TEXTURE ANALYSIS USING COOCCURRENCE MATRICES

by

DURUVASULA KANAKA DURGA

B.S., University of Mysore, India, 1984

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

D.V. Satish Chandra.

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Texture is observed in objects which exhibit some kind of structural pattern such as wood, raffia, grass, wool, sand, canvas etc. These texture models play a very important role in analysis and synthesis of images. Textural features computed from these images are used for image classification.

The textures are broadly classified into macrotextures and microtextures. The distinction between these textures is mainly based on the size of the texture elements or primitives. Macrotextures are assumed to be generated by larger primitives, whereas microtextures are generated by smaller primitives. A microtexture can be easily and adequately described by a number of attributes of textures such as homogeneity, coarseness, directionality etc. There are very few macroanalysis texture procedures described in literature.

The primary purpose of this project is to compute the various features which describe a texture like homogeneity, contrast, inertia, correlation etc., and to classify the texture using a popular technique viz., Spatial Gray Level Dependence Method (SGLDM) to compute the cooccurrence

matrices. These matrices are intermediate matrices which contain most of the textural information and play a very important role in discriminating man-made textures rather than the natural textures. The different features computed by these intermediate matrices measure the visual qualities of different patterns. This technique was particularly chosen for a number of reasons. The first and the foremost reason is that studies show that SGLDM is the most powerful statistical technique compared to other texture analysis algorithms like gray level run length method, gray level difference method etc. The second reason is that its wide variety of applications to a number of image classification procedures. The third reason is the simplicity and effectiveness associated with it.

The results of this report show that SGLDM can be used for texture analysis. The features extracted from spatial gray level dependence matrices contain information about image textural characteristics such as periodicity, homogeneity, contrast, inertia, gray level linear dependencies (correlation), and complexity of the image. Subsequently, the usefulness of the textural features in classifying different images used in this study viz., raffia, french canvas, straw, paper, cork, wire and wood is investigated.

CHAPTER 2: CLASSIFICATION OF TEXTURES

Textures are classified as fine, smooth, granulated, regular, irregular, periodic etc. Textures can be classified as natural or artificial textures, macrotextures or microtextures depending upon the qualities the texture elements exhibit. In image analysis textures are classified into two main categories : statistical and structural [8]. The classification is shown in the Figure 2.1-1.

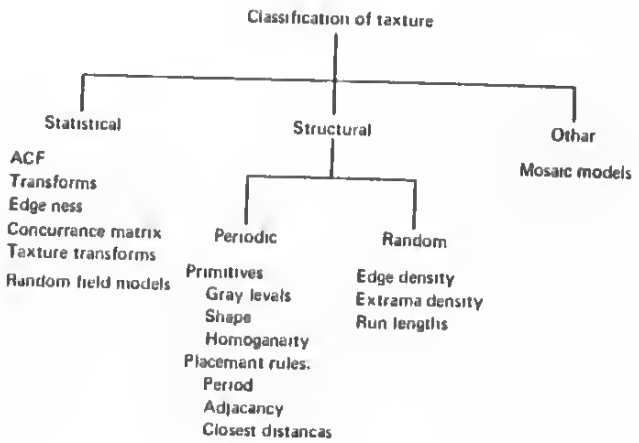


Figure 2.1-1 Texture Classification [A.K. Jain, 'Fundamentals of Digital Image Processing', Prentice Hall Inc., New Jersey, 1989, p395].

2.1 STATISTICAL APPROACH

A texture is defined by a set of statistical properties extracted from a family of picture properties.

Simple statistics such as first order statistics to higher order statistics can be used to classify textures. While first order statistics are used to classify only a limited set of textures, second order statistics are used a great deal in classifying textures. Gray level cooccurrence matrices, which are the main focus of this project, are computed using second order statistics. Higher order statistics were made use of in gray level runlength, Fourier power spectrum, autoregression model etc. Following are some of the statistical models used to classify textures.

AUTOCORRELATION FUNCTION (ACF)

ACF can be used to describe the coarseness or fineness of textures. The width of ACF is considered to be proportional to the coarseness of a texture. Resolution of the image is a factor which plays an important role in the spread of ACF. Since different images can have the same ACF, ACF alone is not sufficient to distinguish among several texture fields.

IMAGE TRANSFORMS

A two-dimensional transform of the input image can be used to estimate the coarseness, fineness and orientation. The transform of an image is passed through several band pass filters or masks. With circular or

angular slits, energy is measured in different frequency bands. These are useful in detecting orientation and periodic or quasiperiodic properties of the textures.

EDGENESS

Density of edge pixels can be used to measure the coarseness of random texture.

COOCCURRENCE MATRIX

Many forms of cooccurrence matrices are described in literature [12]. They are all based on the same cooccurrence principle. Some of the important techniques are described briefly.

Spatial gray level dependence method which is the most powerful technique was suggested by Haralick et al.[7]. The cooccurrence matrix $S(i,j/d,\theta)$ is a probability distribution matrix of relative frequencies where the resolution cells with the gray levels i and j are separated by a distance d in the direction of θ.

