative memory encoding. This new scheme is described in the next section.

6.a ITERATIVE ASSOCIATIVE MEMORY ENCODING

Iterative associative memory is developed to improve the performance of the overall encoding system, by taking into account the deterministic error of the missing code, as was discussed earlier. In this method, a number of iterations are performed, and during each new iteration a better mapping M matrix is constructed. During the first iteration, the first matrix M_0 is calculated based on the training input X' and output Y, where again

$$X' = \begin{bmatrix} X \\ \cdots \\ Y \end{bmatrix} \quad \text{and } Y = I \qquad (6.a\text{-}1)$$

and X is the feature vector matrix and I is the identity matrix (for orthonormal codes). The first matrix M_0 will be of dimensions mx(n+m) where again n is the number of features used, and m is the total number of textures in the class. Without taking into account errors (perfect algorithm), all the unknown inputs will be of the form:

$$X_0 = \begin{bmatrix} X \\ \cdots \\ \emptyset \end{bmatrix} \qquad (6.a\text{-}2)$$

where the code part Y has been set to zero.

The recall will be of the form

$$Y_0 = M_0 X_0 \qquad (6.a\text{-}3)$$

where Y_0 is expected to be a close approximation of the desired output Y. It is now possible to use the output Y_0 as a new training input and with Y as training output (the target output) form a new associative memory M_1, so that

$$Y = M_1 Y_0 \qquad (6.a\text{-}4)$$

substituting (6.a-3) into (6.a-4) yields

$$Y = M_1 M_0 X_0 \qquad (6.a\text{-}5)$$

The matrix $M=M_1M_0$ is now a better mapping matrix that will map the incomplete inout X_0 closer to the target output Y. We also note that the matrix M_1 is of dimensions mxm so that the matrix $M=M_1M_0$ is still of dimensions mx(n+m). Also, since now the training input and output are close to each other, the matrix M_1 will be close to the identity matrix.

The above procedure can be repeated. Using the new matrix M we can find a new output Y_1, i.e.

$$Y_1 = M_1 M_0 X_0 \qquad (6.a\text{-}6)$$

where now we expect that Y_1 will be an even closer approximation of the desired output Y. Using now Y_1 as a training input and Y (the desired output) as the training output, a new M_2 matrix can be generated. Since now Y_1 is even closer to Y, then we expect that M_2 will be even closer to the identity matrix, and the new, improved mapping matrix will be $M=M_2M_1M_0$.

If this procedure is repeated several times, then the training input and output will be so close to each other that their associative memory matrix will converge to the identity matrix. After k iterations the overall mapping matrix will be

$$M = M_kM_{k-1} \cdots\cdots M_1M_0 \qquad (6.a\text{-}7)$$

where matrices with higher index are closer to the identity matrix. The M matrix in eq.(6.a-7) will map the unknown input to the closest much better than an M matrix with only one iteration.

Two possible variations of this scheme are the following:- In eq.(6.a-2) we can add some noise in matrix X so that algorithm imperfections can be taken into account. A second variation is instead of using the approximate

output as training input for the next iteration, we can use the combination of feature matrix-approximate code, i.e. the training input during iteration k will be of the form

$$\begin{bmatrix} X \\ \cdots \\ Y_{k-1} \end{bmatrix} \quad \text{instead of } Y_{k-1} \qquad (6.a\text{-}8)$$

The combined M matrix in this case could be the result of the k^{th} iteration, since the associative memory matrix will not converge to the identity matrix.

One basic problem with this iterative associative memory encoding scheme is again linear independence. If at any iteration, one training input vector is linearly dependent on the others, then that column of the M matrix will not converge to the correct one, and the corresponding output code will be incorrect. Current research is undergoing on modifications of the associative memory algorithm which behave well even in the presence of linear dependence, and results are pending.

7. POLYGONAL SHAPE IMAGE SEGMENTATION

Since each object contour is formed by straight line segments, the Hough technique [44] can be used for line detection. First, from a full-tone image the object contour is extracted by particular edge detection. A number of edge detection methods are available [40]. The edge image will be primarily of noisy straight line segments. Corresponding to the line segments the HT of the edge image will have its peak values at the appropriate normal parameters. These peaks can be extracted either by thresholding, or other techniques [74]. However, these peaks do not reveal information on the segment end points. Furthermore, a HT peak could correspond to many co-linear line segments. While some information on the overall length of the segment can be extracted from the value of the peak, this information could be false because the magnitude of the peak depends both on the line length and the thickness. To find the end points of each line segment, we must return to the edge image. On this image, based on the Hough transform (HT) peak, a window is moved along every predetermined line and the beginning and the end of each line is recorded. In addition, the number of line segments in a given direction is also determined. The result will be a list of all edge image line segments, with their normal parameters and

corresponding end points. An intelligent system can use this information to determine the line segments that belong to an object and the unrelated lines. For example, line segments which have their end points close together may be considered boundaries of the same object. Possible mismatch will be due to inaccuracies and errors.

Using this information, it is possible to segment the image into regions defined by the contour of each object. This can be accomplished by building a mask having the same shape as the object of interest. The mask is used to block off the remaining image. The result will be an image that shows the region of interest. The mask is constructed by drawing those line segments that match the edges of the object. However, when drawing lines that match the edges, because the edges do not represent clearly defined thin lines, and there are errors in detecting the position of the edge, there will be some positional inaccuracy. One technique to correct for these errors is to draw the line segment so that the distance between the line and the edge pixels will be a least-square sense minimum [74]. However, in this case, it is not essential to match the edges closely, but it would be sufficient if the object contour is inscribed in the region defined by the drawn lines. For this reason, and to account for errors, the mask is slightly

enlarged so as to ensure that it will cover the entire object of interest. To construct the mask starting with the normal parameters of the object sides, the p-value of each line is varied in order to produce an enlarged contour. To form a solid mask, the entire region inside the closed contour is painted white. To extract the object or its edges, we then logically AND the mask with either the original or the edge image. Similarly, for every object a mask is produced.

Additional information about the shape of the object can also be extracted. If, for example, the object is a triangle, its type can be examined by consulting its normal parameters. For an orthogonal triangle, two of the θ - parameters should differ by 90°. Similarly for a 4-sided figure contour, a trapezoid, parallelogram, rectangle, square, etc., its shape can be determined.

8. EXPERIMENTAL RESULTS

In this section experimental results will be presented in the same order as they appeared earlier. Therefore, first all the results that are median filter related will be presented, with results related to textures to follow, and finally results on image segmentation.

The imaging equipment used to generate these results consisted of a video camera connected to a high-speed A/D converter. With this facility, almost an entire NTSC frame (512 lines instead of 525) was able to be grabbed and stored into a buffer which was capable to store two independent frames. The content of the buffer was interfaced to a DEC VAX 11/750 computer system, where all the processing was performed. All the input and their results could be stored on computer memory. The result of processing could be viewed on a monitor by transferring the data from the computer back to the same buffer and to a high-speed D/A converter. Appropriate time correctors were also included in the imaging system. Color images were entered as three RGB color components, and the composite NTSC frame was generated in the computer through appropriate software. We proceed now by presenting median filter related results.

Using 512x512 8 bits per pixel digital images as input signals, the visual performance of the VMF as compared to that of the MF is tested. The purpose of these experiments is to demonstrate some of the visual properties of the VMF using realistic images. Fig. 6.a shows an image with computer generated additive uniform pseudo-random "salt and pepper" noise (see Fig. 6.b). The noise consists of uniformly distributed ten black or white spots per horizontal line. This noisy image is then processed with different median-type filters and their performance is evaluated both visually and by their computer run-time.

As was mentioned earlier 1-D filters, like the MF, were used to filter 2-D signals by first filtering the rows of the signal, and later the columns of the result. [6,7] Since the VMF is basically a 1-D filter, a similar scheme can be used for filtering images. However, since the purpose of the experiments is to compare the performance of the VMF to that of the MF, for simplicity only the horizontal lines of the image were filtered. This is equivalent to filtering the sampled video signal.

Figs. 6.c-6.e display the VMF of dimensions 3x1, 4x2, and 5x3, outputs respectively. Each filter is able to remove one sample wide impulses. Note that the VMF_{3x1} is a

MF. While visually all three images seem identical, their computer run-times, shown in Table 1, are quite different. For example, as compared to the MF_3, the VMF_{5x3} requires much less processing time. Figs. 6.f and 6.g show the output of a VMF_{5x1} (MF_5) and VMF_{6x2} respectively. Because now two samples wide impulses are removed, virtually all the impulsive noise is removed. Again, inspecting Figs. 6f and 6g we note no significant visual differences, except that the VMF_{6x2} is computationally faster than the MF_5. Fig. 7 summarizes the speed improvement of the VMF in comparison to the speed of its counterpart MF. Fig. 8 presents a more detailed image example. Here, some image degradation can be observed. For example, by comparing Fig. 8.c with Fig. 8.e, more degradation occurs as additional elements are included in the VMF. By also filtering the vertical lines, some of this degradation can be corrected. However, the difference is not very noticeable.

Similar experiments were performed on color images. Here, for each color image the same type of noise as for the black and white images was added to the three RGB color components. This noise has the appearance of different color spots in the composite NTSC image. Rather than median filtering the composite NTSC signal, a better result is obtained when the VMF-ing is performed on the RGB color

components. The color VMF performance was similar to the black and white VMF image processing performance. Compared to the MF, again the VMF has the faster processing time.

The performance of the 2-D VMF was tested using the same images as in the 1-D VMF. The window used was of rectangular shape with the median vector being a stripe (quasi-2D VMF) of horizontal or vertical orientation. The performance obtained using the horizontal or vertical median vector was found to be almost identical. This is expected because using a vertical stripe is equivalent to using a horizontal stripe, and transposing the image. Since the performance was in both cases very similar, in the subsequent discussion only horizontal stripes will be considered. The filter window size is denoted by N1 x N2 x M where N1 is the vertical dimension, N2 is the horizontal dimension, and M is the horizontal median vector dimension. In addition, N1 is restricted to be odd, where N2 and M are restricted to be both odd or even. Also note that a 2-D filter of dimension N1=1 is actually a 1-D VMF. The trend test originally used was the derivative in the horizontal direction, i.e., the same one used for 1-D VMF. However, the performance of this test was acceptable only for two-element median vectors, and quite unacceptable for larger ones. To improve the performance, the next trend

test that was tested was the directional derivative in both horizontal and vertical directions. In this test, each pixel is assigned two tags depending on the difference of that pixel and the adjacent ones in the horizontal and vertical directions. If the difference is positive (negative) then the value of each tag is +1 (-1) and zero for equal values. For the trend test the values of the tags of all the elements within the filter window are added. If the result is positive (negative) then the direction of sorting is increasing (decreasing), and arbitrary for zero result. This trend test performed satisfactorily for median vectors up to three elements wide. Since this is sufficient for most practical applications, no further searching was done.

Fig. 9 displays the result of different filters applied on the noisy image of Fig. 6.b. Visually all the images of similar window dimension appear to be similar. For example, the result of a 3x3x1 VMF looks very similar to that of 3x4x2, where the result of 5x5x1 is very similar to that of 5x6x2. Fig. 10 summarizes the speed improvement of various 2-D MFs obtained by using their closest quasi-2D VMF extensions. Table 1 displays computer run times for different window dimensions. Also the same table displays the RMS error between the original image and the filtered

one for different window sizes. The results of table 1 are also plotted in Fig. 11 for visual inspection. By inspecting both table 1 and Fig. 11 it is apparent that with a small penalty in RMS error, and with very few noticeable differences, significant time saving is possible by using a VMF in place of a MF. Additionally, as was also discussed earlier, the VMF being a multiparameter filter, it offers a wider choice of filtering choices. As an additional illustration, Fig. 12 is another example of VMF filtering where again similar comments can be made.

The performance of the fast algorithm was tested by filtering an image with two identical VMF's. One VMF was implemented using standard sort, where the second one was implemented using the fast algorithm. The figure of merit used was the computer run time. The results are summarized in table 1 and also plotted on Fig. 11. By inspecting Fig. 11 (or table 1), two general comments can be made. One is that with the fast algorithm, for larger windows there is a tremendous improvement in computer time. In addition, by observing Fig. 11 we can see that there is a drastic increase in computer time when standard sort is used, while when using the fast algorithm the computer time increases very slowly with increasing window dimensions. The second comment is that even when the fast algorithm is used, there

is a further improvement when using a VMF in place of a MF. It is estimated from table 1 that the percent improvement by using a VMF in place of a MF is approximately the same for both the standard or fast algorithm. Therefore, Fig. 10 applies for both standard and fast algorithms. The result is that when the VMF is combined with the fast algorithm, it becomes a very efficient processing tool.

Experimental results on signal restoration using median filter and transform are now presented. In these experiments a signal was corrupted by sinousoidal interference and possibly by random noise, and the objective was to recover as closely as possible the original signal from the noisy one. The three signals used in these experiments were a decaying exponential, a pulse, and a square-wave. All signals were sampled at a rate of 128 samples per second, and also a 128 points DFT was used. Better results were obtained by median filtering the real and the imaginary components of the transform rather than filtering the magnitude and the phase.

Fig. 13.a displays a decaying exponential where Fig. 13.b is the same signal with four sinousoids added. The interfering signals are of frequencies 20Hz, 30Hz, 40Hz and

60Hz with amplitudes 1V, 2V, 2V, and 1V respectively. Fig. 13.c displays the real and imaginary parts of the FFT of the clean signal, where Fig. 13.d displays the same information for the corrupted signal. Fig. 13.e is the result of filtering Fig. 13.d using a MF_{11} and Fig. 13.f is the inverse FFT of Fig. 13.e, which is the recovered signal. For the purpose of comparison, Fig. 13.g displays both the original and the recovered signal, where a very close match can be noticed. Some further improvement is possible by MF post-filtering the recovered signal to remove residual oscillations. Fig. 13.h displays a recovered exponential where a MF_7 was used in the frequency domain and the MF_5 was used for post-filtering.

Similar experiments were performed with a single pulse. Fig. 14.a displays a two sample pulse, where Fig. 14.b shows the pulse corrupted by the same interfering signals as in the exponential signal experiments. The spectrum of the noisy pulse was MF_5 filtered, and Fig. 14.c displays the inverse FFT of the filtered spectrum. We can notice from Fig. 14.c that the pulse is very satisfactorily recovered. Post-filtering by a MF is not appropriate in this case, since the MF will eliminate the pulse. Thresholding will be a more appropriate type of post-processing.

More experiments were performed using square-waves. A square-wave could represent a binary signal which was corrupted in the transmission path. Fig. 15.a displays a square-wave, and Fig. 15.b is the same signal with interference added. The type of interference is the same as the one used in earlier experiments. Fig. 15.c displays the magnitude of the Fourier transform of the clean square-wave, where a large DC component is noticed, followed by the fundamental frequency pulse harmonics. Fig. 15.d displays the spectrum of the noisy square-wave, where the interference is evident by the additional peaks in the transform. Fig. 15.e displays the result of MF_3 filtering of the noisy transform. While it seems that the signal information is lost, Fig. 15.f which is the inverse FFT of Fig. 15.e shows that a very good approximation of the square-wave is recovered. For the purpose of comparison the same figure simultaneously displays the original square-wave. Fig. 15.g displays a recovered vs. the original signal when a MF_7 was used. Further improvement is possible by post-filtering with a MF. Fig. 15.h shows the result of a MF_{13} applied on the recovered signal of Fig. 15.f where we notice that most of the residual noise is removed, leaving an excellent approximation of the original signal.

In another set of experiments, random noise was included in addition to the sinousoidal interference. Fig. 16.a shows a square-wave with uniform noise added. Fig. 16.b displays the noisy square-wave with the same sinousoidal interference used in the earlier experiments. Figs. 16.c and 16.d display the spectrum of the noisy square-wave and the noisy one with the sinousoids added, where Fig. 16.e shows th result of MF_3 filtering. Fig. 16.f is the inverse FFT of the Fig. 16.e and displays the recovered square-wave. Fig. 16.g also displays a recovered square-wave using a MF_{11}. Comparing Fig. 16.g (the recovered signal) with Fig. 16.a (the original noisy signal) we observe that the recovered signal is less noisy than the original. Figs. 16.h and 16.i display the recovered signals of Figs. 16.f and 16.g with MF_{13} post-filtering, where again excellent performance can be observed.

Similarly experiments were performed in the case where the interfering frequencies were very close to the fundamental frequency of the square-wave. Fig. 17.a shows the noisy square-wave used in earlier experiment, with four sinousoids added. Their frequencies are 3, 6, 9, and 15 Hertz with amplitudes of 1, 2, 1, and 2 volts respectively. Fig. 17.b is a recovered signal using a MF_3 in the frequency domain, and a MF_{13} for post-processing. Figs. 18.a and 18.b

is a second example where the interfering frequencies are 2, 4, 6, and 10 Hertz, with amplitudes 1, 2, 1, and 2 volts respectively. In both experiments an excellent reconstruction is possible, although by observing Fig. 17.a and Fig. 18.a no clue is evident of the nature of the original signal. Finally Fig. 19.a represents the noisy square-wave with four sinousoids of frequencies 4, 6, 12, and 20 Hertz added. The amplitudes of all four sinousoids are 900 volts. This experiment represents high-power jammers, jamming a low-level noisy signal. Fig. 19.b is the recovered square-wave using a MF_{11} in the frequency domain followed by a MF_{15} for post-processing. Even though the reconstruction performance in this case cannot be considered as excellent, however, for many applications it can be judged as very satisfactory.

In the following paragraphs we present and discuss texture related computer simulation results. The first step in these experiments was the computation of the Hough transform of an input texture. The HT p-axis is quantized uniformly to 361 levels while the θ-axis is quantized uniformly to 90 levels. Since the 2D register has 32,490 accumulators, it would be impractical to print the value of each accumulator, and instead, visual display on the screen of a monitor was chosen. For this purpose, the entire HT is

inserted in an NTSC frame with the proper display format. The values of every row of the 2D register become pixel values of one horizontal scan line. Interlaced scanning is employed. Since the HT will not fill up the entire frame, the boundaries of the transform and the p-axis are drawn. Furthermore, divisions on the θ-axis are displayed as bright ticks. The texture to be transformed is entered into the computer through a video camera and digitized in the form of a 512x512 array quantized to 8 bits per pixel. The NTSC sync information is also included in the array. The computer used for these experiments was a DEC VAX 11/750 system.

Figure 20. shows the reference axes used in generating the HT. Fig. 20.a represents the texture in the x-y plane. The texture is assumed to be a rectangular of dimensions 2Xm x 2Ym, with the origin located at its center. By locating the origin at the center of the texture, a better utilization of the p-θ plane is possible. Fig. 20.b represnts the p-θ (HT) plane as it will appear in the following illustrations.

Figure 21.a shows the texture image of French canvas (Brodatz's plate No. 20). Notice the line structure of this texture. Figure 21.b is the HT of a scaled block of this

texture. It is evident that a smaller block of the texture contains the same structural information as a larger one.

In Figure 21.b the leftmost part of the transform corresponds to $\theta = 0°$ and the rightmost to $\theta = 178°$. The p=0 axis is indicated by the horizontal solid line around the center of the transform. The bright spots around $\theta = 0°$ and $\theta = 90°$ correspond to peak values in the transform. Note that the spacing of the peaks follows the arrangement of the lines in the texture. Note also the background noise, especially at the center of the transform, because of "crosstalk" between lines.

As was mentioned earlier, preprocessing the texture by thinning the lines (skeletoning), will increase the classification accuracy. Figure 21.c is the HT of the same texture block used for Figure 21.b, but having undergone a process of line thinning. Comparing Figure 21.b with 21.c, we see that the peak values corresponding to texture lines are located at the same position, but in Figure 21.c they are much sharper, and the background noise (crosstalk) is significantly lower. It is obvious that even though both Figures 21.b and 21.c reveal the same information about the structure of the texture, this information can be more easily and accurately extracted from Figure 21.c. Finally,

Figure 21.d is another block of the same texture under a different scale, which has been preprocessed for line thinning, and rotated by 45°. Here we note that the sharp peaks are now located around θ=45° and θ=135°, where the spacing of the peaks again it follows the spacing of the lines in the texture.

Figure 22.a is another texture of Herringbone weave (Brodatz's Plate 17), where Fig. 22.b is the HT of a preprocessed block of this texture. Notice again that the location of the peaks correspond to the locations of the lines in the texture. We also note that since the lines around θ=45° and θ=135° are actually non-contiguous line segments which are equivalent to a shorter line, that those peaks appear to be smaller compared to the peaks at θ=0° which correspond to a long line.

It is interesting to examine what would be the effect of noise on the HT. Figure 23.a is an idealized test pattern and Fig. 23.b is the HT of Fig. 23.a. Figure 23.c shows the texture of Figure 23.a with computer generated psuedo-random uniform "salt and pepper" noise added. We can predict that the additional noise pixels will add more sinusoidal curves in the transform domain, and the result will be that more background noise will be present in the

transform. Figure 23.d shows the HT of the noisy texture of Figure 23.a. Comparing Figure 23.d with Figure 23.b, i.e. the HT of the texture without noise, we notice that indeed the noisy transform has larger background noise. However, the peak values can still be extracted from the noisy background. The conclusion is that the HT technique is immune to noise, since even for noisy textures we can still isolate the HT peak values.