A study using Generalized Cooccurrence Matrices(GCM) was performed by Davis et al.[3]. Computation of GCM is very similar to SGLDM, but this method identifies the spatial distribution of elements like edges and lines. SGLDM describes the distribution of pixel intensities. GCM

can be computed by specifying three parameters. For example, the parameters could be the edge pixel, the orientation and the specified distance. Consider a GCM(M^{O},N^{O}), where each element of GCM is increased by one if there is an edge pixel with M^{O} and a neighbouring edge pixel with N^{O} within the specified distance S.

The gray level run length method was suggested by Galloway [5]. The gray level run is defined as the set of picture points having the same gray level value and the run length being the total number of points in the run. Each element in the run length matrix $S(i,j/\theta)$ is the count of the number of times the picture consists of gray levels i in the run length j for different angles of θ.

An angularly independent technique of computing cooccurrence matrices was suggested by Sun and Wee [11]. These matrices are called Neighbouring Gray Level Dependence Matrices (NGLDM). These matrices are computed by the gray level relationship of an element with its neighbours in all directions instead of a particular direction. A Q matrix, Q(K,S) is constructed using NGLDM method such that K is the gray level and S is the NGLDM number. The larger the S the smoother the image and vice versa. This technique is insensitive to rotation and linear gray level transformation.

RANDOM TEXTURE FIELDS

Several texture models were suggested for texture classification. De Souza [4] suggested an autoregressive model for texture classification. Kashyap et al., [10] suggested another random field model known as Simultaneous Auto Regressive model (SAR). Natural textures can be classified using Gaussian-Markov model and was suggested by Kaneko and Yodogawa [9].

2.2 STRUCTURAL APPROACH

A number of textural classifications have been described based on primitives and placement rules. Texels or texture elements are defined by their gray level, shape and homogeneity. Placement rules define spatial relationships. For deterministic structures the spatial relationships may in turn be expressed in terms of adjacency, closest distance, periodicities etc. For randomly placed texels, placement rules can be expressed in terms of edge density, run lengths of maximally connected texels and extrema density.

2.3 OTHER APPROACHES

Other approaches include mosaic models which are

combination of both statistical and structural approaches. Random geometrical processes are represented by these models.

CHAPTER 3: SPATIAL GRAY LEVEL DEPENDENCE METHOD AND TEXTURAL FEATURES FOR IMAGE CLASSIFICATION

3.1 IMAGE DEFINITIONS

Image refers to a two-dimensional intensity function $f(x,y)$ where variables x and y denote the spatial coordinates and the value of f at any point (x,y) is proportional to the gray level of the image. A continuous image $f(x,y)$ is approximated by a N_x x N_y array where N_x is the number of resolution cells in the x direction and N_y is the number of resolution cells in the y direction. Each element of the array which is referred to as pixel or pel is assigned some gray value G which ranges from 1 to N_g.

Classification of pictorial data is achieved by basically performing various analyses on the two-dimensional image $f(x,y)$. Various features are extracted from the block of resolution cells and images are classified using a linear classifier which is one of the pattern recognition techniques.

3.2 SPATIAL GRAY LEVEL DEPENDENCE METHOD (SGLDM)

Computation of spatial gray level dependence matrices is one of the most essential components of SGLDM. The SGLDM

method is based on the second order joint conditional probability density functions $S(i,j/d,\theta)$, where each element is the estimated probability of going from gray level i to gray level j given the interspace sampling distance d and the angle θ. Computation of these matrices can be explained by the following notion of adjacent or nearest neighbour resolution cells as shown in Figure 3.2-1. The resolution cell (rc) under consideration is adjacent to its neighbours in the direction of 0^{o}, 45^{o}, 90^{o}, and 135^{o}. For example, cells #4 and #5 are the nearest neighbours in 0^{o} direction to the rc, cells #2,and #7 are the nearest neighbours in 90^{o} direction to the rc under consideration. The principal orientations are 0^{o}, 45^{o}, 90^{o}, and 135^{o}. Neighbours along 0^{o} are horizontal neighbours, neighbours along 90^{o} are vertical neighbours and neighbours along 45^{o} and 135^{o} are neighbours along right diagonal and left diagonal directions respectively. The commonly used distances are 1 to 80. Discriminatory power can be increased by considering more values of θ and d.

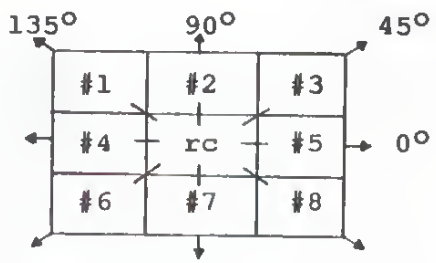


Figure 3.2-1 Nearest neighbours to the rc under consideration

Computation of spatial gray level dependence matrices is easy and straightforward. A 4x4 image with four gray levels ranging from 0-3 and the generalized form of any spatial gray level dependence matrix are shown in the Figure 3.2-2. Each element #(i,j) in the spatial gray level dependence matrix is a count of the number of times the gray levels i and j are adjacent to each other at a given angle. For example, consider an element in the (1,2) position. In distance one horizontal S_h matrix, the element in the (1,2) position is a count of the total number of times two gray levels 1 and 2 occurred horizontally adjacent to each other at a distance of one. Similarly in distance 1 vertical S_v matrix, the element in (1,2) position is the total number of times two gray levels 1 and 2 occurred vertically adjacent to each other at a distance of one. The spatial gray level dependence matrices, S_h and S_v at d = 1 are shown in the Figure 3.2-3.

image :

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

j \ gray level i			
#(0,0)	#(0,1)	#(0,2)	#(0,3)
#(1,0)	#(1,1)	#(1,2)	#(1,3)
#(2,0)	#(2,1)	#(2,2)	#(2,3)
#(3,0)	#(3,1)	#(3,2)	#(3,3)

Figure 3.2-2 A 4x4 image and the generalized form of spatial gray level dependence matrix.

S_h: (d = 1)

$$\begin{vmatrix} 4 & 2 & 1 & 0 \\ 2 & 4 & 0 & 0 \\ 1 & 0 & 6 & 1 \\ 0 & 0 & 1 & 2 \end{vmatrix}$$

S_v: (d = 1)

$$\begin{vmatrix} 6 & 0 & 2 & 0 \\ 0 & 4 & 2 & 0 \\ 2 & 2 & 2 & 2 \\ 0 & 0 & 2 & 0 \end{vmatrix}$$

Figure 3.2-3 Horizontal and vertical spatial gray level dependence matrices at d = 1

Using the generalized formulas, spatial gray level dependence matrices can be derived for $\theta = 0^o, 45^o, 90^o, 135^o$.

$$S_h = S(i,j/d,0^o)$$
$$= \#\{((k,l),(m,n) : f(k,l)=i,\ f(m,n)=j,\ [|k-m|=d,\ l-n=0]\} \quad (3.2\text{-}1)$$

$$S_l = S(i,j/d,135^o)$$
$$= \#\{((k,l),(m,n) : f(k,l)=i,\ f(m,n)=j,\ [(k-m=d,\ l-n=-d) \text{ or } (k-m=-d,\ l-n=d)]\} \quad (3.2\text{-}2)$$

$$S_v = S(i,j/d,90^o)$$
$$= \#\{((k,l),(m,n) : f(k,l)=i,\ f(m,n)=j,\ [k-m=0,\ |l-n|=d]\} \quad (3.2\text{-}3)$$

$$S_r = S(i,j/d,45^o)$$
$$= \#\{((k,l),(m,n) : f(k,l)=i,\ f(m,n)=j,\ [(k-m=d,\ l-n=d) \text{ or } (k-m=-d,\ l-n=-d)]\} \quad (3.2\text{-}4)$$

where,

$S(i,j/d,\theta)$ is the matrix of relative frequencies where the neighbouring resolution cells with the gray levels i and j are separated by distance d in the direction of angle θ,

# denotes the number of elements in the set,

i and j are the gray levels of the cells located at (k,1) and (m,n), and

d is the distance between the resolution cells with the gray levels i and j.

The different distances indicated in the above equations are explained as follows. A 3x3 image is shown in the Figure 3.2-4. Consider elements in (k,1),(m,n) or (m,n),(k,1) positions. For example, if (1,1),(1,2) or (1,2),(1,1) are distance one horizontal neighbours ($\theta = 0^{o}$), then (k-m) = 0 and |1-n| = d = 1. Similarly, if (1,1),(2,1) or (2,1),(1,1) are vertical neighbours ($\theta = 90^{o}$), then |k-m| = d = 1, (1-n) = 0.

Similarly, if (1,2),(2,3) or (2,3),(1,2) are right diagonal neighbours ($\theta = 45^{o}$), then (k-m) = -d = -1, (1-n) = -d = -1 or (k-m) = d = 1, (1-n) = d = 1.

Similarly, if (2,1),(1,2) or (1,2),(2,1) are left diagonal neighbours ($\theta = 135^{o}$), then (k-m) = d = 1, (1-n) =-d =-1 or (k-m) = -d = -1, (1-n) = d = 1.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

Figure 3.2-4 A 3x3 image.

The matrices that are obtained are unnormalized frequency matrices. Appropriate normalized frequency matrices can be computed. Normalization of the matrix can be done by dividing each entry in the matrix by an integer R. For distance 1 and 0° direction ($d = 1$, $\theta = 0^{\circ}$) there are a total of $R = 2 \times N_x - 1 \times N_y$ horizontal pairs. When the distance is 1 and $\theta = 45^{\circ}$ ($d = 1$, $\theta = 45^{\circ}$) there are a total of $R = 2 \times (N_y - 1) \times (N_x - 1)$ right diagonal pairs. By symmetry, there are $R = 2 \times N_x \times N_y - 1$ vertical neighbour pairs with $d = 1$ and $\theta = 90^{\circ}$ and $R = 2 \times N_x - 1 \times N_y - 1$ horizontal neighbour pairs with $d = 1$ and $\theta = 135^{\circ}$.