As an additional illustration, Fig. 24.a shows the pattern of Fig. 23.a under severe scaling (and translation). From Fig. 24.a it is evident that there is still enough information present for correct classification. Fig. 24.b is the HT of Fig. 24.a. We observe that basically only two lines were detected in each orientation. The feature vectors of the textures of Fig. 23.a and Fig. 24.a were calculated using the proposed classification algorithm and were found to be:

For Fig. 23.a $\underline{x} = [2, 0.014, 0.290]$

For Fig. 24.a $\underline{x} = [2, 0.019, 0.283]$

We observe that the two views of the same texture yield very similar feature vectors.

For classification, the feature vector of each texture is computed following the steps of the classification algorithm. The peak values in the transform were isolated using both simple thresholding and median filtering. Both approaches performed well, but the median filter was slightly superior. The important result to examine is whether different views of the same texture yield the same feature vector, and if different textures yield different feature vectors. Since the first element of the feature vector is the number of orientations in the texture, then textures with different numbers of orientations will certainly yield dissimilar feature vectors.

Experiments performed showed that different views of the same texture always yield almost the same feature vector. The critical part was what would happen with different textures with the same number of orientations. Experiments showed that there was sufficient dissimilarity in the feature vectors for correct classification. Figure 25 is a scattering diagram of three textures, all with two orientations. Two of them have lines intersecting at 90° and the third one has lines intersecting at 79°. We notice that the first two cluster around the same value of x_3 (angle variance), at different values of x_2 (distance

variance). From the scattering diagram we can see that the clusters are well separated in the feature space.

Table 2 depicts the application of the classification algorithm to two different views of the same texture. The first entry in the table is the result of searching for peaks in the transform and recording the angle at which they occur and the distance between them. We notice that the second view was rotated by 20° with respect to the first, and also that the second was taken at a smaller scale, since more peaks were found in each angle. The next two steps are to normalize all the distances by dividing by the maximum one, and to average the normalized distances in each angle. Next we circularly shift so that the maximum average (one in this case) will be at the leftmost position. Then we subtract (modulo-180°) the first angle from all the others and normalize all angles to 180°. After these steps we note that the entries in the table are almost identical. Then we form the feature vectors by calculating the variance (here the standard deviation was used) of the normalized distances and angles. Note that the first element of the feature vector is $x_1=3$ since three directions were detected in the texture.

Two types of experiments were performed on associative memory encoding. In the first type, eight different textures were described using nine-element feature vectors. In the second type the same eight texture patterns were described by three numbers. The nine-element feature vectors were obtained by deleting the last step of the classification algorithm which includes calculation of variances (see table 2). Since the texture patterns used had up to a maximum of four directions, a nine-element feature vector is formed where the first element is the number of directions detected, the next four are the normalized angles, and the last four are the average normalized separation in each direction. In the case where less than four directions are detected, the unused entries are filled by zeroes. For example, for the pattern depicted in table 2, the nine-element feature vector will be:

$$\underline{x} = [3, 0, 0.21, 0.79, 0, 1, 0.79, 0.78, 0]$$

The code vectors used were eight element orthonormal vectors. The reference feature vectors were obtained by running the algorithm four times on four different versions of the same texture pattern, and averaging the resulting feature vectors. Using these reference features as training inputs and their corresponding code vectors as training

outputs, an associative memory is calculated. The performance of this associative memory was tested by submitting an unknown feature vector resulting from running the algorithm on one of the texture patterns, or simply by adding noise on one of the training inputs and using that as test input. In both cases the test input is submitted to the filter and the desired information is obtained by locating the position of the maximum element in the output. In the event of two equal peaks, then a "don't know" is declared.

Extensive testing of this scheme found it to be very successful, with an average percentage of 95% correct classification. Most of the unsuccessful attempts were "don't knows" with some incorrect cases observed. As an illustration, for one of the texture patterns used, the reference feature vector was:

Training input:

$\underline{x}$ = [3, 0, 0.2, 0.8, 0, 1, 0.77, 0.79, 0]

The corresponding code vector was:

Training output:

$\underline{y}$ = [0, 0, 0, 1, 0, 0, 0, 0]

The testing input was:

$\underline{x}$ = [3, 0, 0.26, 0.84, 0, 1, 0.70, 0.83, 0]

The resulting output was:

$\underline{y}$ = [0.4, -0.2, 0.3, 0.8, 0.1, -0.1, 0.6, 0.2]

During the second type of experiments, only three numbers were used to described the patterns (the calculation of the variances was included). The training input was formed by stacking the three classification numbers with the code vector into a single vector, where the training output was again the code vector. The reason the code vector was stacked with the feature vector was to increase the rank of the training input, as was explained in detail earlier. The calculation of the associative memory was done the same way as was described in the previous experiments. For the testing, the unknown feature vector was stacked to a zero vector and submitted to the filter, and again the desired information was obtained by finding the position of the maximum element in the output vector.

Again extensive testing found this scheme to be less successful than the one described earlier; however, the performance was very satisfactory. The average percentage of success was estimated to 70% which could be increased to about 80% by using iterative encoding. It should be noted that the reported success of other far more elaborate

algorithms is averaging values varying from 46% to 82% maximum with a median value of around 60%. In addition, compared to elaborate clustering algorithms, this method is simpler and more formal. One additional comment is that in this type of experiments more incorrect classification cases were noted. About half of the failures were incorrect classification and the other half were "don't knows". Again, for the purpose of illustration, the training input of a pattern was:

$\underline{x}$ = [3, 0.12, 0.41, 0, 0, 0, 1, 0, 0, 0, 0]

The training output was:

$\underline{y}$ = [0, 0, 0, 1, 0, 0, 0, 0]

The testing input was:

$\underline{x}$ = [3, 0.11, 0.42, 0, 0, 0, 0, 0, 0, 0, 0]

The resulting output was:

$\underline{y}$ = [0.2, 0.3, 0.1, 0.7, -0.2, 0.5, 0.4, 0.1]

When iterative encoding was used the resulting output was:

$\underline{y}$ = [0.1, 0.2, -0.1, 0.8, 0.1, -0.1, 0. -0.2]

Note that this case represents a successful classification case.

Computer experiments on polygonal shape image segmentation consisted of segmenting an image with some geometrical figures into a number of images containing only one figure. Fig. 26.a displays a binary image of a triangle and a 4-sided figure. Notice that the two figures do not have any colinear sides. Fig. 26.b is the result of Sobel edge detector with some threshold applied. Some edge-thinning algorithm could be applied as further post-processing, but in these experiments it was not. Fig. 26.c is the HT of Fig. 26.b where Fig. 26.d is the thresholded HT of Fig. 26.c. From the peaks in the HT the vertices of each figure were found in the edge image, and two masks were able to be constructed. Figs. 26.e and 26.f represent the two generated masks which have the same shape as the original figures, but they are slightly enlarged to compensate for location inaccuracies. Using these two masks it is easy to extract each figure by logically ANDing the mask image with the original or edge images.

As an additional illustration of the above, Fig. 27.a shows a similar binary images as in Fig. 26.a with some noise added. The noise is computer generated pseudo-random uniform 'salt and pepper' noise. Also some stray line segments were added and are considered for this experiment as noise. While most of the 'salt and pepper' noise could

be removed by median filtering, we do not do so since we wish to examine the performance of the algorithm under noisy conditions. Figure 27.b is the result of the Sobel operator with some threshold. We notice that the noise is now more pronounced. Figure 27.c is the HT of Fig. 27.b. Here the effect of the 'salt and pepper' noise is a noisier background in the HT. However, because the peaks are still higher than the background and therefore even with simple thresholding, it is possible to extract them. Figure 27.d displays the threshold HT of Fig. 27.c. The process of finding segment end points is now complicated by the presence of the noise. However, when tracing a line searching for bright pixels, we ignore isolated pixels and we look for sequences of bright pixels. A similar strategy is followed when we look for dark pixels that mark the end of a line. Following this scheme the effect of noise can be minimized. To extract the two figures of interest Figs. 27.e and 27.f represent the two masks generated, where Figs. 27.g and 27.h represent the segmented images. Some residual noise can be observed around the contour of each shape and is the result of the enlarged masks.

Similar experiments were performed with images containing 3-D objects. Fig. 28.a is a pseudo-binary image of a cube and a triangle. Additionally, the two objects

have two colinear sides. Fig. 28.b is the thresholded Sobel operator and Fig. 28.c is the thresholded HT of Fig. 28.b. In Fig. 28.c eleven peaks can be detected, where ten of them represent line segments in the edge image, and the small peak in the center of HT image is an incorrect one to be rejected later. The colinear line segments are represented by the same peak. Here, the fact that more than two line segments meet at a point does not complicate the algorithm; however, the decision about the shape is more difficult. As a result of application of the algorithm, Fig. 28.d represents a reconstructed cube. The interior of this cube can be painted white, and be used as segmenting mask. However, in this case a simpler mask could be a rectangle formed by parallel line segments passing through the four extreme points of the cube. Such a mask is shown in Fig. 28.e where Figs. 28.f and 28.g display the segmented original and edge images. The triangle can be extracted by constructing a mask, as was described earlier.

9. SUMMARY

The contributions of this work in the related areas are summarized in this section. In the area of median filtering a new median-type filter, the vector median filter, with MF as its special case, has been introduced. The principal advantage of the VMF over the MF is that it is computationally faster while maintaining comparable visual performance. The speed improvement is due to faster window movement along the data array. If the signal contains extended constant regions, a further speed improvement is possible, because in this case we can eliminate the sort procedure. The sinusoidal response of a VMF filter was examined and found to have a smoother response than its MF counterpart. The statistical performance of the VMF was described and, using an example, the SNR performance of the VMF and MF were compared. Experimental results on actual images were presented. With visual performance comparable to the MF, the computational speed improvement of the VMF was demonstrated. Some image degradation was observed when the output vector was increased. This is expected and is due to the deposition of some samples in the filtered signal. However, there are classes of signals like images or speech signals where some distortion is tolerable, but

there would be always a trade-off between speed and degradation.

Vector median filtering was extended in two dimensions. The quasi-2D VMF which uses a 2-D filter window and a 1-D median vector was introduced and its performance was tested using realistic images. Computer simulations showed that again with performance comparable to that of the 2-D MF, the 2-D VMF was able to complete the filtering in less time. Some slight degradation was within tolerable limits. A fast algorithm for both 1-D and 2-D VMF's was developed. The performance of the algorithm was evaluated by comparing computer execution time, and it was found to result in drastic decrease of processing time. Overall, the VMF when implemented with the fast algorithm is a very effective tool for different types of processing applications.

A novel filtering scheme using MF and a linear transform was also introduced. A special case is when the linear transform is the Fourier transform. In this case the filter is able to remove sinusoids of unknown characteristics from signal with a relatively smooth spectrum. This is not very restrictive since a large class of commonly used signals have a smooth spectrum. Experiments on square-waves, pulses, and exponentials showed

excellent performance even for high levels of interference and in the presence of noise. Furthermore, this type of filtering can be done in real time since both FFT and MF integrated circuits are available. The result is an effective filter with many practical applications.

In the area of pattern classification a new structural approach for texture classification has been introduced. The texture primitives used for classification are the location and the relative orientation of lines in the texture. The Hough transform is used for line detection. This technique is computationally very efficient and can be applied for shapes other than lines, such as circles, parabolas, ellipses etc. If some preprocessing is applied on the texture so that wide lines become thin, then the method can be applied with excellent classification results. The feature extraction algorithm proposed is independent of geometrical transformations such as translation, rotation and scaling. Several experimental results on idealized and natural textures were presented. Also, experiments performed on noisy textures showed that correct classification is possible even in noisy environments.

Linear associative memory mapping was used to encode the output of the classification algorithm into classes. In

this new application, LAMM replaces tedious classification algorithms and time consuming similarity measures resulting in a very compact decision maker. Extensive simulation showed a high degree of accuracy which can be further increased by using the also new scheme of iterative encoding.

In the area of image segmentation, a new method of segmentation formulated for the class of images containing polygonal shapes was introduced. This method is based on edge detection followed by line detection using the HT. Noise and stray lines can be effectively suppressed. The same information that is used for segmentation can be furthermore utilized for shape recognition. Additionally, this approach can be generalized to other shapes by different preprocessing steps.

I L L U S T R A T I O N S

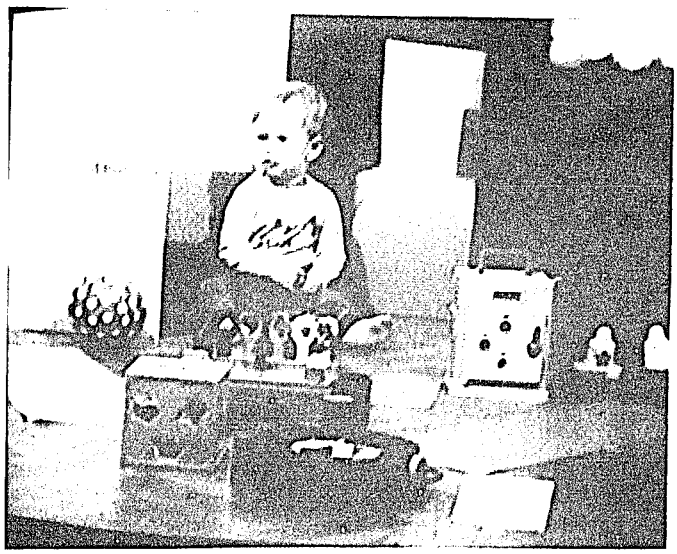

Fig. 6a. Source image.

Fig. 6b. Source image with noise added.

Fig. 6c. Noisy image filtered with a MF_3.

Fig. 6d. Noisy image filtered with a VMF_{4x2}.

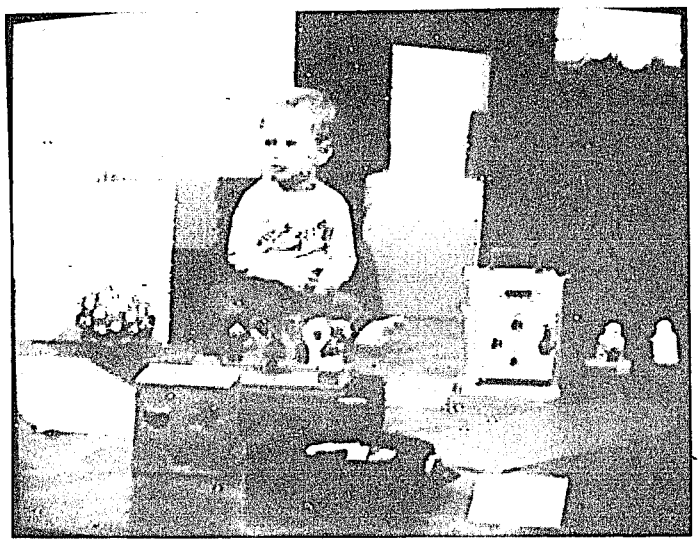

Fig. 6e. Noisy image filtered with a VMF_{5x3}.

Fig. 6f. Noisy image filtered with a MF_5.

Fig. 6g. Noisy image filtered with a VMF_{6x2}.

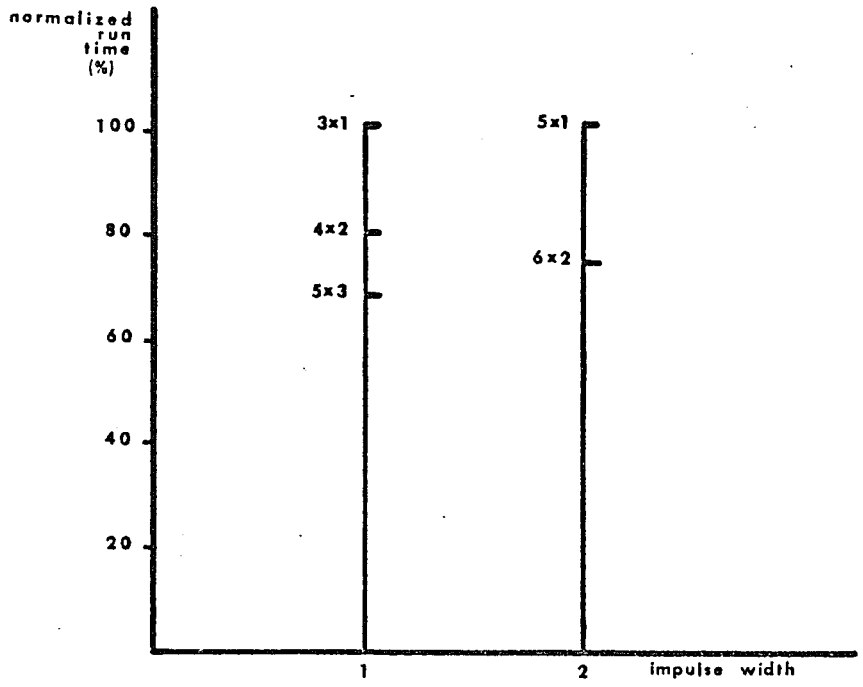

Fig. 7. Speed comparison of MF and VMF.

Fig. 8a. Original image.

Fig. 8b. Noisy image.

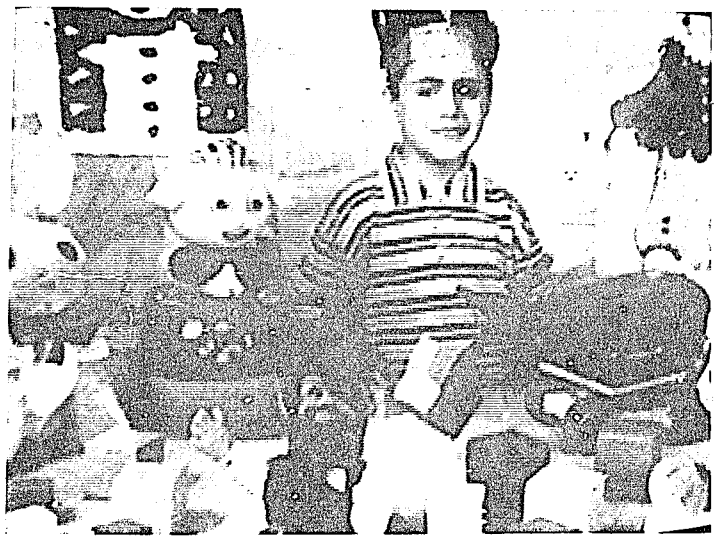

Fig. 8c. Output of a MF_3.

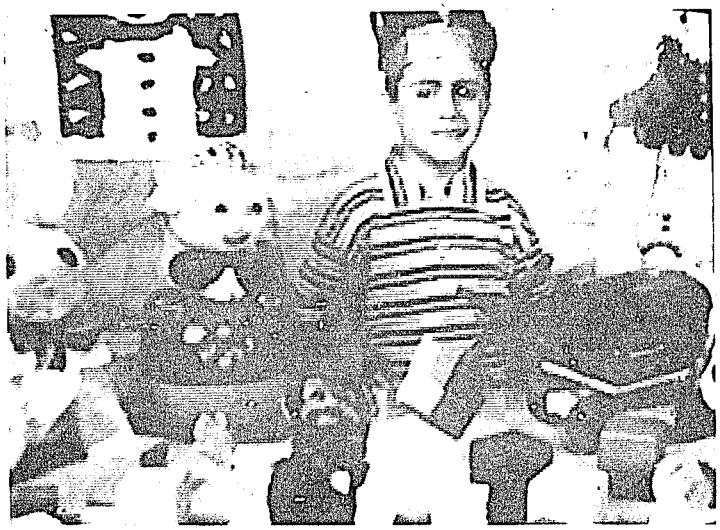

Fig. 8d. Output of a VMF_{4x2}.

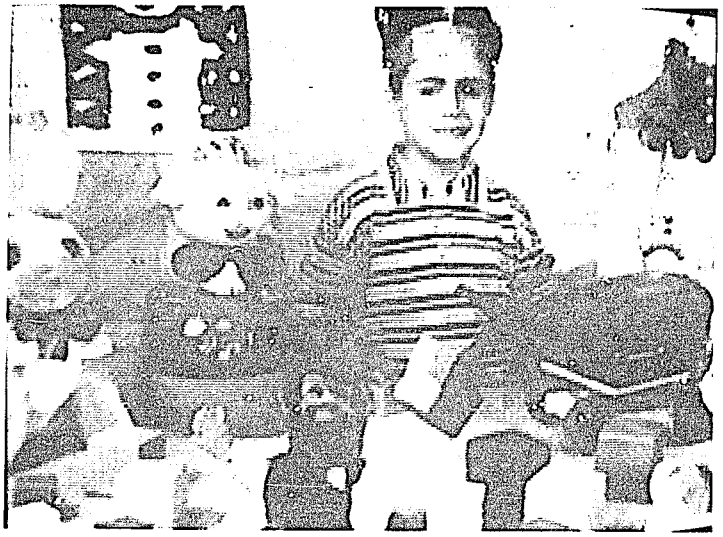

Fig. 8e. Output of a VMF_{5x3}.

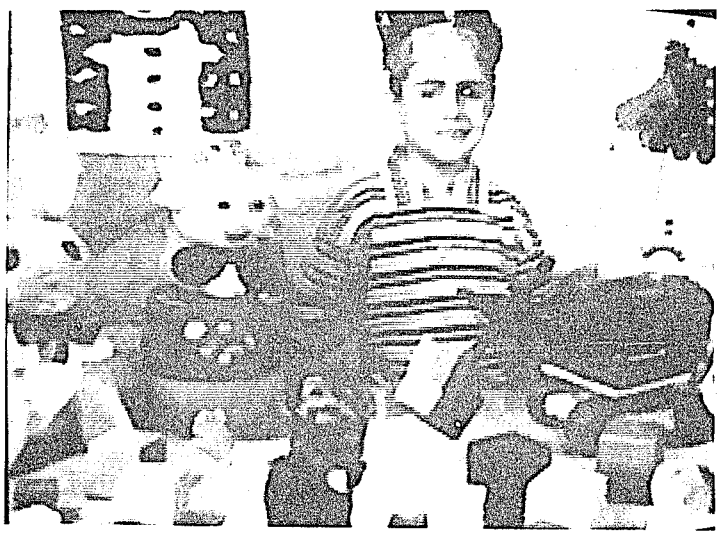

Fig. 8f. Output of a MF_5.

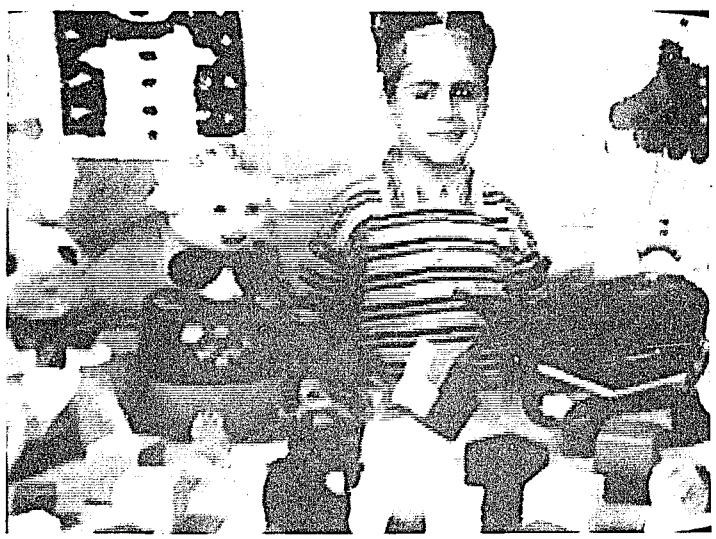

Fig. 8g. Output of a VMF_{6x2}.

Fig. 9.a. Output of VMF_{3x3x1} applied on the noisy image of Fig. 6.b.

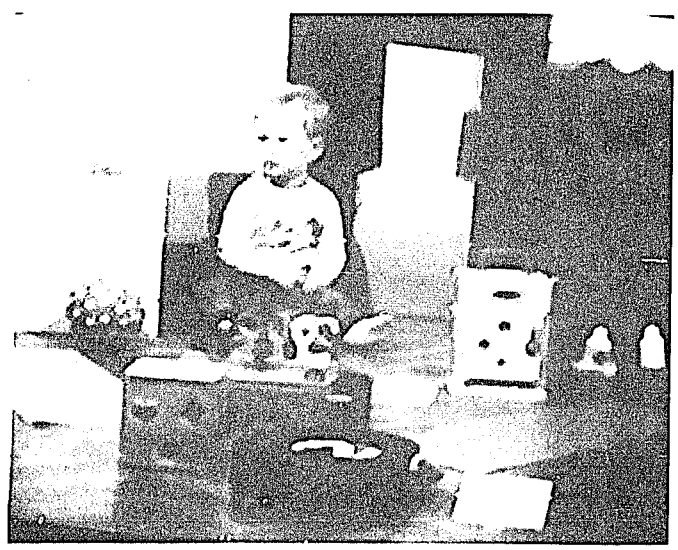

Fig. 9.b. Output of VMF_{3x4x2} applied on the noisy image of Fig. 6.b.

Fig. 9.c. Output of VMF_{3x5x3} applied on the noisy image of Fig. 6.b.

Fig. 9.d. Output of VMF_{5x3x1} applied on the noisy image of Fig. 6.b.

Fig. 9.e. Output of VMF_{5x4x2} applied on the noisy image of Fig. 6.b.

Fig. 9.f. Output of VMF_{5x5x3} applied on the noisy image of Fig. 6.b.

Fig. 9.g. Output of VMF_{5x5x1} applied on the noisy image of Fig. 6.b.

Fig. 9.h. Output of VMF_{5x6x2} applied on the noisy image of Fig. 6.b.

FILTER (N_1xN_2xM)	RUN TIME (SEC)		RMS ERROR
	STANDARD SORT	FAST ALGORITHM	
1x3x1	108	99	3.33
1x4x2	86	81	3.40
1x5x3	76	69	3.95
1x5x1	132	112	3.01
1x6x2	93	86	3.38
3x3x1	282	120	3.12
3x4x2	212	99	3.25
3x5x3	198	90	3.98
5x3x1	732	122	3.31
5x4x2	580	107	3.52
5x5x3	520	95	4.13
5x5x1	1208	129	4.16
5x6x2	892	109	4.87

NOTES: N_1 = Vertical window dimension.

N_2 = Horizontal window dimension.

M = Median Vector dimension.

RMS Error is between the original and filtered image.

Table 1. Computer run time and RMS error of different VMFs.

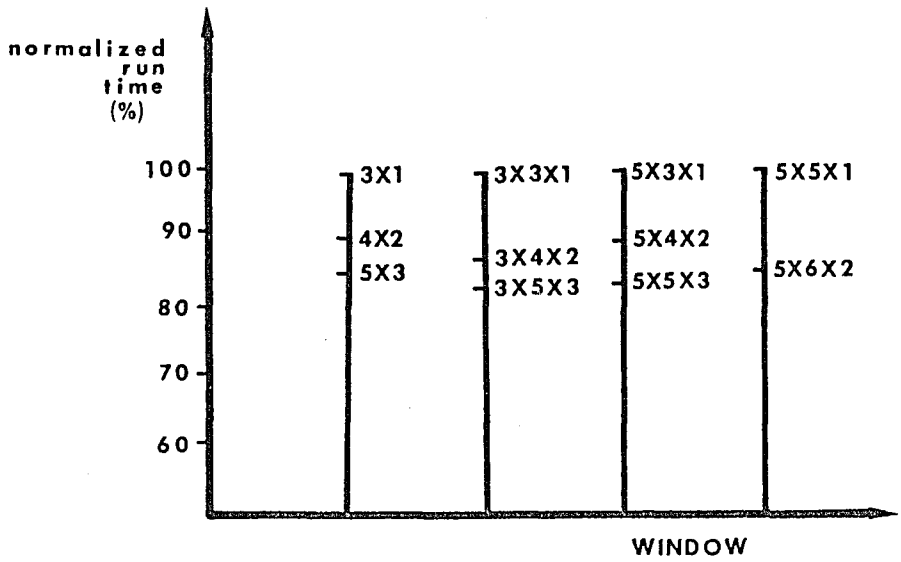

Fig. 10. Speed comparison of different VMF's.

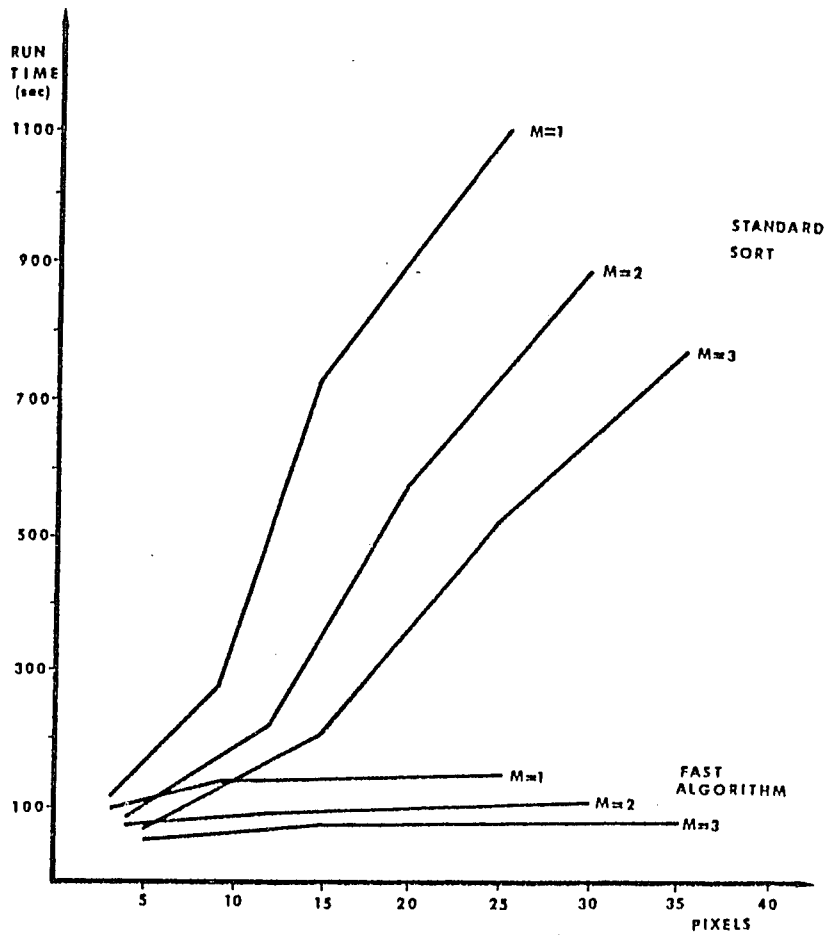

Fig. 11. Computer run time of different VMF's using both standard and fast algorithms.

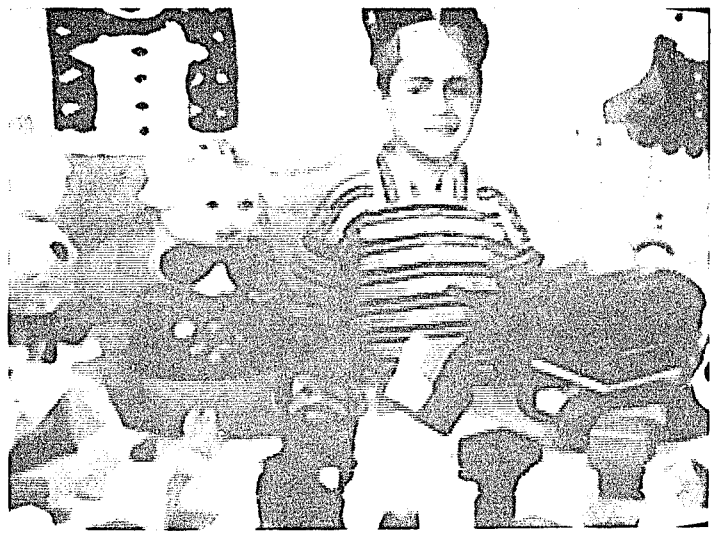

Fig. 12.a. Output of VMF_{3x3x1} applied on the noisy image of Fig. 8.b.

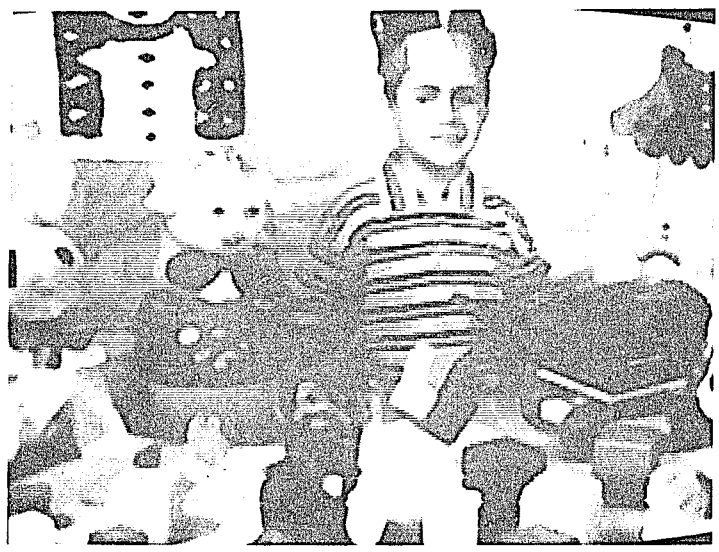

Fig. 12.b. Output of VMF_{3x4x2} applied on the noisy image of Fig. 8.b.

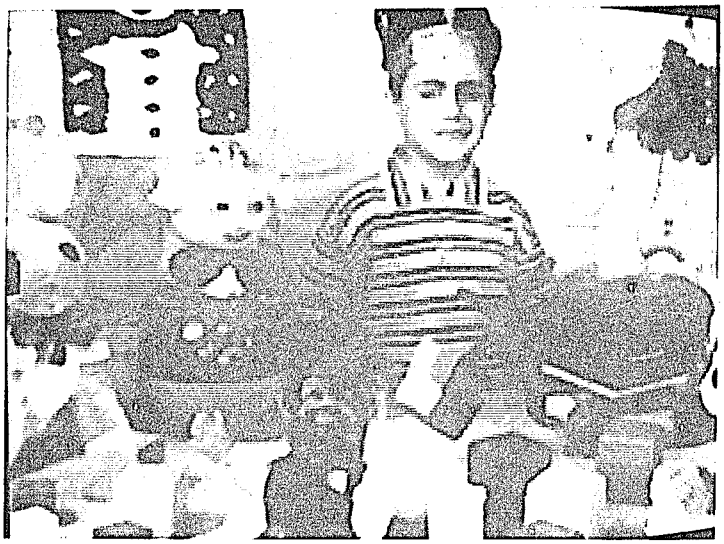

Fig. 12.c. Output of VMF_{3x5x3} applied on the noisy image of Fig. 8.b.

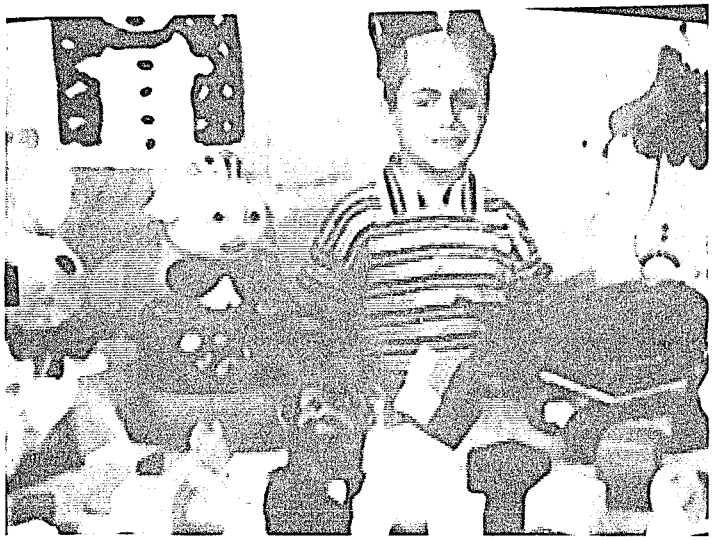

Fig. 12.d. Output of VMF_{5x3x1} applied on the noisy image of Fig. 8.b.

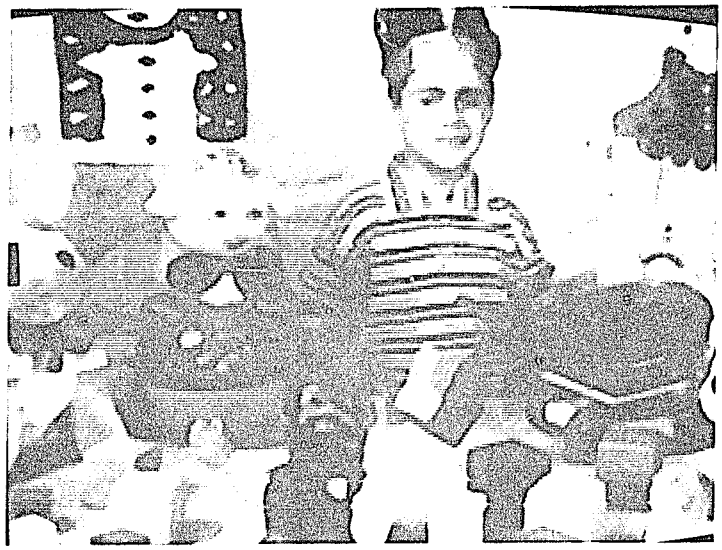

Fig. 12.e. Output of VMF_{5x4x2} applied on the noisy image of Fig. 8.b.

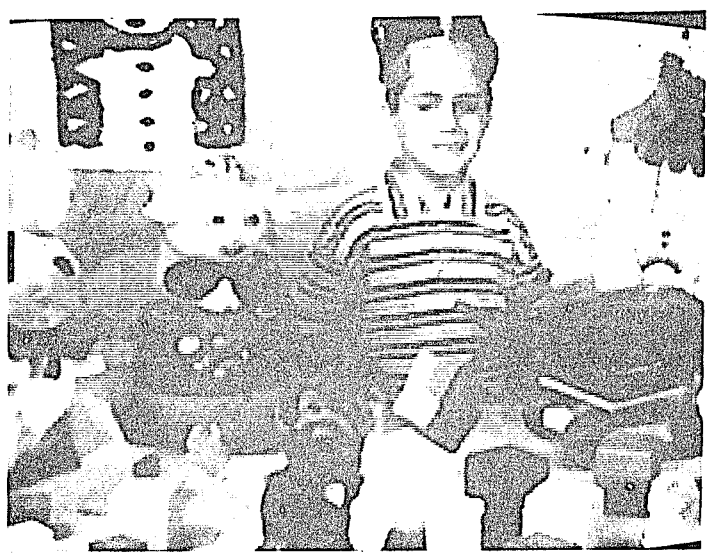

Fig. 12.f. Output of VMF_{5x5x3} applied on the noisy image of Fig. 8.b.

Fig. 12.g. Output of VMF_{5x5x1} applied on the noisy image of Fig. 8.b.

Fig. 12.h. Output of VMF_{5x6x2} applied on the noisy image of Fig. 8.b.

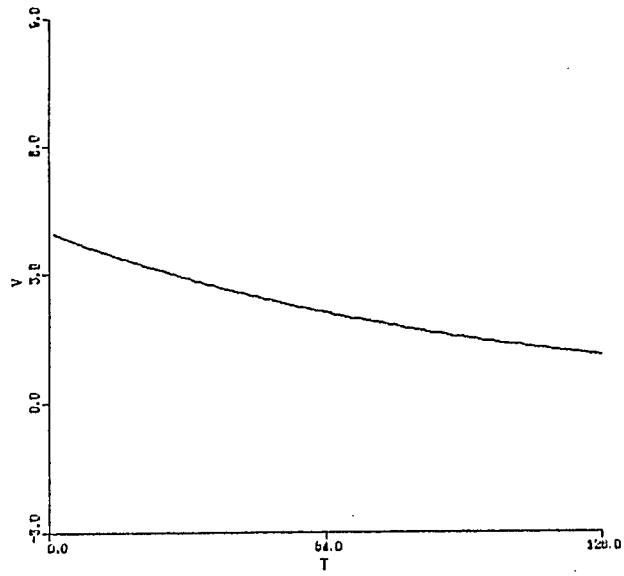

Fig. 13.a. A clean decaying exponential.

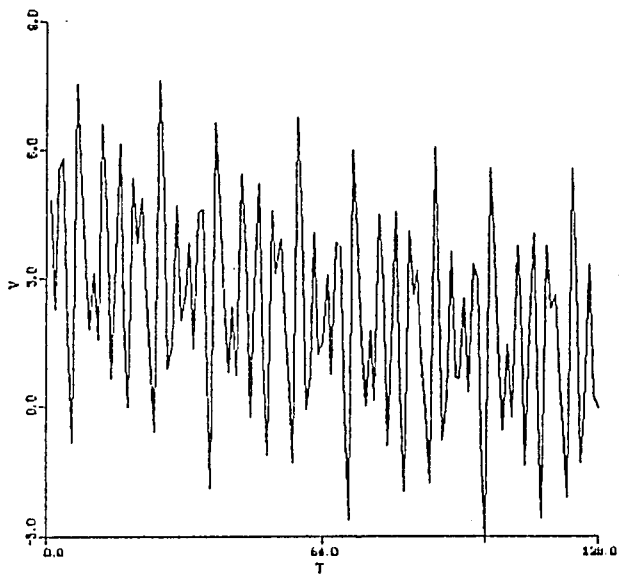

Fig. 13.b. Corrupted exponential.

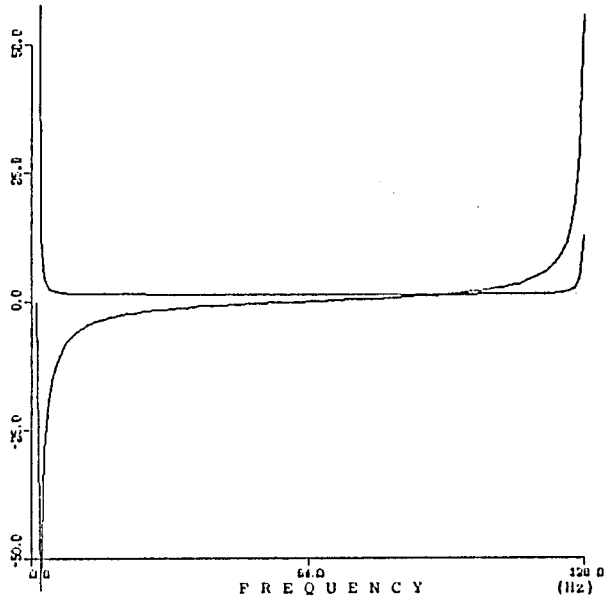

Fig. 13.c. Real and Imaginary parts of the FFT of Fig. 13.a.