3.3 EXTRACTION OF TEXTURAL FEATURES

A set of textural features computed from cooccurrence matrices can be used to determine various visual qualities of a pattern. There is a loss of textural information while computing cooccurrence matrices from the digital image; again there is a loss of information when the transition occurs from the cooccurrence matrix to a set of textural features. A set of textural features are needed to describe a pattern since any one of them does not contain all the textural information. In this study, textural features computed from cooccurrence matrices include entropy, correlation, homogeneity, inertia measure and contrast.

HOMOGENEITY (ASM)

This is measured from the following formula.

$$L(S(d,\theta)) = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} [S(i,j/d,\theta)] \qquad (3.3\text{-}1)$$

The angular second moment (ASM) is a measure of homogeneity of the image. If the image is homogeneous, the cooccurrence matrix consists of a very few entries of larger magnitude and on the other hand, if the image is nonhomogeneous, there are a large number of entries with smaller magnitude.

CORRELATION

Correlation is given by

$$C(S(d,\theta)) = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} (i-\mu_x)(j-\mu_y)\, S(i,j/d,\theta)/\sigma_x\sigma_y \qquad (3.3\text{-}2)$$

where $\mu_x = \sum_{i=0}^{N_g-1} i \sum_{j=0}^{N_g-1} S(i,j/d,\theta)$

$\mu_y = \sum_{j=0}^{N_g-1} j \sum_{i=0}^{N_g-1} S(i,j/d,\theta)$

$\sigma_x = \sum_{i=0}^{N_g-1} (i-\mu_x) \sum_{j=0}^{N_g-1} S(i,j/d,\theta)$

$\sigma_y = \sum_{j=0}^{N_g-1} (j-\mu_y) \sum_{i=0}^{N_g-1} S(i,j/d,\theta)$

Correlation is a measure of linear dependencies of the image.

ENTROPY

Entropy is a measure of change of brightness. Entropy is small when the change of brightness in the region is severe. Entropy is large when change in brightness is smooth and is given by the following formula.

$$H(S(d,\theta)) = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} S(i,j/d,\theta) \log S(i,j/d,\theta) \quad (3.3\text{-}3)$$

CONTRAST

The contrast is a measure of the amount of local variations of the image and is given by the following formula.

$$L(S(d,\theta)) = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} |i-j| \, S(i,j/d,\theta) \quad (3.3\text{-}4)$$

INERTIA

The inertia measure is believed to measure the qualities of texture periodicity and texture gradient. The importance of periodicity detection is that it can be used to identify any special type of unit patterns and is given by the following formula.

$$I(S(d,\theta)) = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} (i-j)^2 \, S(i,j/d,\theta) \quad (3.3\text{-}5)$$

The textural features described above are the most commonly used textural measures and this measurement set

was used in this study. Many textural measures have been described in the literature, out of which some measures seems to be quite promising. New textural measures like cluster shade and cluster prominence which were suggested by Connors et al.[2], are believed to measure the uniformity and proximity of a texture. A few other statistical texture measures like sum entropy, sum variance etc., were suggested by Haralick et al.[7].

CHAPTER 4: RESULTS

4.1 INTRODUCTION

The image processing system consists of a digitizing camera, a video display monitor and a general purpose computer. The images used for this project are digital images recorded using a digitizing camera. A video display monitor may be used to display the images. These operations or commands are administered using a general purpose computer.

The image processing system that was used for this study was Grinnell GMR 270 series. The Grinnell captures the images as 256 x 256 monochrome digital images using a television camera. Each image consists of 256 lines with the 256 pixels spaced along each line. The brightness for each pixel ranges from 0 to 255. A pixel with 0 gray level corresponds to a dark element and a pixel with gray level 255 corresponds to a bright element. In essence, the Grinnell image processing system digitizes and displays 256 x 256 pixels with 256 gray levels. These images were displayed on a Mitsubishi model #C3922 LPK high resolution color TV monitor. A host computer VAX 11/750 was used to control the operations of the Grinnell system.

Computer programs were written to obtain two types of plots, two-dimensional scatter plots,inertia plots and the final phase of the study included a classifier program. These programs are user interactive and hence provide flexibility to the user to some extent.

4.2 SCATTER PLOTS

Two-dimensional scatter plots or three-dimensional scatter plots are the plots of one feature against the other. These plots illustrate the classification power of the features. The images used for this study are 256 x 256 images with 256 gray levels. Four windows of size 32 x 32 with 32 gray levels were created for each image and hence four sets of samples were obtained for each image. A set of five features viz., inertia, entropy, correlation, contrast and homogeneity were computed from each of the reduced samples.