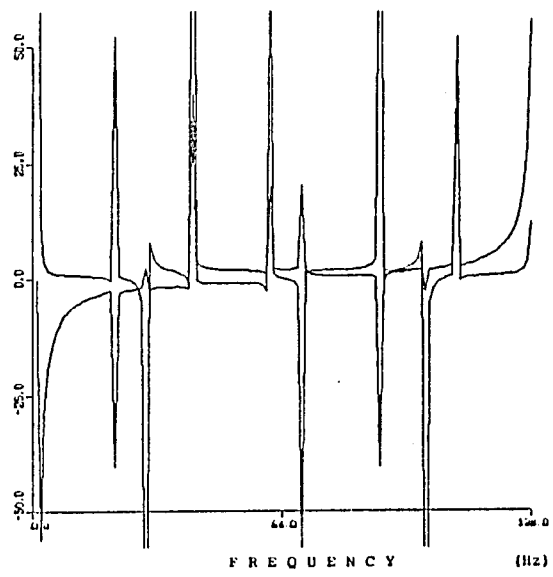

Fig. 13.d. Real and Imaginary parts of the FFT of Fig. 13.b.

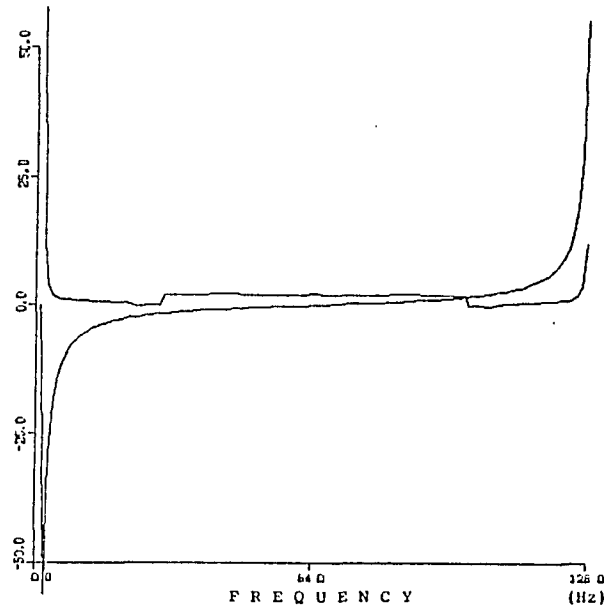

Fig. 13.e. The result of filtering Fig. 13.d. using a MF_{11}.

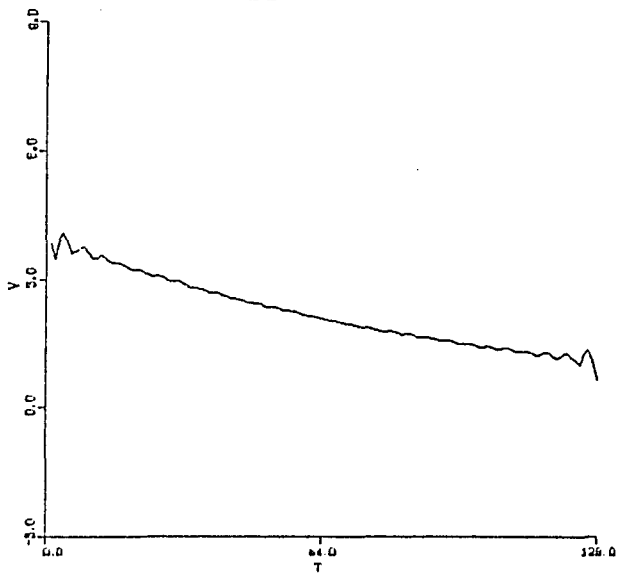

Fig. 13.f. Inverse FFT of Fig. 13.e. (recovered signal).

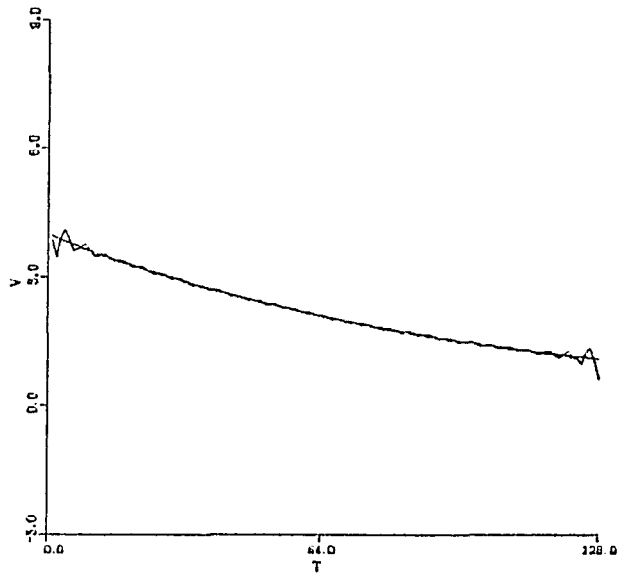

Fig. 13.g. Comparison of original and recovered signals.

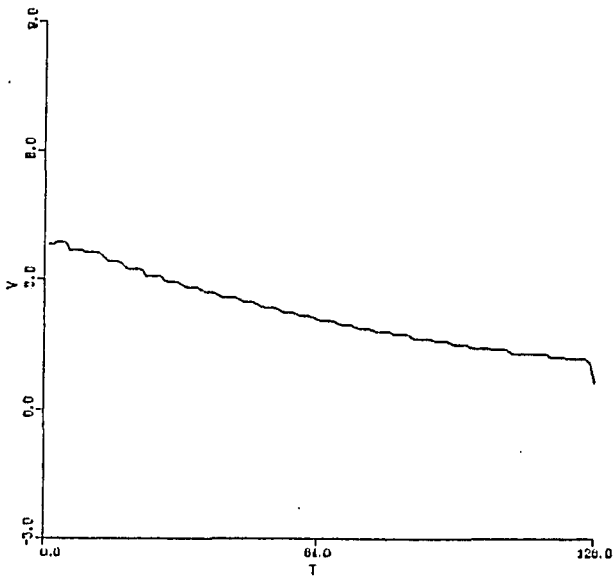

Fig. 13.h. Recovered exponential using a MF_7 in frequency domain and a MF_5 for post-processing.

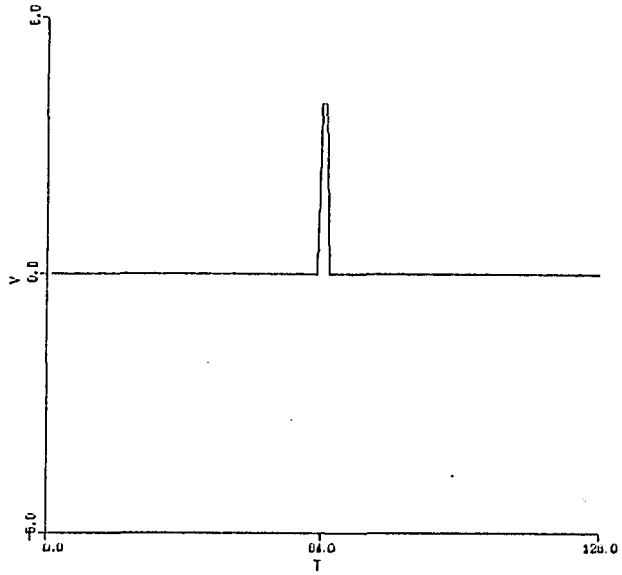

Fig. 14.a. Single pulse signal.

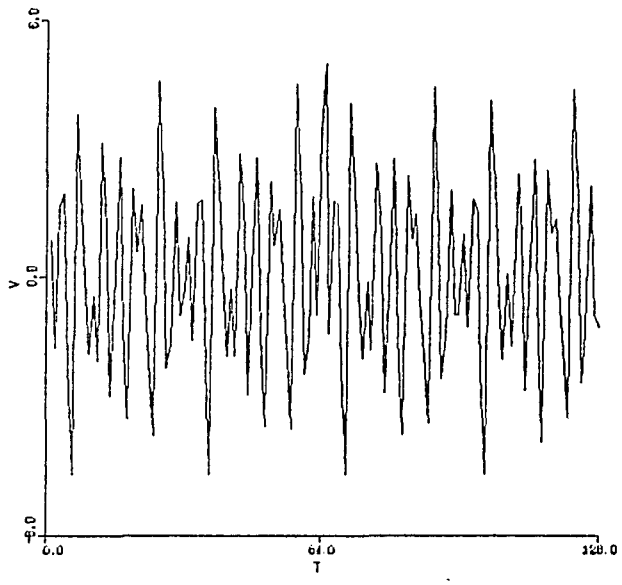

Fig. 14.b. Single pulse corrupted by sinousoids.

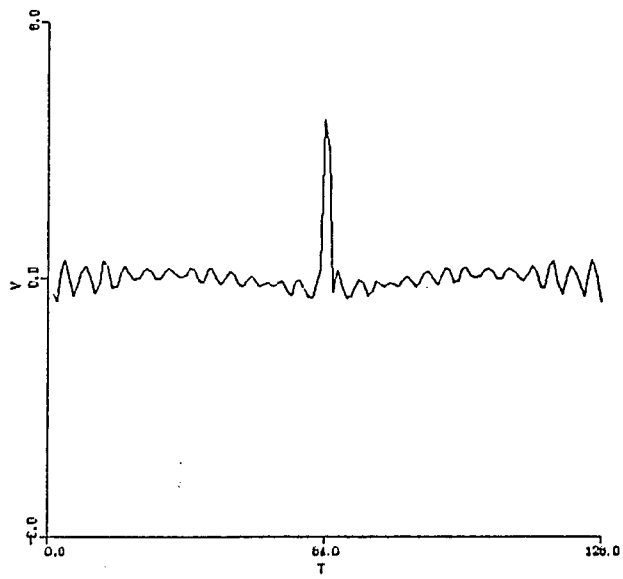

Fig. 14.c. Recovered pulse using a MF_5 in the frequency domain.

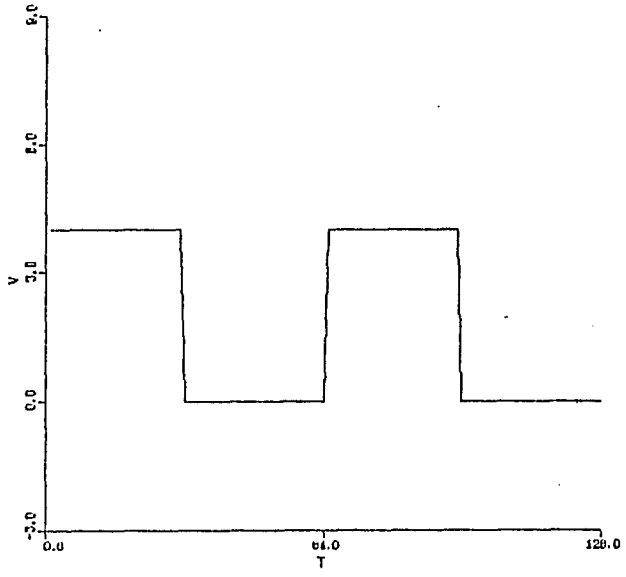

Fig. 15.a. Clean square-wave.

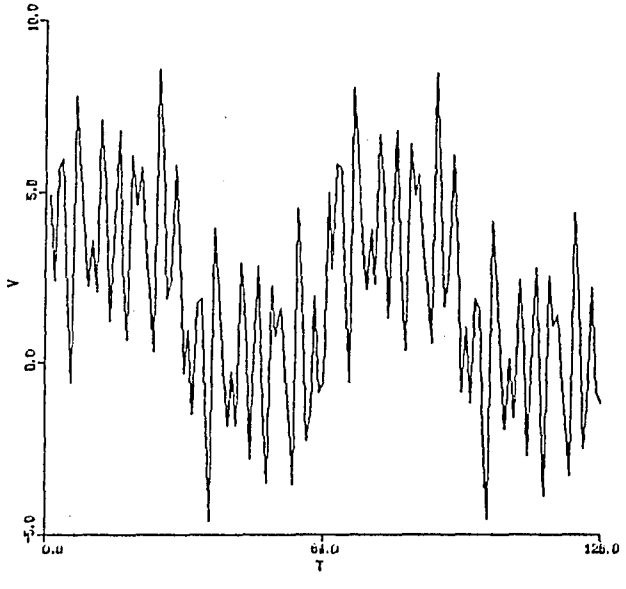

Fig. 15.b. Square-wave corrupted by sinousoids.

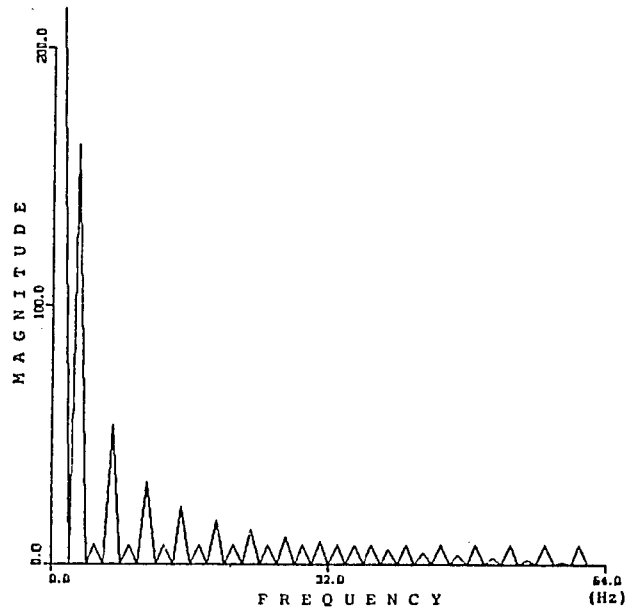

Fig. 15.c. Magnitude FFT of Fig. 15.a.

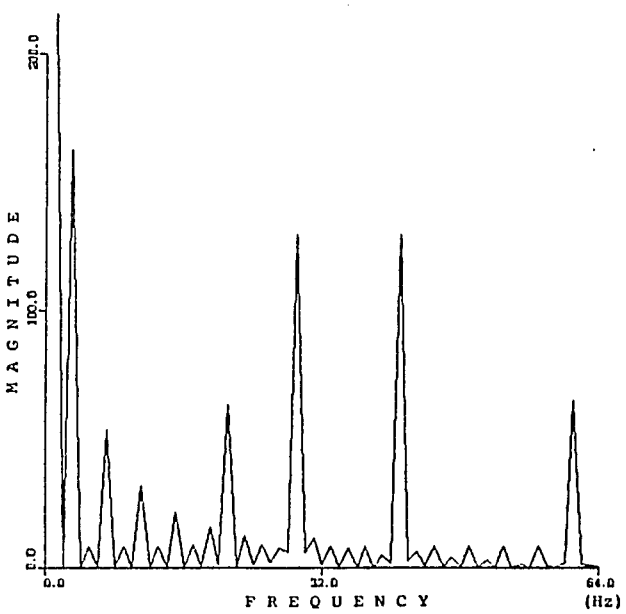

Fig. 15.d. Magnitude FFT of Fig. 15.b.

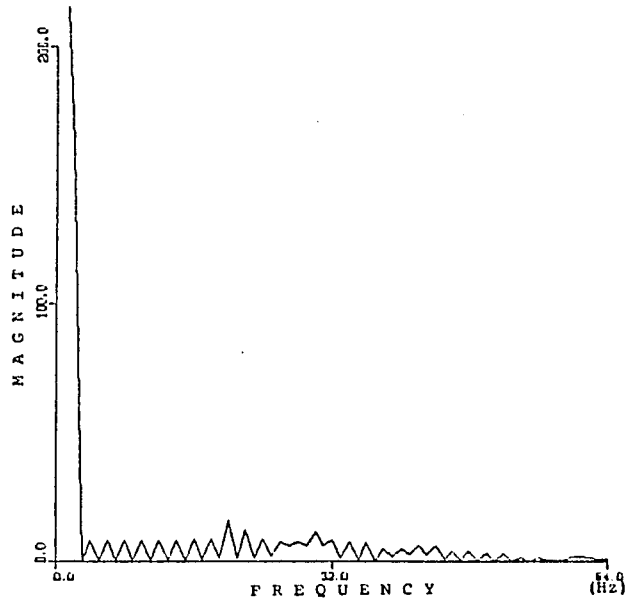

Fig. 15.e. Magnitude of the MF_3 filtered FFT of Fig. 15.d.

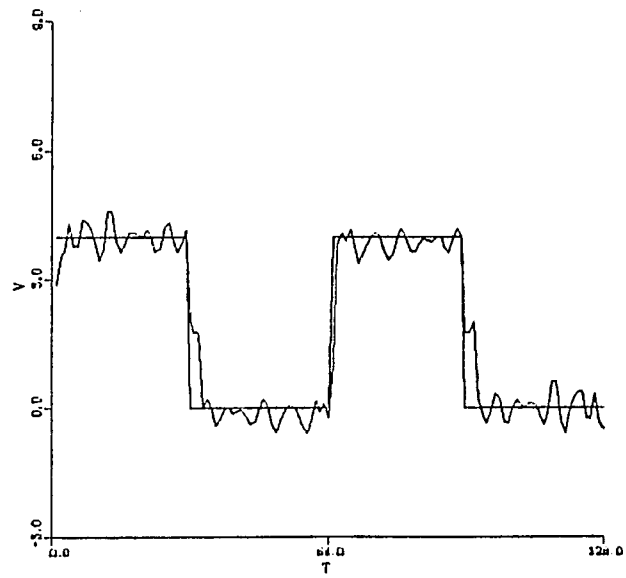

Fig. 15.f. Inverse FFT of Fig. 15.e. (recovered signal) vs. original signal.

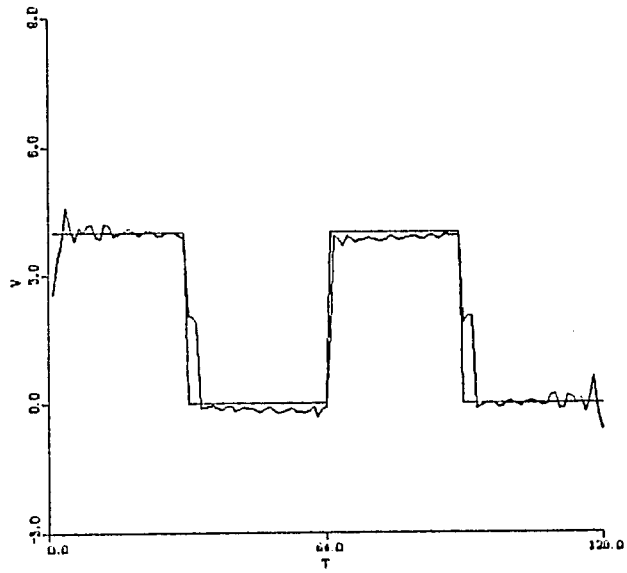

Fig. 15.g. Original and recovered signals using a MF_5 in the frequency domain.

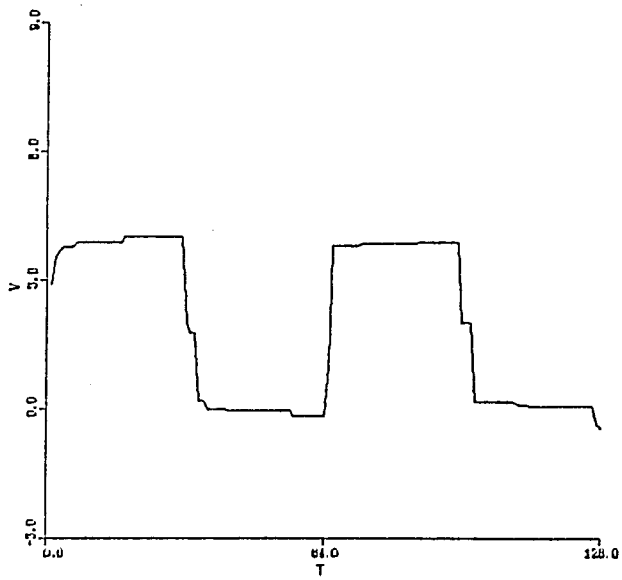

Fig. 15.h. The recovered signal of Fig. 15.f. with MF_{13} post-filtering.

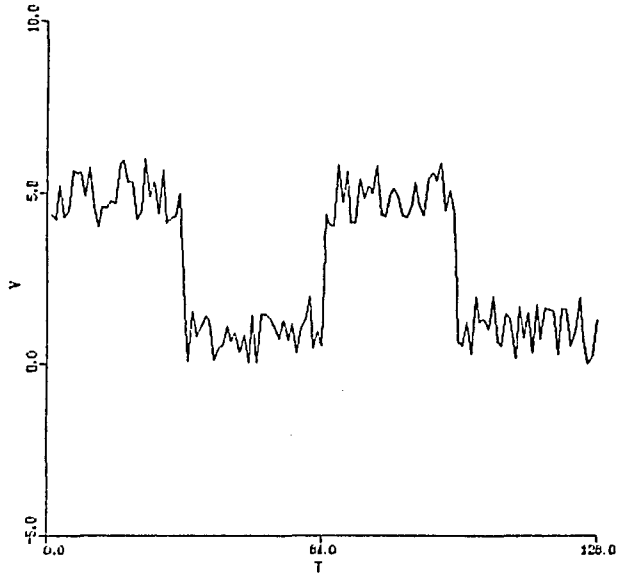

Fig. 16.a. Noisy square-wave.