The two dimensional scatter plots which were employed in this study, plot one feature value against the other for all the eight images. For the scatter plots the following notation for eight classes of textures were used.

paper	=	p
wood	=	o
grass	=	g
cork	=	c

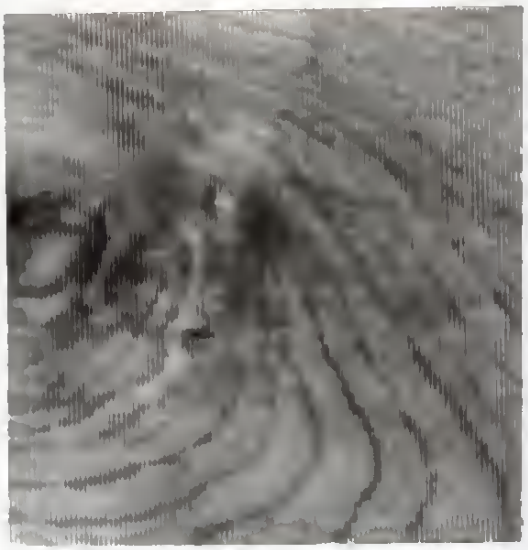
french canvas	=	f
raffia	=	r
straw	=	s
wire	=	w

The textures paper, wood, wire and cork are shown in Figure 4.2-1 and Figure 4.2-2. The textures french canvas, raffia, straw and grass will be shown later in this chapter. Figures 4.2-3 to 4.2-5 show sample scatter plots. Different classes of textures form different clusters which can be readily seen from these plots.

Figure 4.2-3 shows a scatter plot of correlation and contrast against entropy. The samples wire, wood, cork and to some extent paper form separate clusters. The remaining samples overlap one another. Correlation vs. contrast and correlation vs. entropy plots are shown in the Figure 4.2-4. These scatter plots show that most of the samples are overlapped except wood, french canvas and cork which form good clusters for classification. Figure 4.2-5 shows the scatter plots of correlation and entropy against contrast. Clusters of samples wood, wire and cork are readily seen from these plots, but the remaining samples do not form distinct clusters.

From these two-dimensional scatter plots, a conclusion can be drawn regarding the classification power of the features. These features exhibit a moderate classification

power. All these features can classify at least four texture classes. The samples grass, raffia, straw and to a certain extent paper could not be very well classified. Hence inertia plots were plotted to detect the textures which exhibit periodicity. The results are discussed in the next section.