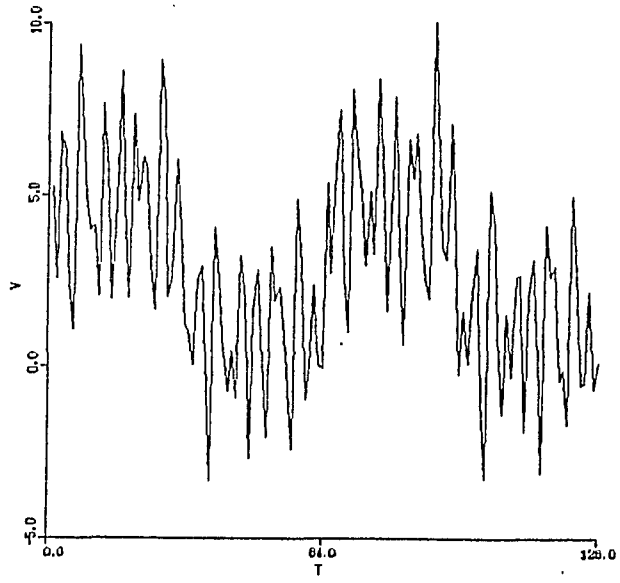

Fig. 16.b. Noisy square-wave corrupted by sinousoids.

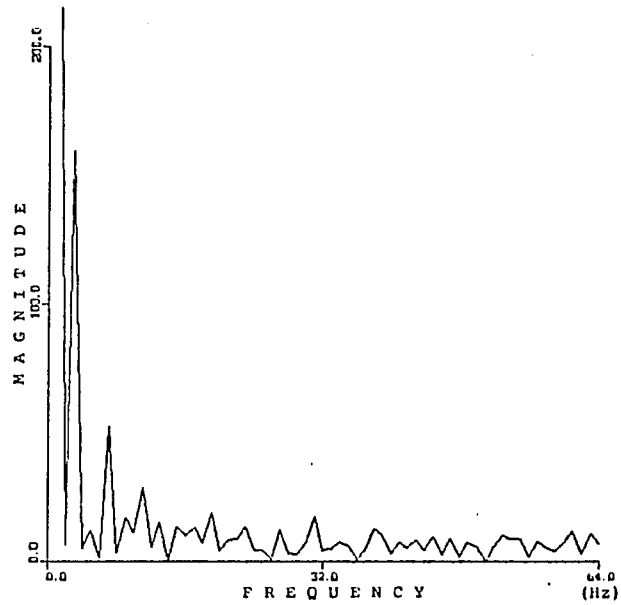

Fig. 16.c. Magnitude FFT of Fig. 16.a.

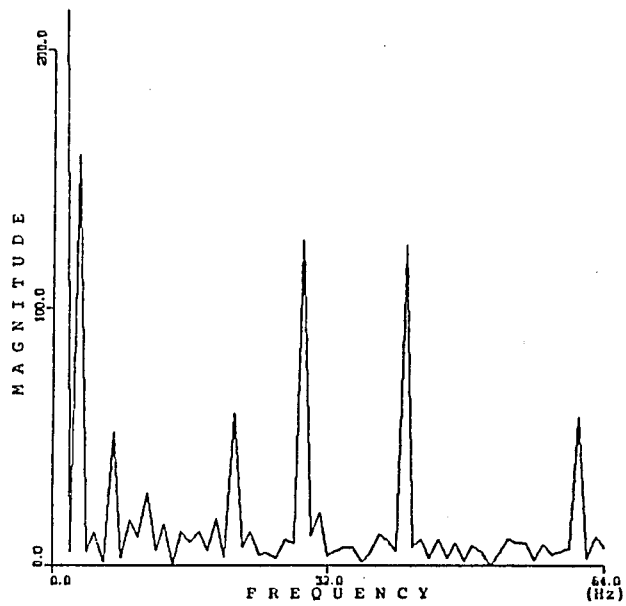

Fig. 16.d. Magnitude FFT of Fig. 16.b.

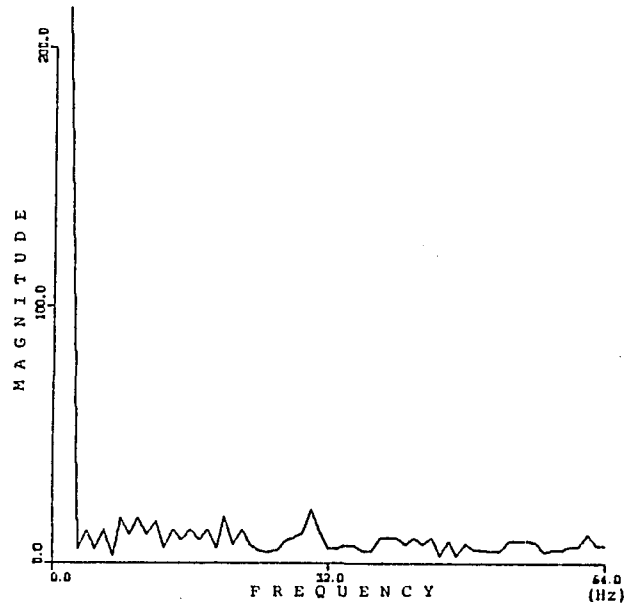

Fig. 16.e. Magnitude of MF_3 filtered FFT of Fig. 16.b.

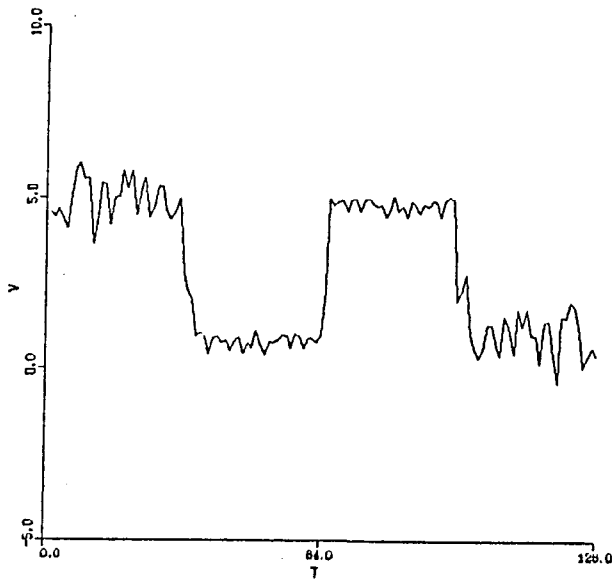

Fig. 16.f. Inverse FFT of Fig. 16.e.
(Recovered signal).

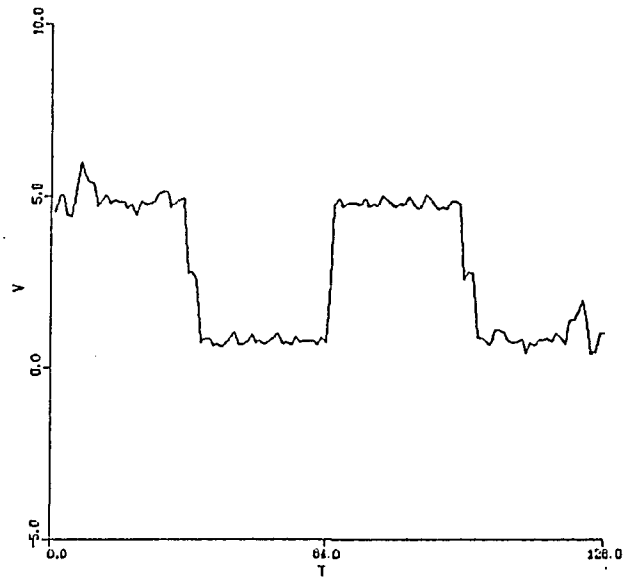

Fig. 16.g. MF_{11} recovered signal.

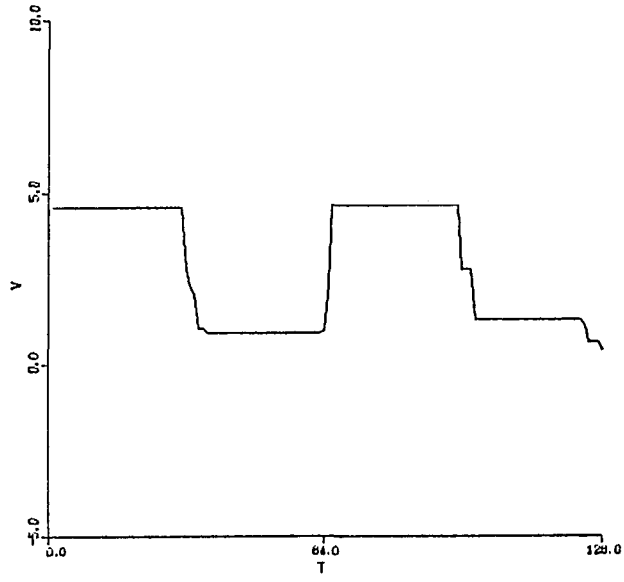

Fig. 16.h. The recovered signal of Fig. 16.f. with MF_{13} post-filtering.

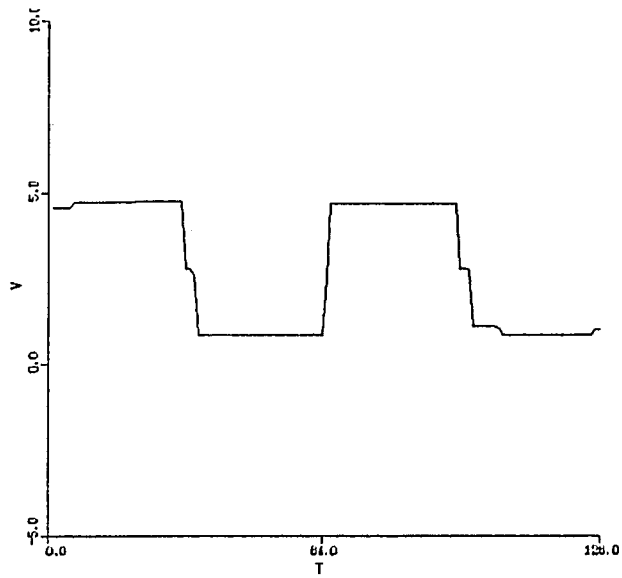

Fig. 16.i. The recovered signal of Fig. 16.g. with MF_{13} post-filtering.

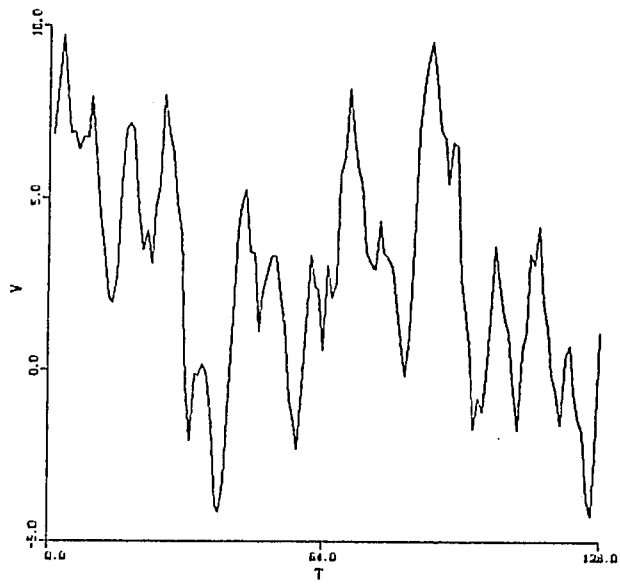

Fig. 17.a. Noisy square-wave.

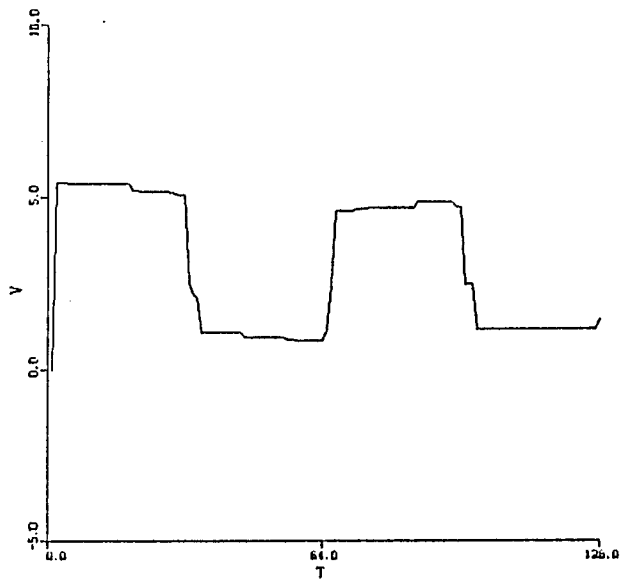

Fig. 17.b. MF_3 recovered square-wave with MF_{13} post-filtering.

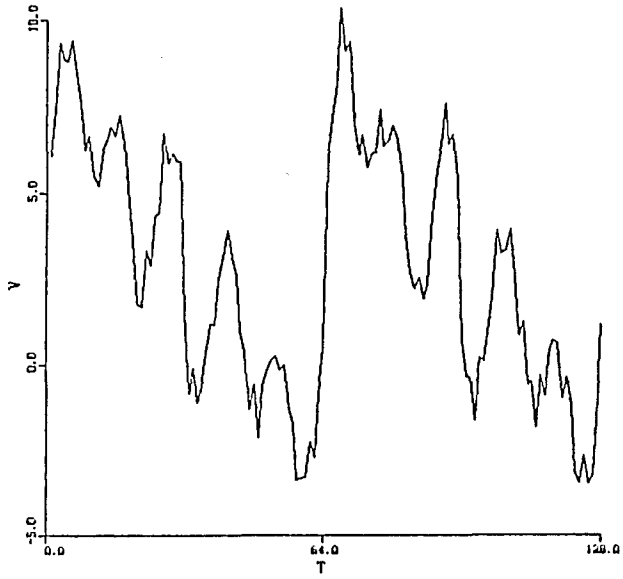

Fig. 18.a. Noisy square-wave.

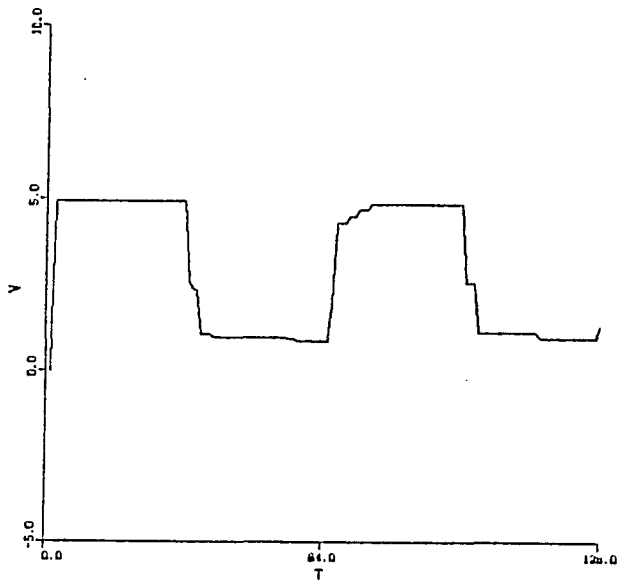

Fig. 18.b. MF_3 recovered signal with MF_{13} post-filtering.

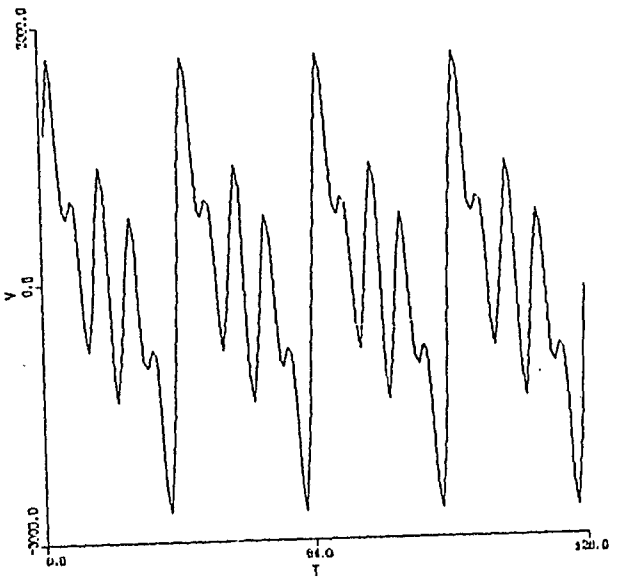

Fig. 19.a. High-power jammed square-wave.

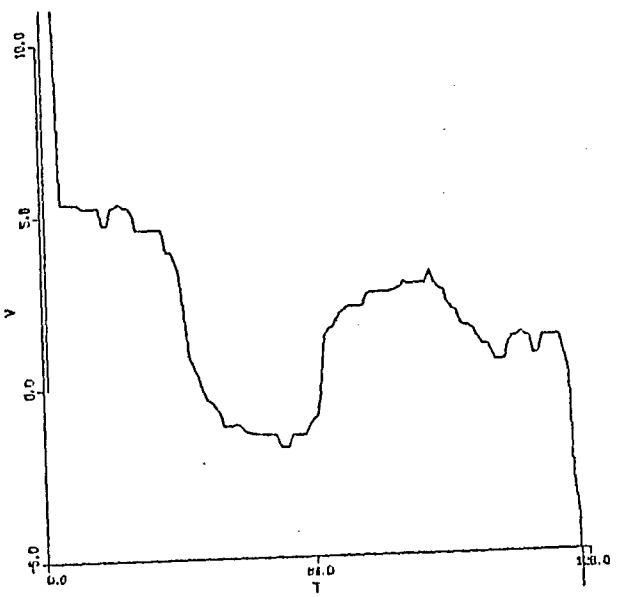

Fig. 19.b. MF_{11} recovered signal with MF_{15} post-filtering.

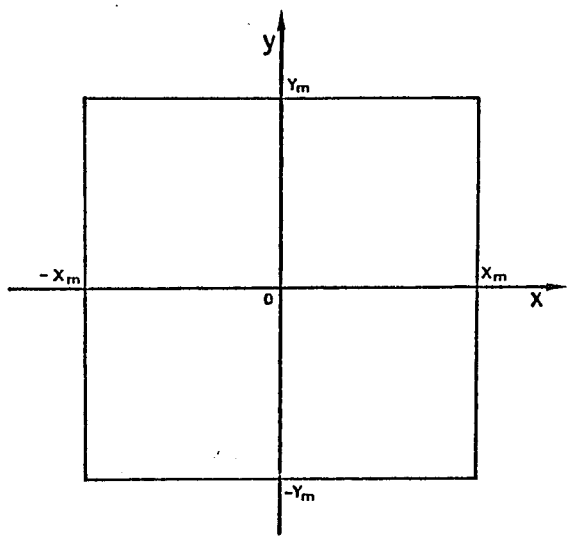

Fig. 20.a. X-Y Plane with reference axes.

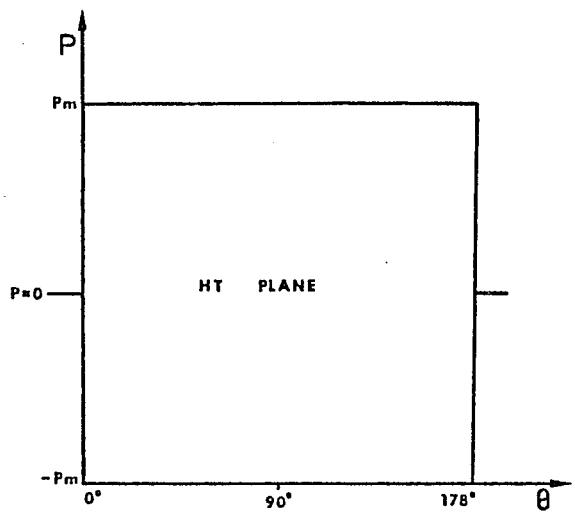

Fig. 20.b. HT Plane with reference axes.

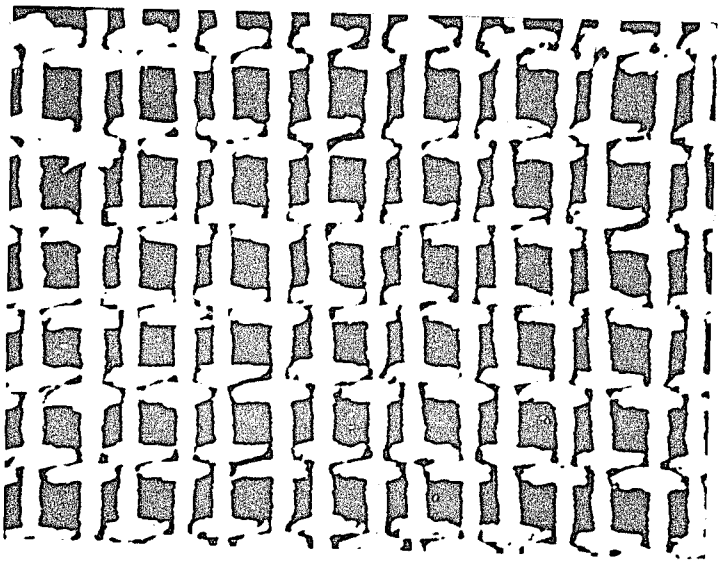

Fig. 21.a. Texture of French canvas.

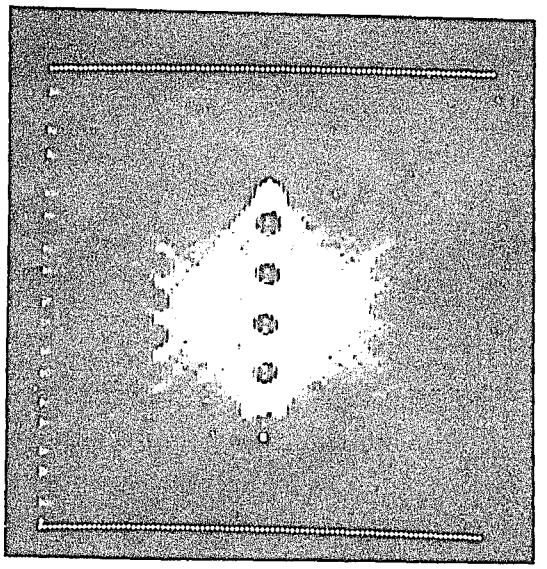

Fig. 21.b. HT of a block of Fig. 21.a.

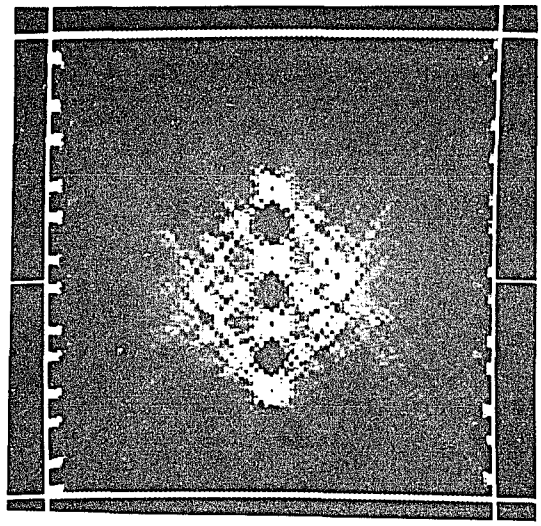

Fig. 21.c. HT of a block of Fig. 21.a. with preprocessing for line thinning.

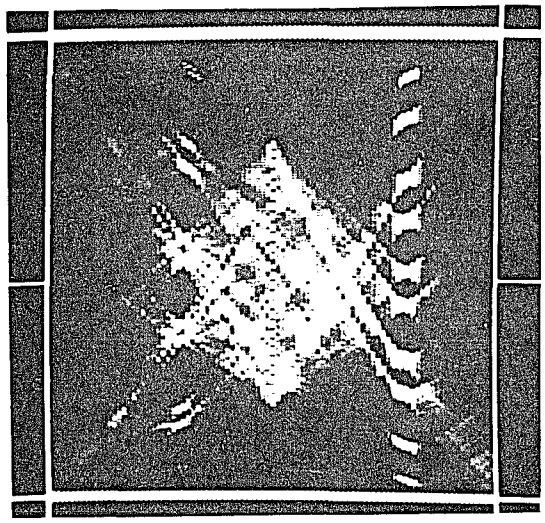

Fig. 21.d. HT of a rotated and scaled block of Fig. 21.a with preprocessing for line thinning.

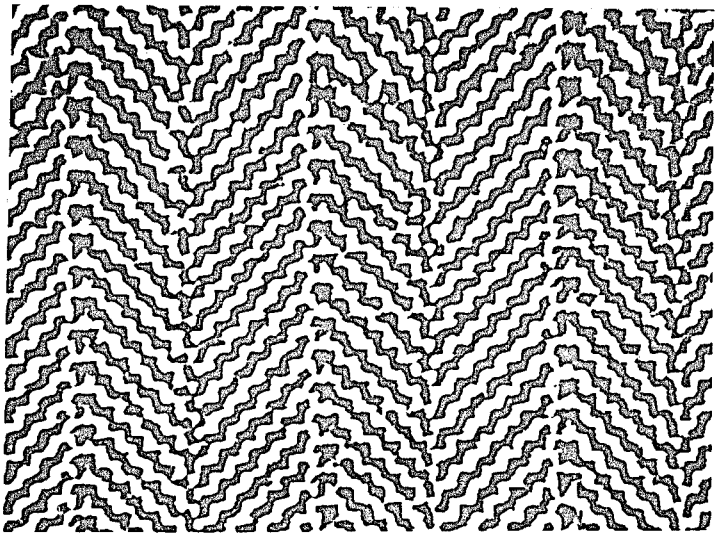

Fig. 22.a. Texture of Herringbone weave.

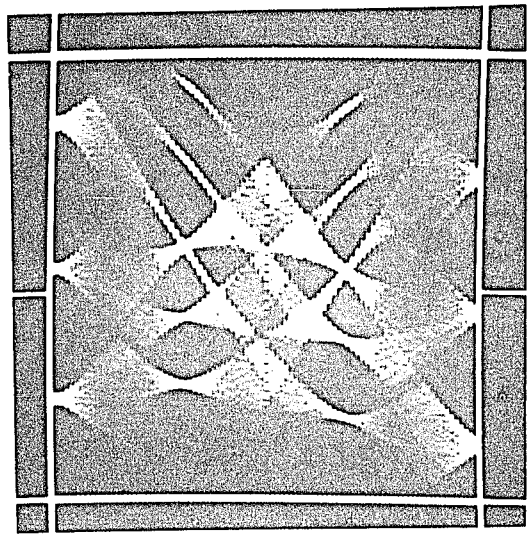

Fig. 22.b. HT of a preprocessed block of Fig. 22.a.

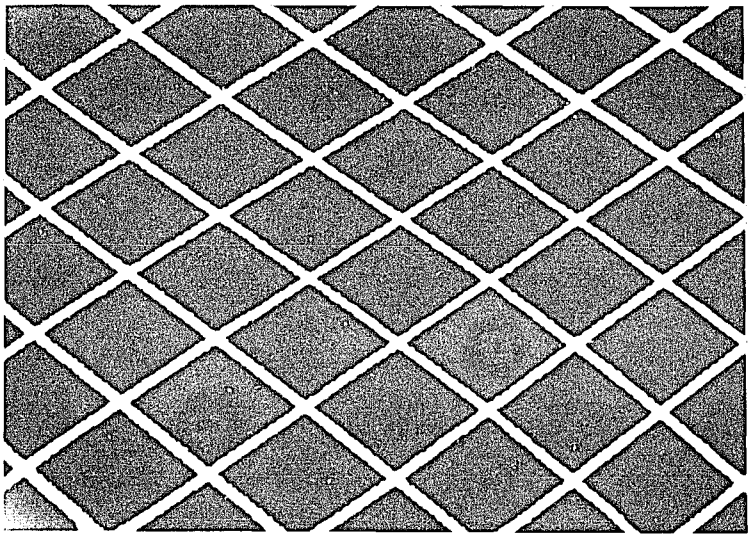

Fig. 23.a. Test texture pattern.

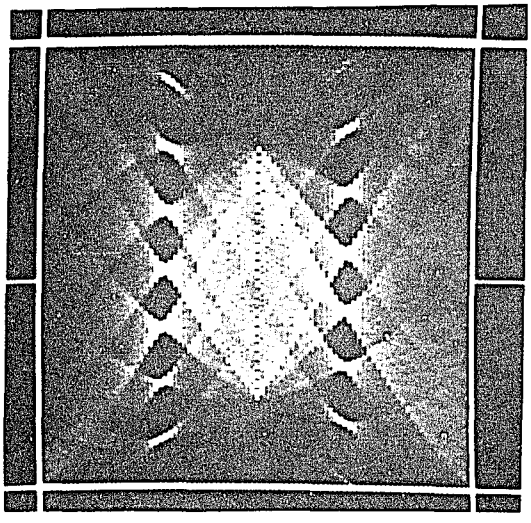

Fig. 23.b. HT of Fig. 23.a.

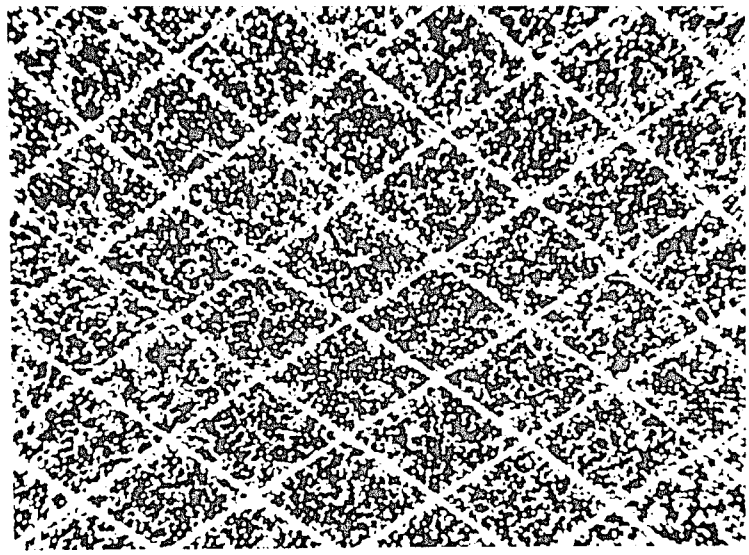

Fig. 23.c. Fig. 23.a. with uniform "salt and pepper" noise added.

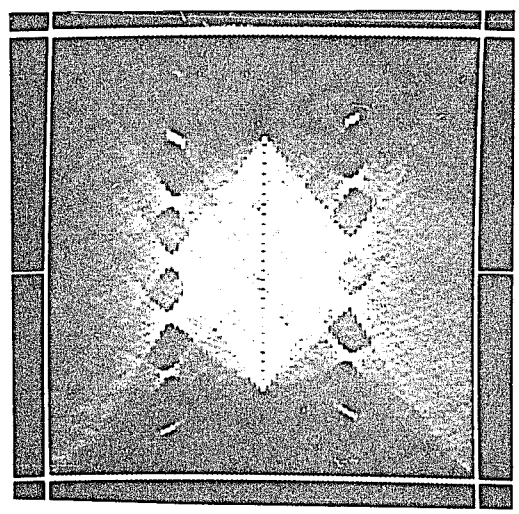

Fig. 23.d. HT of Fig. 23.c.

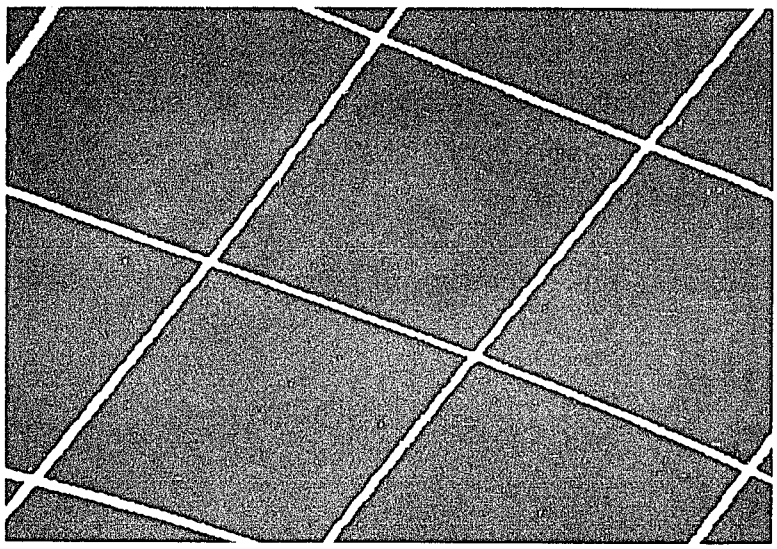

Fig. 24.a. Fig. 23.a under scaling and translation.

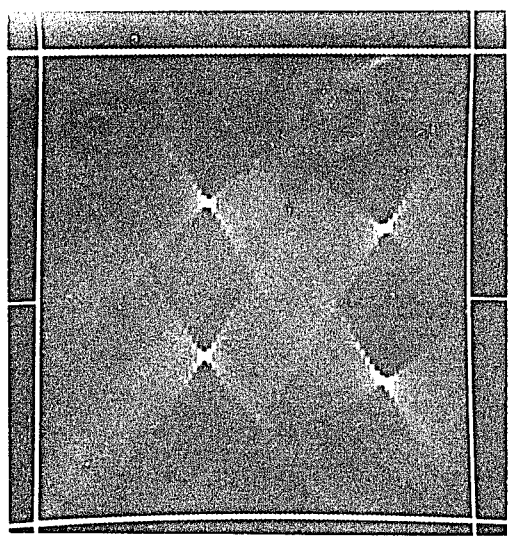

Fig. 24.b. HT of Fig. 24.a

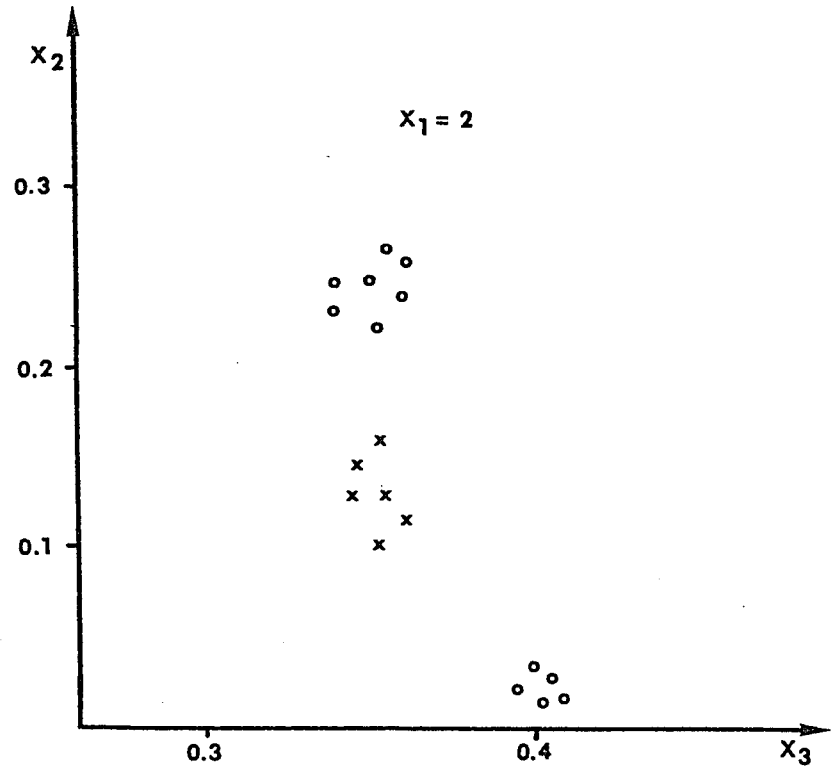

Fig. 25. Scattering diagram of three 2-directional test patterns.

θ	52	90	128	θ	72	110	148
d	71	90	71	d	49	66	55
	70	0	69		51	66	54
	0	0	72		51	0	53
					0	0	53
					0	0	57

Normalize distances

52	90	128	72	110	148
0.79	1	0.79	0.74	1	0.83
0.78	0	0.77	0.77	1	0.82
0	0	0.80	0.77	0	0.80
			0	0	0.80
			0	0	0.86

Average distances

52	90	128	72	110	148
0.79	1	0.78	0.76	1	0.82

Shift circularly

90	128	52	110	148	72
1	0.79	0.78	1	0.82	0.76

Normalize angles

0	38	142	0	38	142
1	0.79	0.78	1	0.82	0.76

Normalize angles to 180°

0	0.21	0.79	0	0.21	0.79
1	0.79	0.78	1	0.82	0.76

Calculate variances	Form feature vector
$x = \{3, 0.12, 0.41\}$	$x = \{3, 0.12, 0.41\}$

Table 2. Classification algorithm applied on two views of the same texture pattern.

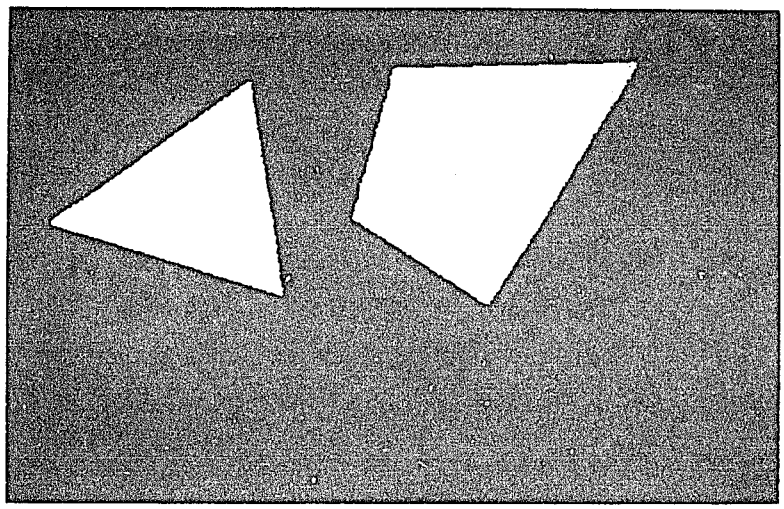

Fig. 26.a. A binary image.

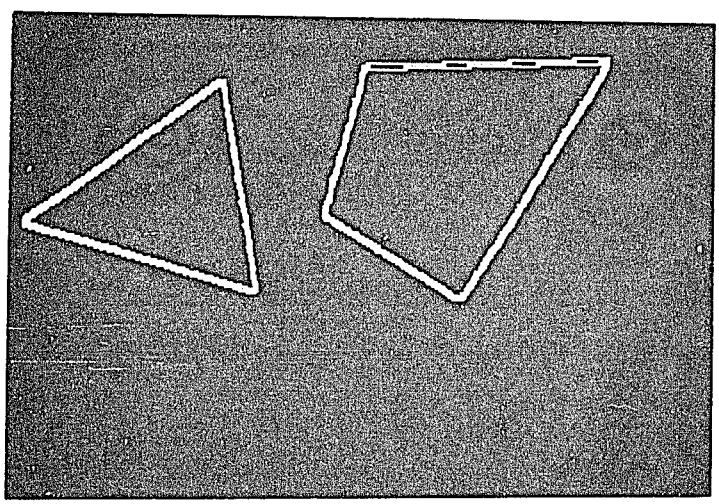

Fig. 26.b. Sobel edge detector applied on Fig. 26.a.

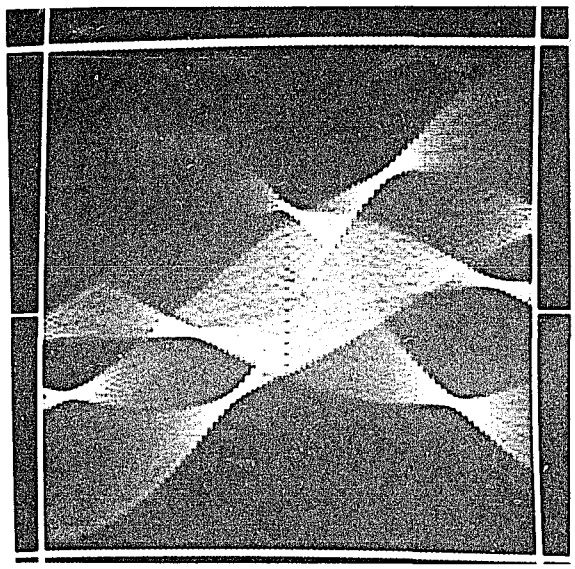

Fig. 26.c. HT of Fig. 26.b.

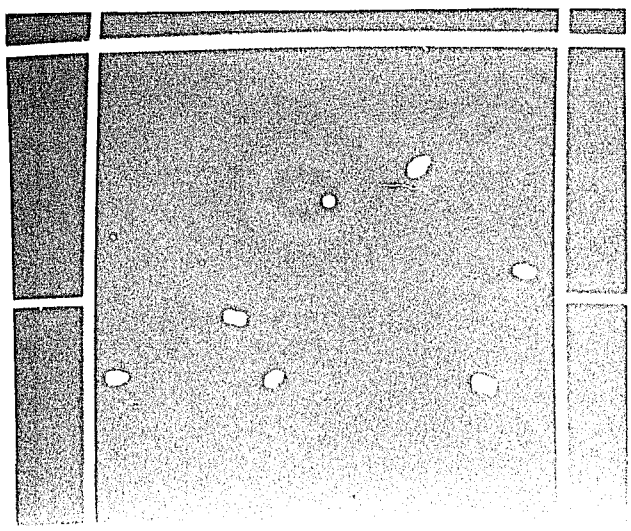

Fig. 26.d. Threshold HT of Fig. 26.c.

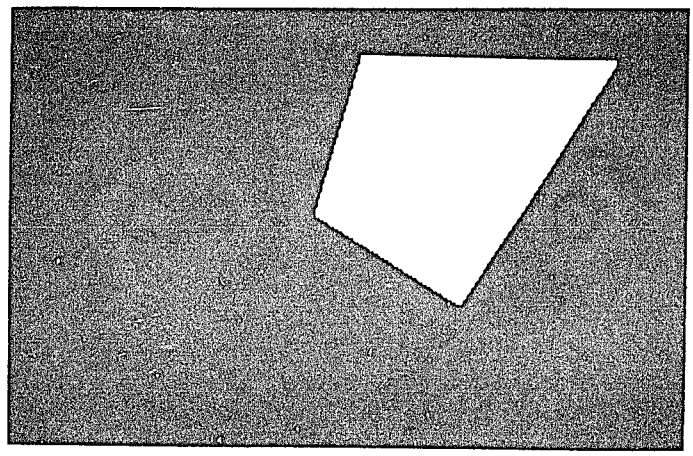

Fig. 26.e. Generated mask for image segmentation.