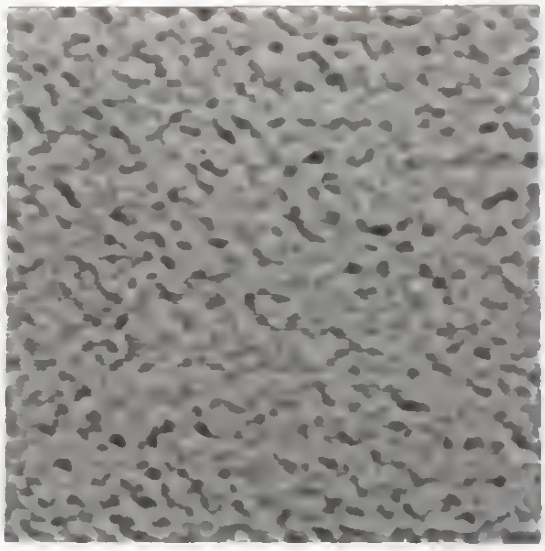


a. Wood

b. Wire

Figure 4.2-1 Texture samples

a. Cork



b. Paper

Figure 4.2-2 Texture samples

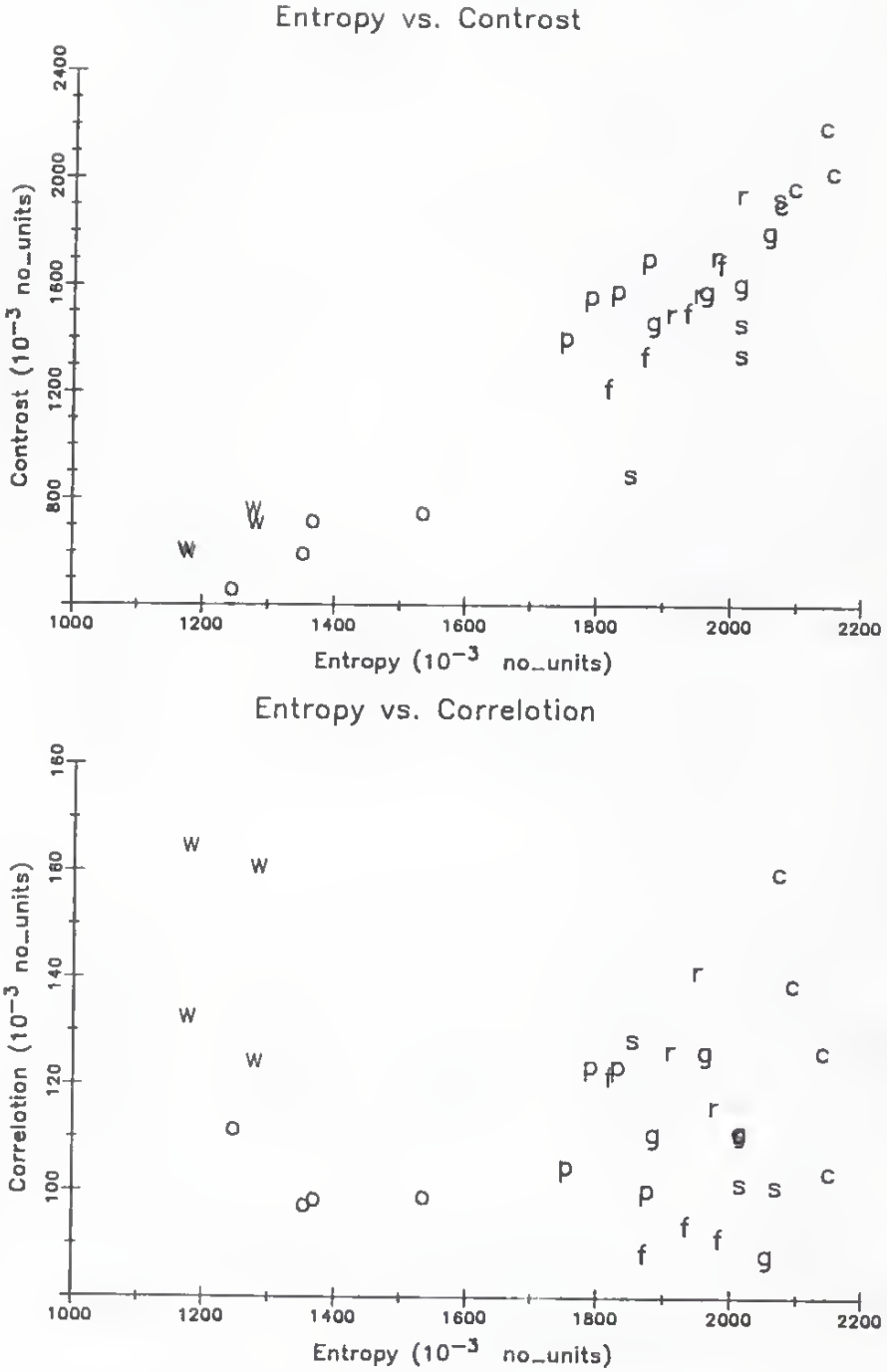


Figure 4.2-3 Scatter plots of entropy vs. contrast and correlation.