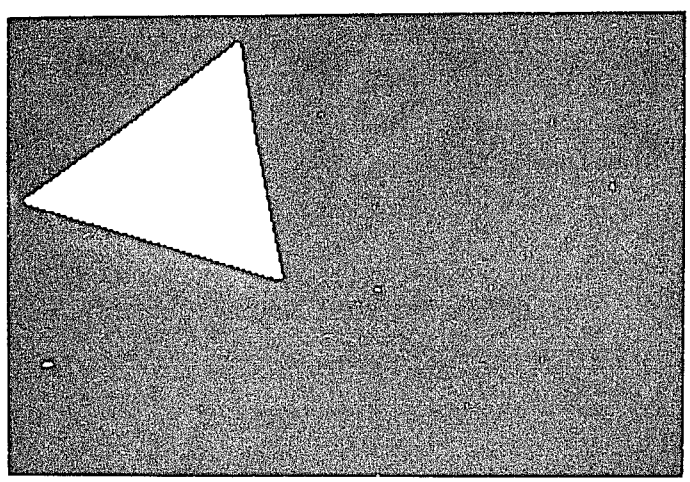

Fig. 26.f. Generated mask for image segmentation.

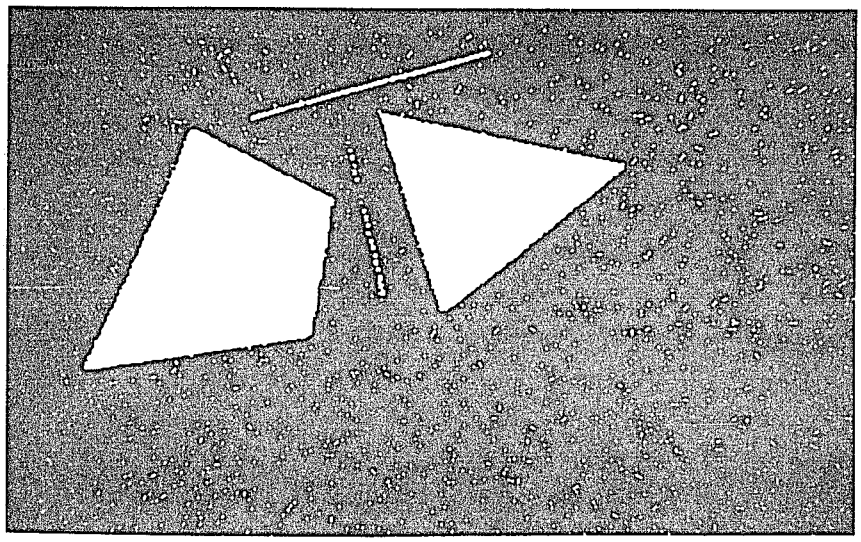

Fig. 27.a. Noisy binary image.

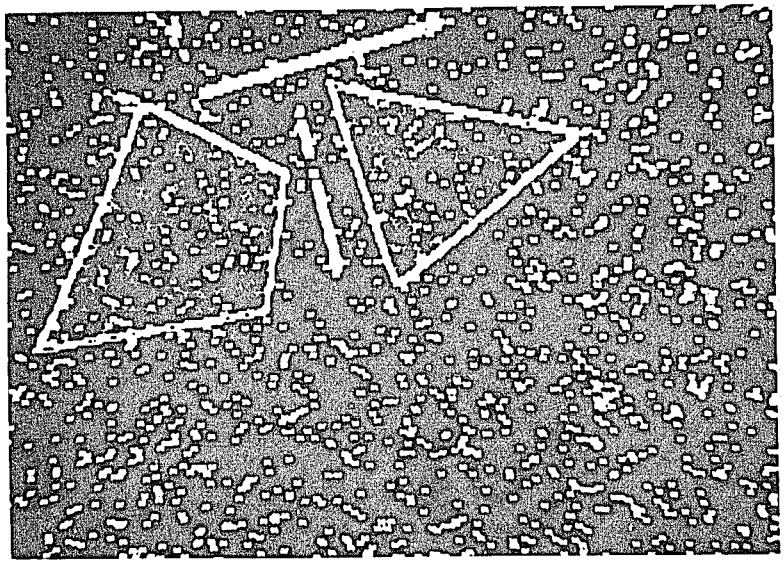

Fig. 27.b. Sobel operator applied on Fig. 27.a.

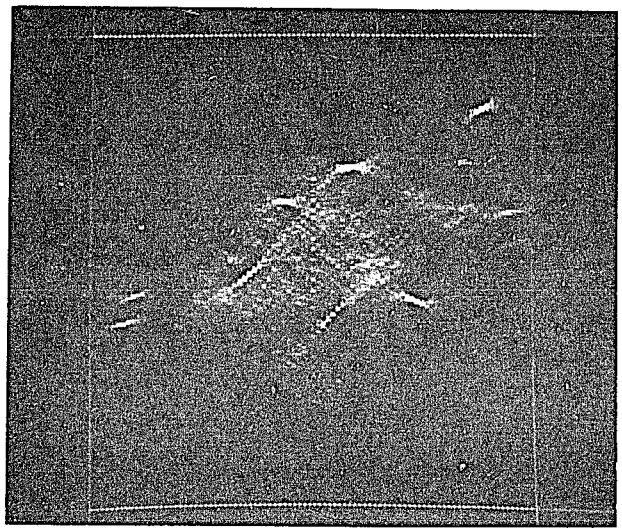

Fig. 27.c. HT of Fig. 27.b.

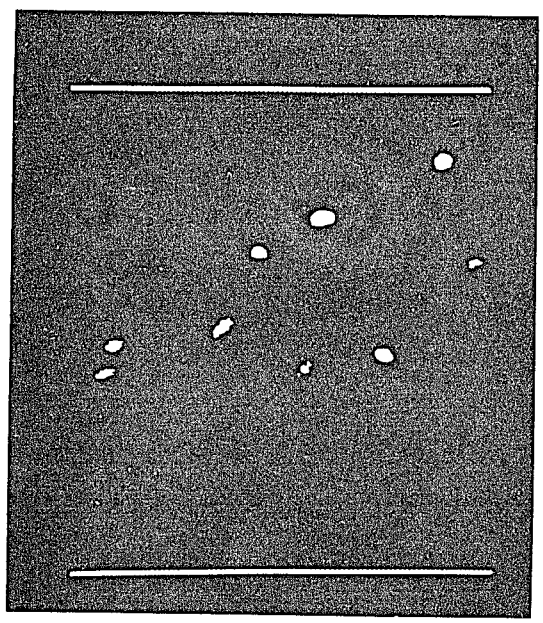

Fig. 27.d. Threshold of Fig. 27.c.

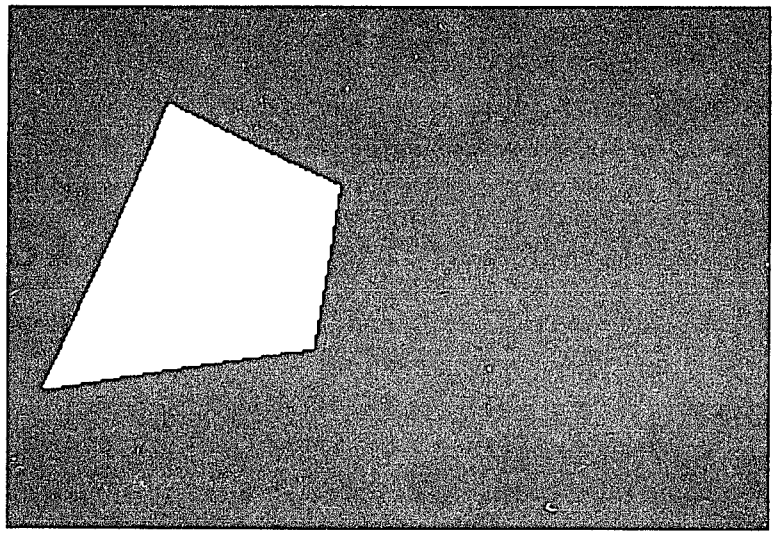

Fig. 27.e. Generated mask for image segmentation.

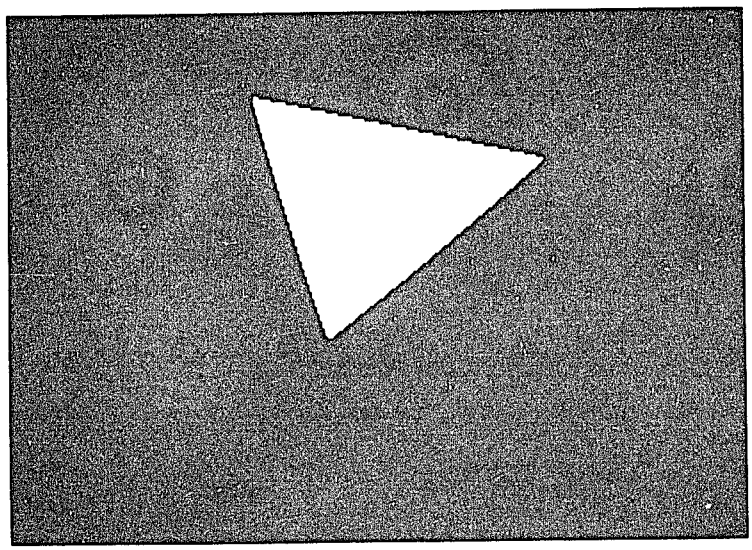

Fig. 27.f. Generated mask for image segmentation.

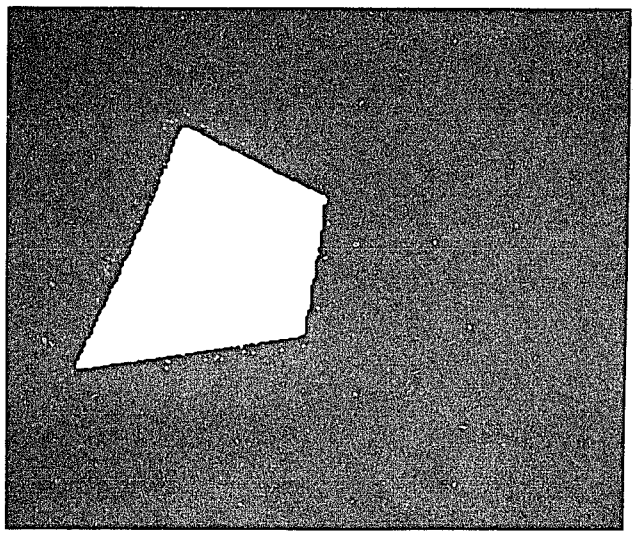

Fig. 27.g. Segmented image.

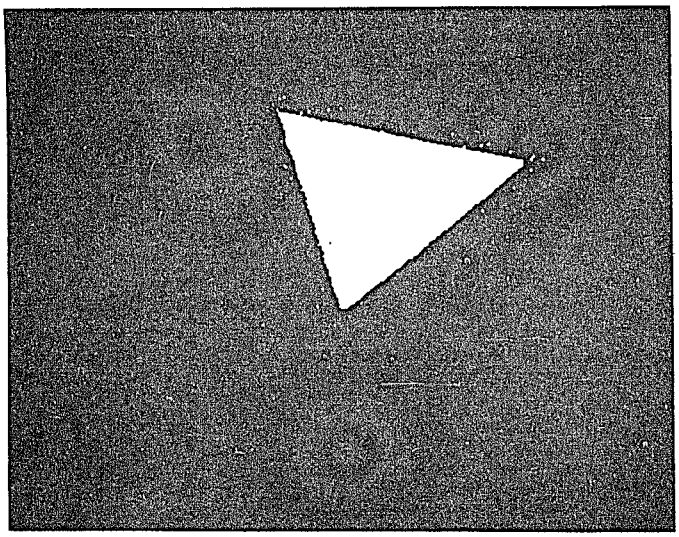

Fig. 27.h. Segmented image.

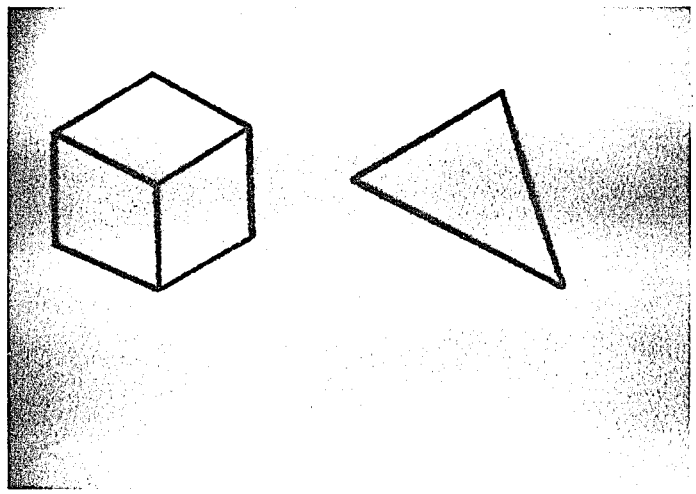

Fig. 28.a. Original image.

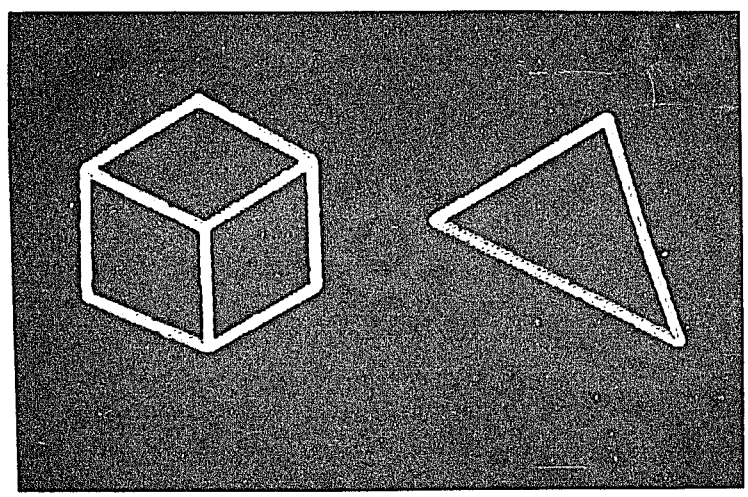

Fig. 28.b. Sobel operator of Fig. 28.a.

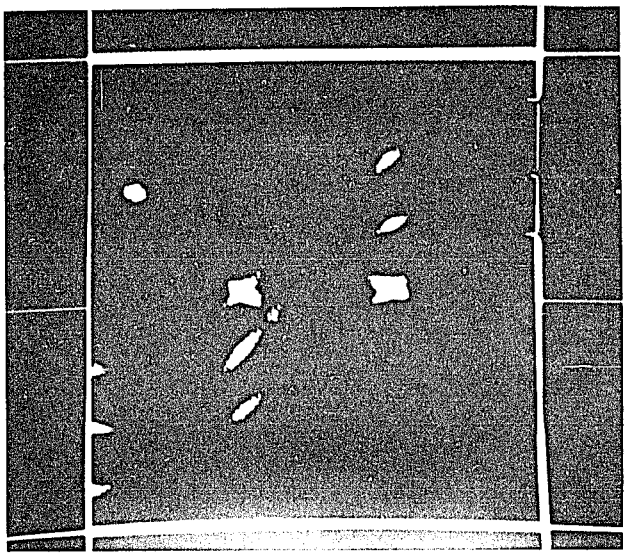

Fig. 28.c. Thresholded HT of Fig. 28.b.

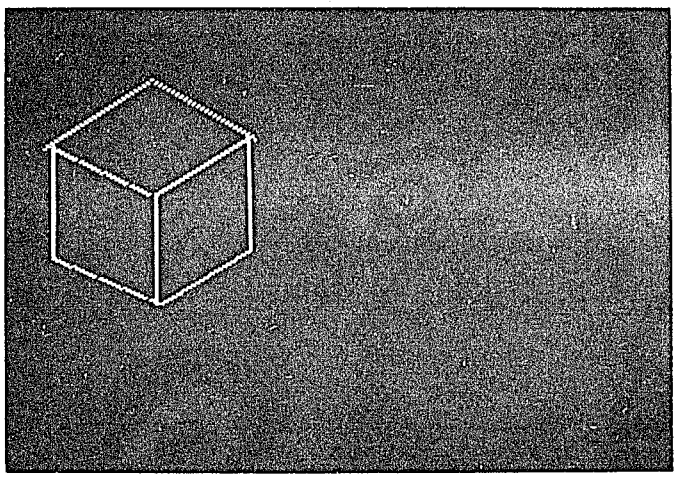

Fig. 28.d. Reconstructed cube.

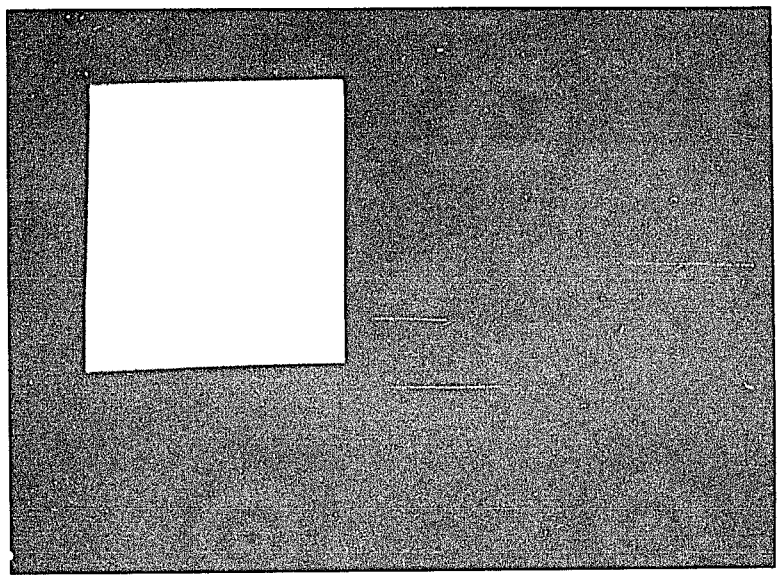

Fig. 28.e. Generated mask suitable for image segmentation.

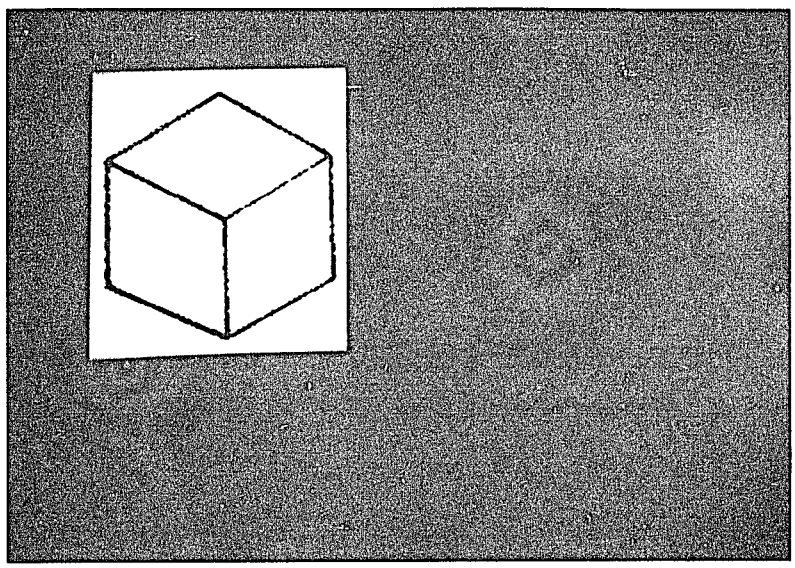

Fig. 28.f. Segmented original image.

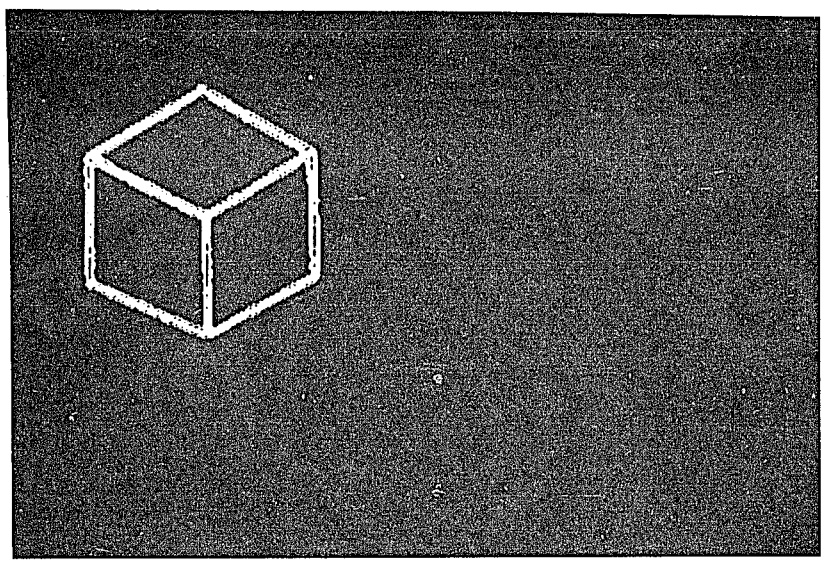

Fig. 28.g. Segmented edge image.