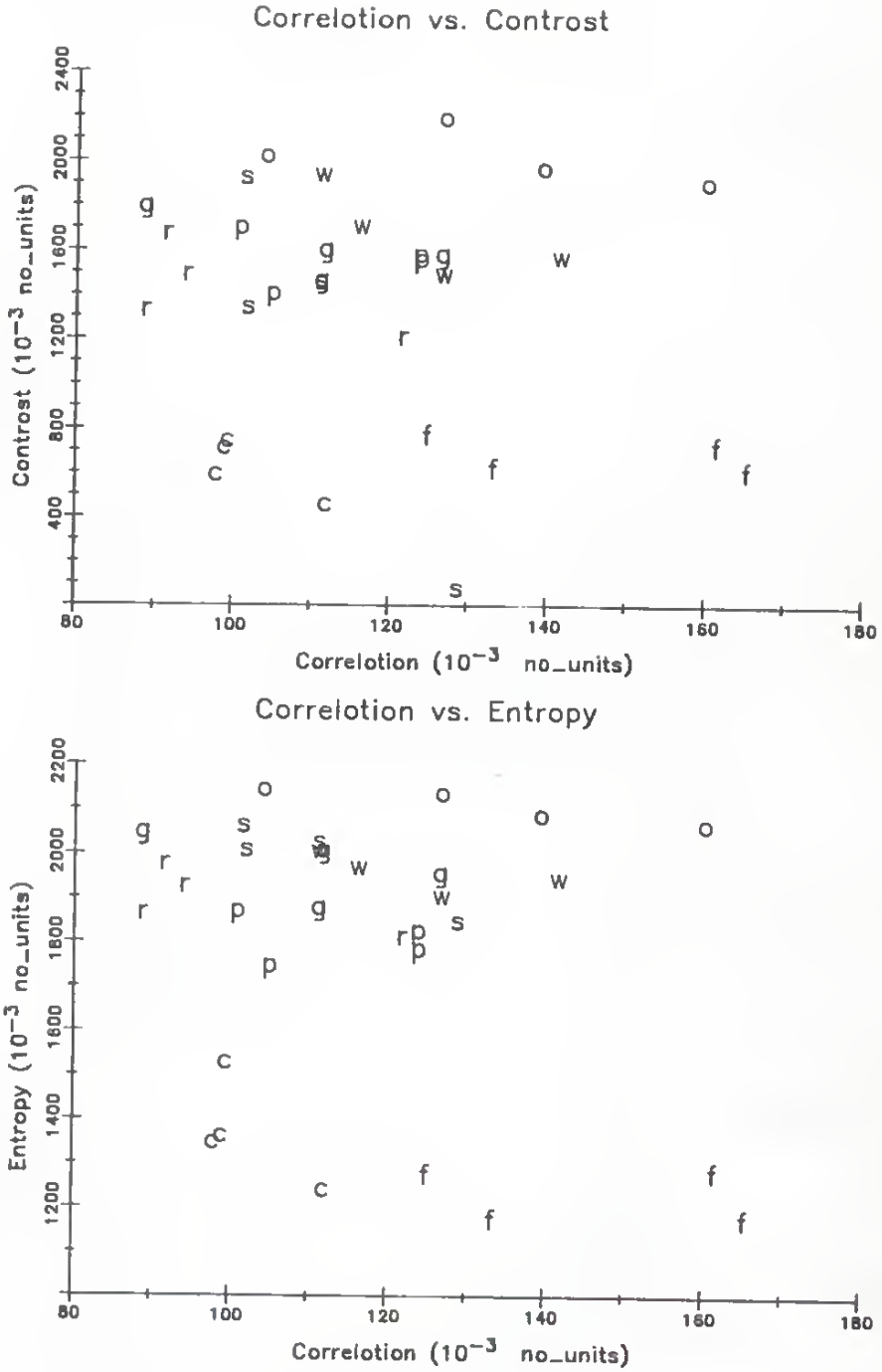


Figure 4.2-4 Scatter plots of correlation vs. contrast and entropy.

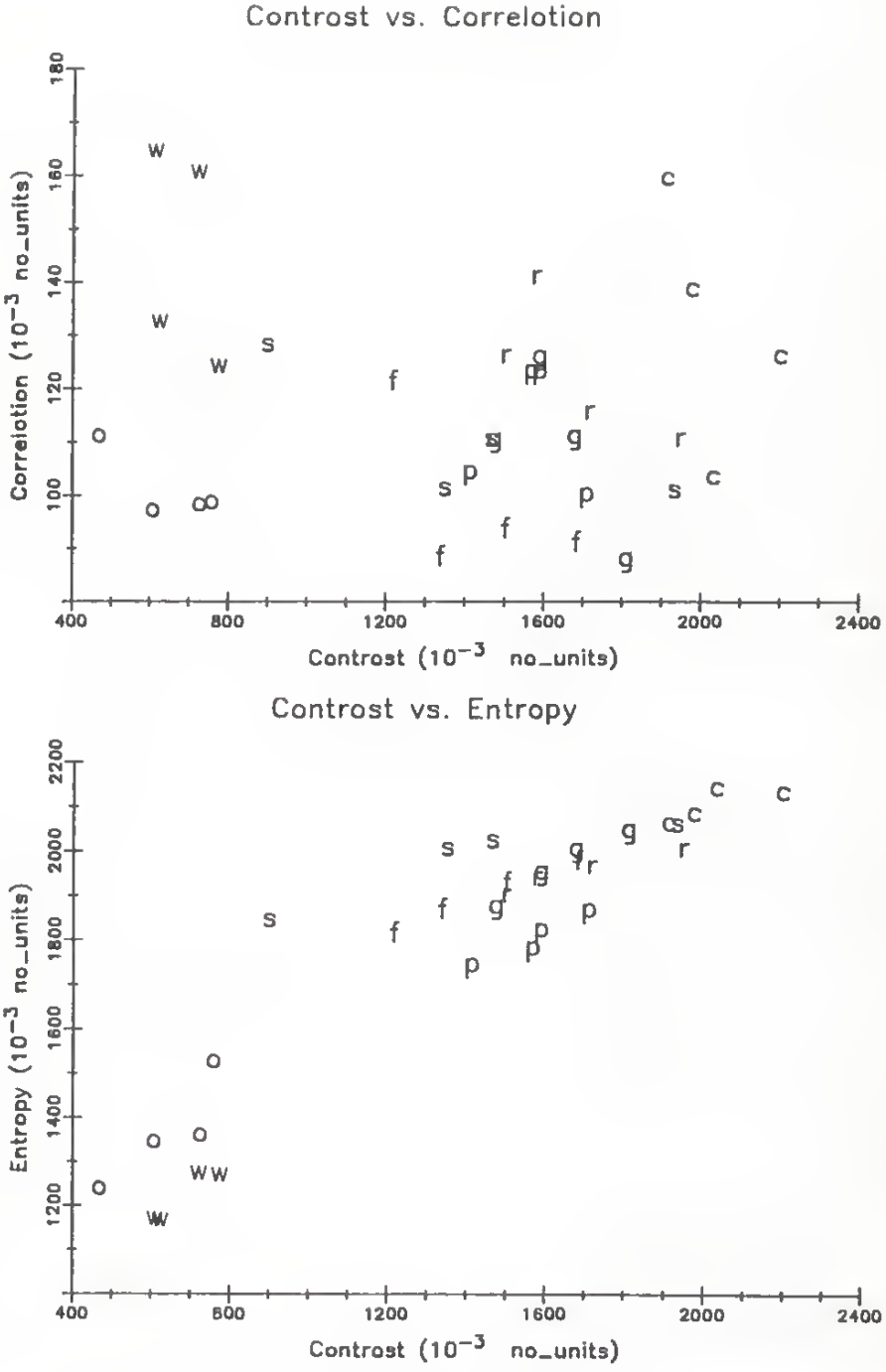


Figure 4.2-5 Scatter plots of contrast vs. correlation and entropy.