A P P E N D I C E S

APPENDIX A

The EPD is given as:

$$f_x(x) = Ae^{-x'} \qquad 0 \le x < \infty \qquad (A.1)$$

with $A = \frac{1}{p2}$ and $x' = \frac{x}{p2}$

where p is a distribution parameter. The mean value of an EPD is $\bar{x} = E[x] = p^2$, while its variance is $\bar{x}^2 = \sigma_x^2 = p^4$. If we define the signal to noise ratio (SNR) of a random variable x as:

$$SNR_x = \frac{E[x]}{\sigma x} \qquad (A.2)$$

then for the EPD the SNR is unity. We now assume that the random variables of Eq. (A.1) are passed once through a VMF_{NxM}.

Substituting Eq. (A.1) into Eq. (4-1) yields:

$$f_y(y_i) = \frac{N!}{P!\,Q!} A^2 \left[1-e^{-y_i'}\right]^P \cdot \left[e^{-y_i'}\right]^{Q+1} \qquad (A.3)$$

with $y_i' = \frac{y_i}{p^2}$, $0 \le y_i < \infty$, and $i=1, 2 \ldots M$

For the case when M=i=1, Eq. (A.3) reduces to the MF output distribution.

The mean value of y_i is given as

$$E_i = E[y_i] = \int_{-\infty}^{\infty} y_i f_{y_i}(y_i) dy_i \qquad (A.4)$$

substituting Eq. (A.3) into Eq. (A.4) we find, either by direct integration or by using integration tables, that:

$$E_i = p^2 \sum_{k=0}^{P} \left(\frac{1}{N-k}\right) \qquad i=1,2...M \qquad (A.5)$$

where, again, $P = \frac{1}{2}(N-M)+i-1$. When $M=i=1$ (i.e. the MF), then:

$$E_1 = E[y] = p^2 \sum_{k=0}^{n} \left(\frac{1}{N-k}\right) \qquad \text{where } n = \tfrac{1}{2}(N-1) \qquad (A.6)$$

This matches the result of Ref. [20]. We can rewrite Eq. (A.5) as

$$E_i = p^2 \sum_{k=0}^{n+m} \left(\frac{1}{N-k}\right) \qquad \text{where } m=i-\tfrac{1}{2}(M+1) \qquad (A.7)$$

Comparing Eqs. (A.6) and (A.7) we can see that, depending on the values of M and i, the summation in (A.7) can be extended to a greater, equal, or fewer number of terms compared to the sum in (A.6). Thus, the mean values of the elements in the output data vector can be greater, equal, or less than the

mean value of the true median. Equality occurs when the median is included in the output vector.

Since the elements in the output vector do not have the same mean, we define a vector mean, i.e. a vector of means. Furthermore, we define a vector magnitude mean E as the magnitude of the vector mean, i.e.

$$E = \left[\sum_{i=1}^{M} E_i^2 \right]^{\frac{1}{2}} \tag{A.8}$$

The mean value E_i can be approximated, using the Euler-Maclaurin summation formula,

$$E_i = E[y_i] = p^2 \left\{ \ln \left[\frac{2(N+1)}{N+M-2i+2} \right] + \frac{1}{2} \left[\frac{N-M+2i}{(N+M-2i+2)(N+1)} \right] \right\} \tag{A.9}$$

Asymptotically, when $N \to \infty$, then $E_i = p^2 \ln 2$, that is, all output elements have approximately the same mean as the mean of the MF [see Eq. (A.6)].

A similar calculation for the variance S_i yields:

$$S_i = \sigma_{y_i}^2 = p^4 \sum_{k=0}^{P} \frac{1}{(N-k)^2} \qquad i=1,2...M \tag{A.10}$$

Since the summation in Eq. (A.10) is the same form as in Eq.

(A.5), similar comments to those made for the mean apply. Thus, we can define a vector variance and a magnitude variance S given by:

$$S = \sum_{i=1}^{M} S_i \tag{A.11}$$

Eq. (A.10) can be approximated by

$$S = \sigma_{y_i}^2 \cong p^4 \left\{ \frac{N-M+2i+2}{(N+M-2i)(N+1)} - \frac{1}{2(N+1)^2} - \frac{2}{(N+M-2i)} \right\} \tag{A.12}$$

Asymptotically when $N \to \infty$, $S_i \cong p^4 \frac{1}{N}$ for all elements, which is also the variance of the true median.

From Eqs. (A.5) and (A.10), the SNR of the VMF_{NxM} can be calculated. As a figure of merit, Fig. A.1 compares the magnitude SNR of the VMF with that of the MF. As can be seen from Fig. A.1, by using the VMF, a slightly larger SNR can be achieved. Asymptotically, for larger window sizes, when $N \to \infty$, the SNR is

$$\underset{N\to\infty}{SNR} = p^2 \frac{1n2}{N} \tag{A.13}$$

The SNR of Eq. (A.13) represents both a large window MF, and also a large data window and small output window size VMF.

In this sense, asymptotically the MF and the VMF are statistically similar in performance.

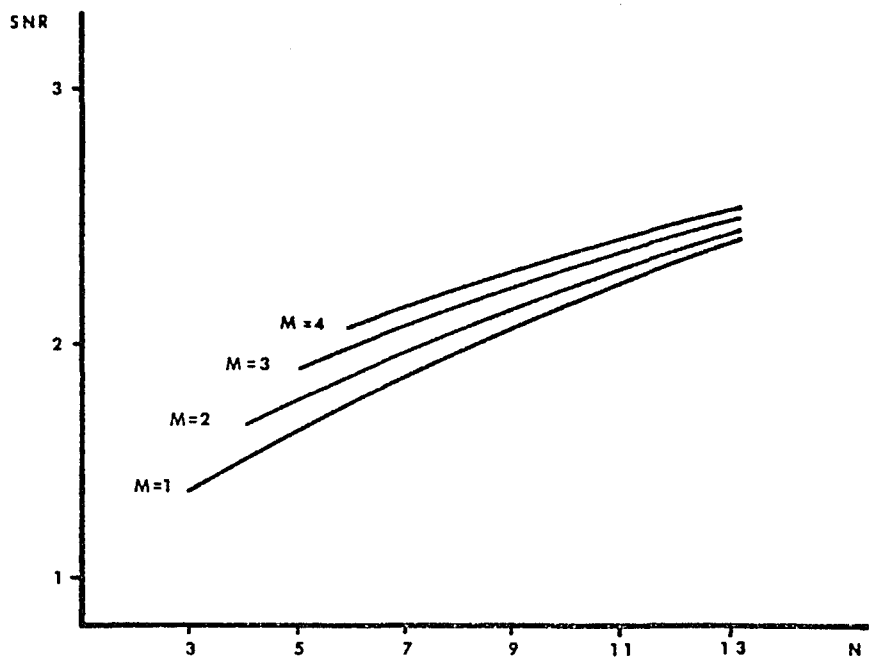

Fig. A.1 SNR of VMF and MF.

APPENDIX B

The associative memory mapping of an input vector x_k into an output vector y_k can be described by the transfer relation:

$$y_k = Mx_k \qquad k=1,....P \qquad (B1)$$

where M is the unknown associative memory and P is the total number of vector pairs. An iterative method for the determination of the matrix M will now be presented.

Let $\{x_k\}$, $\{y_k\}$, k=1,...P, be two sets of vectors. Forming the two rectangular matrices, X and Y, with the vectors x_k and y_k as their columns, as

$$X = [x_1, x_2,...,x_P] \ , \ Y = [y_1, y_2,..., y_P], \qquad (B2)$$

every pair of vectors from sets $\{x_k\}$, $\{y_k\}$, can be related by the matrix equation

$$Y = MX \qquad (B3)$$

Using the so called "Moore-Penrose" method [75], the least squares optimal solution of Eq. (B3) is found as

$$M = Y\ X^{+} \tag{B4}$$

where X^{+} is the pseudo-inverse of matrix X. The iterative solution of Eq. (B4) has the form of the difference equation [73, 75]

$$M_k = M_{k-1} + (y_k - M_{k-1}\ x_k)\ [C_k]^t \tag{B5}$$

where M_k and M_{k-1} are the new and previous optimal matrices, x_k and y_k are the new observation vectors, and $[C_k]^t$ is a gain vector that defines the correction. The iterations may be initiated with $M_0=0$ or $M_0=I$, where I is the identity matrix. The expression for the gain factor is:

$$[C_k]^t = [h_k]^t \ / \ \|h_k\|^2 \quad \text{for } h_k \neq 0 \tag{B6}$$

where h_k is the orthogonal projection of x_k on the subspace spanned by $h_1, \ldots h_{k-1}$, and the index denotes the transpose.

The orthogonal projection of the vector x_k to the set of previous columns of the matrix H = h_1, h_2, ... , can be obtained using the Gram-Schmidt orthogonalization [76]:

(B7)

$$h_k = x_k - \sum_{i=1}^{k-1} (x_k, h_i).h_i \,/\, || h_i ||^2 , \quad || h_i || \neq 0$$

where (x_k, h_i) is the vector inner product, and $|| h_i ||$ is the Euclidean norm of the vector h_i.

REFERENCES

[1] Y. Nakagawa and A. Rosenfeld, "A note on the use of local min and max operators in digital picture processing," IEEE Trans. Syst. Man and Cybern., vol. SMC-8, pp. 632-635, 1978

[2] B.I Justusson, "Median Filtering: Statistical Properties," Two-dimensional Digital Transforms and Filters, T.S. Huang, ed., Springer-Verlag, Berlin, Germany, 1981

[3] L.R. Rabiner, M.R. Sambur and C.E. Schmidt, "Applications of a nonlinear smoothing algorithm to speech processing", IEEE Trans. Acoust. Speech and Signal Proc., vol. ASSP-23, pp. 552-557, 1975

[4] P.F. Velleman, "Robust Non-linear Data Smoothing," Tech. Rept. 89, Ser.2, Dept. of Statistics, Princeton U., Princeton, N.J. 1975

[5] T.S. Huang, G.J. Yang and G.Y. Tang, "A fast two-dimensional median filtering algorithm," IEEE Trans. Acoust. Speech and Signal Processing, vol. ASSP-27, pp. 13-18, 1979

[6] W.K. Pratt, Digital Image Processing, Wiley, New York, N.Y. 1978

[7] P. Narenda, "A separable median filter for image noise smoothing," IEEE Trans. Pattern Anal. Machine Intel., vol. PAMI-3, No. 1, pp. 20-29, 1981

[8] J.W. Tukey, Exploratory Data Analysis (preliminary ed.), Addison-Wesley, Reading, Mass. 1971

[9] G. Arce and N. Gallagher, "BTC Image Coding Using Median Filter Roots," IEEE Trans. on Comm., Vol. COM-31, pp. 784-793, 1983

[10] T. Nodes and N. Gallagher, "Median Filters: Some Modifications and Their Properties," IEEE Trans. Acoust. Speech and Signal Processing, vol. ASSP-30, pp. 739-746, 1982

[11] A. Bovik, T. Huang and D. Munson, "A Generalization of Median Filtering Using Linear Combinations of Order Statistic,," IEEE Trans. Acoust. Speech and Signal Processing, vol. ASSP-31, pp. 1342-1250, 1983

[12] L. Ling, R. Yin and X. Wang, "Nonlinear Filters for Reducing Spiky Noises in Two-Dimensions," ICASSP 1984, pp. 31.8.1-31.8.4

[13] J.P. Fitch, E.J. Coyle, and N.C. Gallagher," The Analog Median Filter," IEEE Trans. Circuits Syst., Vol. CAS-33, pp. 94-102, 1986

[14] K. Oflazer, "Design and Implementation of a Single-Chip 1-D Median Filter," IEEE Trans. Acoust. Speech and Signal Processing, vol. ASSP-31, pp. 1164-1168, 1983

[15] J.A. Roskind, "A Fast Sort-Selection Filter Chip with Effectively Linear Hardware Complexity," Proc. ICASSP (Tampa), 1985, pp. 1519-1522

[16] R.G. Harber and S.C. Bass, "VLSI implementation of a fast rank order algorithm," Proc. ICASSPC (Tampa), 1985

[17] N. Gallagher and G. Wise, "A theoretical analysis of the properties of median filters," IEEE Trans. Acoust. Speech and Signal Proc., vol. ASSP-29, pp. 1136-1141, 1981

[18] E. Ataman and K. Wong, "Some Statistical Properties of Median Filters," IEEE Trans. Acoust. Speech and Signal Processing, vol. ASSP-29, pp. 1073-1075, 1981

[19] H.A. David, Order Statistics, Wiley, N.Y. 1970

[20] R. Frieden "Some Statistical Properties of the Median Window," in Transformations in Optical Signal Processing Proc. of SPIE, vol. 373, pp. 219-224, 1981

[21] D.H. Berger, "Texture as a Discriminant of Crops on Radar Imagery," IEEE Trans. on Geosc. Electr., Vol. GE-8, no. 4, Oct. 1970, pp. 344-348.

[22] T.L. Chang, "Texture Analysis of Digitized Fingerprints for Singularity Detection," Proc. of the 5th Int. Conf. on Pattern Recognition, 1980, pp. 478-480.

[23] D.G. Hubel and T.N. Wiesel, "Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex," J. Physiology (London), Vol. 160, pp. 106-154, 1962.

[24] P.H. Lindsay and D.A. Norman, Human Information Processing, Academic Press, New York, 1972.

[25] A. Rosenfeld and E.B. Troy, "Visual Texture Analysis," Conf. Rec. for Symposium on Feature Extraction and Selection in Pattern Recognition, IEEE publication 70-C-51C, Argonne, Ill., Oct. 1970 pp.115-124.

[26] A. Rosenfeld and M. Thurston, "Edge and Curve Detection for Visual Scene Analysis," IEEE Trans. Comput., Vol. C-20, pp. 562-569, May 1971.

[27] R. Sutton and E. Hall, "Texture Measures of Automatic Classification of Pulmonary Disease," IEEE Trans. Comput., Vol. C-21, No. 1, pp. 667-676, 1972.

[28] D.H. Ballard and C.R. Brown, Computer Vision, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[29] R.M. Haralick, K. Shanmugam, and I. Denstein, "Textural Features for Image Classification," IEEE Trans. Syst., Man. Cybern., Vol. SMC-6, pp. 610-621, 1973.

[30] M. Galloway, "Texture Analysis Using Gray Level Run Lengths," Comput. Graph. Image Proc., Vol. 4, pp. 172-199, 1974.

[31] J. Weszka, C.R. Dyer, and A. Rosenfeld, "A Comparative Study of Textures Measures for Terrain Classification," IEEE Trans. Syst., Man, Cybern., Vol. SMC-6, pp. 269-285, Apr. 1976.

[32] R. Bajcsy, "Computer Identification of Visual Surfaces," Comput. Graph. Image Proc., Vol. 2., pp. 118-130.

[33] S.W. Zucker, "Toward a Model of Texture," Comput. Graph. Image Proc., Vol. 5, pp. 190-202, 1976.

[34] L. Carlucci, "A Formal System for Texture Languages," Pattern Recog., Vol. 4, pp. 53-72, 1972.

[35] S.Y. Lu and K.S. Fu, "A Syntactic Approach to Texture Analysis," Comput. Graph. Image Proc., Vol. 7, pp. 303-330, 1978.

[36] R.W. Ehrich and J.P. Foith, "A View of Texture Topology and Texture Description," Comput. Graph. Image Proc., Vol. 8, pp. 174-202, 1978.

[37] R.W. Conners and C.A. Harlow, "Towards a Structural Textural Analyzer Based on Statistical Methods," Comput. Graph. Image Proc.., Vol. 12, pp. 224-256, 1980.

[38] B.P. Kjell and C.R. Dyer, "Edge Separation and Orientation Texture Measures," Conf. Rec. Pattern Recog. Conf. (1985) pp. 306-311.

[39] F.M. Vilnrotter, R. Nevatia, and K.E. Price, "Structural Analysis of Natural Textures", IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-8, pp. 76-89, 1986.

[40] M.D. Levine, Vision in Man and Machine, McGraw-Hill, New York, 1985.

[41] R.M. Haralick, "Statistical and Structural Approaches to Texture," Proc. IEEE, Vol. 67, pp. 786-804, 1979.

[42] P. Brodatz, Textures, New York: Dover, 1966.

[43] T. Pavlidis, "Segmentation of Pictures and Maps Through Functional Approximation," Computer Graphics and Image Processing 1, 1972, pp. 360-372.

[44] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by Directed Split-and-Merge Procedure," Proc. Sec. Int. Joint Conf. Pattern Recognition, 1974, pp. 424-433.

[44] E.M. Riseman and M.A. Arbib, "Segmentation of Static Scenes," Computer Graphics and Image Processing 6, 1977, pp. 221-276.

[46] S.W. Zucker, "Region Growing: Childhood and Adolescence," Computer Graphics and Image Processing 6, 1976, pp. 382-399.

[47] T. Kanade, "Region Segmentation: Signals vs Semantics," Computer Graphics and Image Processing 13, 1980, pp. 279-297.

[48] P.V.C. Hough, "Method and Means for Recognizing Complex Patterns." U.S. Patent 3,069,654, Dec. 18, 1962.

[49] R.O. Duda, and P.E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Commun. Ass. Comput. Mach., Vol. 15, pp. 11-15, 1972.

[50] C. Kimme, D. Ballard, and J. Sklansky, "Finding Circles by an Array of Accumulators," Commun. Ass. Comput. Mach., Vol. 18, pp. 120-122.

[51] J. Sklansky, "On the Hough technique for curve detection" IEEE Trans. Comput., Vol. C-24, pp. 923-926, Oct. 1978.

[52] S.D. Shapiro, "Transformations for the Computer Detection of Curves in Noisy Pictures," Comput. Graph. Image Proc., Vol. 4, pp. 328-338, Dec. 1975.

[53] J. Radon, Ber. Sachs. Akad. Wiss, Leipzig 69 (1917) 262.

[54] I.M. Gelfand, M.I. Graev, and N.Y. Vilkin, Generalized Functions, Vol. 5, McGraw-Hill, New York, 1966.

[55] S.R. Deans, IEEE Trans. Pattern Anal. Mach. Intell., PAMI-2 (1981) 185.

[56] Brown, Curtiss and Sher, "Advanced Hough Transform Implementations," Proceedings IJCAI, 1983

[57] G. Eichmann and Z. Dong, "Coherent Optical Production of the Hough Transform," App. Opt., Vol. 22, No. 6 (1982).

[58] G.R. Gindi and A.F. Gmitro, Opt. Engr., 23, 499 (1984).

[59] W.H. Steier and R.K. Shori, "Optical Hough Transform," ICQE (1985).

[60] E. Ataman, V.K. Aatre, K.N. Wong, "A Fast Method for Real-Time Median Filtering," IEEE Trans. Acoust. Speech and Signal Processing, Vol. ASSP-28, pp. 415-421, 1980.

[61] P.E. Danielsson, "Getting the median faster," Comput. Graph. Image Proc. 17, pp. 71-78, 1981.

[62] Urs E. Ruttimann, R.L. Webber, "Fast Computing Median Filters on General-Purpose Image Processing Systems," Optical Engineering, Vol. 25, pp. 1064-1067, Sept. 1986.

[63] G.J. Wolfe and J.L. Mannos, "Fast Median Filter Implementations," in Applications of Digital Image Processing III, Proc. SPIE 207, pp. 154-160, 1979.

[64] D.R. Morgan, "Analog sorting network ranks inputs by amplitude and allows selection," Electron. Design, pp. 72-73 Jan. 1973. Also, "Correction," Electron. Design, p. 1, Aug. 1973.

[65] W. Swindell, H. Barrett, "Real-time Median-Window Filtering of Video Signals," in Transformations in Optical Signal Processing, Proc. SPIE 373, pp. 135-139, 1981.

[66] V.V. Bapeswara Rao and Sankara Rao, "A New Algorithm for Real-Time Median Filtering," IEEE Trans. Acoust. Speech and Signal Processing, Vol. ASSP-34, pp. 1674-1675, 1986.

[67] G. Gariboto and L. Lambarelli, "Fast On-line Implementation of Two dimensional Median Filtering," Electron. Lett., Vol. 15, pp. 24-25, Jan 1979.

[68] "Four-Chip Set Processes Images in Real Time," Electron. Design, pp. 39-46, May 1987.

[69] "CMOS building blocks shrink and speed up FFT systems," Electron. Design, pp. 101-106, Aug. 1987.

[70] "DSP chip runs Fourier transforms on nonstop input," Electronic Products, pp. 26-32, June 1987.

[71] L.K. Rangasami and A. Khotanzad, "A Model-Based Method for Rotation Invariant Texture Classification," IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-8, No. 4, pp. 472-81, 1986.

[72] O.D. Faugeras and W.K. Pratt, "Decorrelation Methods of Texture Feature Extraction," IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-2. pp. 323-332, July 1980.

[73] T. Kohonen, Self-organization and Associative Memory, Springer-Verlag, New York, 1984.

[74] S.A. Dudani and A.L. Luk, "Locating Straight-Line Edge Segments on Outdoor Sceenes," Pattern Recognition, Vol. 10, No. 3, 1978, pp. 145-157.

[75] T.N.E. Greville, "Some Applications of the Pseudo-inverse of a Matrix," SIAM Rev., 2, pp. 15-22, 1960.

[76] A. Albert, Regression and the Moore-Penrose Pseudo-inverse, Academic Press, New York, 1972.