4.3 INERTIA PLOTS

Inertia measure is one of the textural measures computed from cooccurrence matrices and is given by the following formula.

$$I[S(d,\theta)] = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} (i-j)^2 \; S(i,j/d,\theta)$$

Discriminatory power and hence the classification power can be increased by increasing the number of angles and intersampling distances. The normally used angles are $\theta = 0^{o}, 45^{o}, 90^{o}$, and 135^{o} and the distances are d =1 to 80. In this study, horizontal and vertical inertia were computed along $\theta = 0^{o}$ and $\theta = 90^{o}$ for intersampling distances of d = 1 to 45.

The images used for inertia plots are raffia, french canvas, grass and straw. The important reason for selecting inertia measure is its ability to detect the texture periodicity and texture gradient. The scatter plots did not give a very good classification for aforementioned images. The images raffia and french canvas exhibit periodicity, which would distinguish them from grass and straw which do not exhibit any periodicity. This will be later shown in this section.

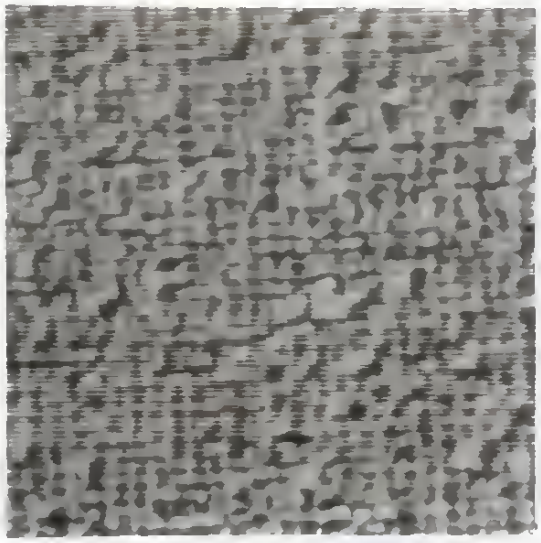
Figures 4.3-1 and 4.3-2 show the texture samples of raffia, french canvas, grass, and straw. Figure 4.3-3 shows

the plots of horizontal and vertical inertia values against intersampling distance for the image french canvas. The image french canvas was originally scanned at a resolution of 256 x 256 with 256 gray levels. The inertia values were computed from the reduced image by creating a window of size 45x45 and reducing the gray levels from 256 to 45. Note that in both the plots of Figure 4.3-3 periodicity is clearly exhibited by the image french canvas. These plots can also be used to guage the pattern size of the french canvas [1]. The horizontal and vertical inertia values are minimum at the intersampling distances of d_h = 21 and d_v = 21, which indicates that the size of the unit pattern for french canvas is 21 x 21 pixels. In other words the size of each unit pattern of canvas is 21 pixels in the horizontal direction and 21 pixels in the vertical direction. Similarly the next minimum inertia values are at d_h = 43 and d_v = 42 which again indicates that the size of the unit pattern is approximately 22 x 21 pixels. The periodicity is slightly distorted due to the nonuniformity of the texture fabric.

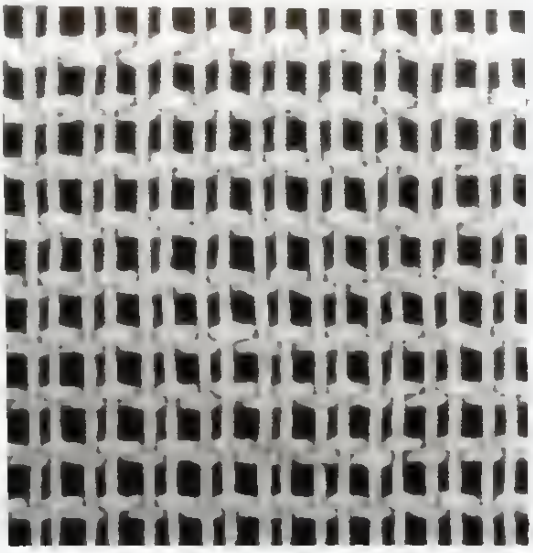
The texture of raffia shows the unit rectangular pattern which is a different unit pattern compared to canvas. The inertia values were again computed from a reduced image of 45 x 45 with 45 gray levels. Figure 4.3-4 shows the plots of horizontal and vertical inertia against

the intersampling distance d. Note a different type of periodicity is exhibited by the image raffia. It is again possible to measure the unit pattern size from these plots [1]. These plots indicate that the size of each pattern is 6x8 pixels, 6 pixels in vertical direction and 8 pixels in the horizontal direction. Figures 4.3-5 and 4.3-6 show the horizontal and inertia plots against intersampling distance d for the images straw and grass. These images were originally scanned at a resolution of 256x256 with 256 gray levels. The inertia values were computed for reduced images of size 45x45 with 45 gray values. These images can be clearly distinguished from raffia and canvas since these do not exhibit any periodicity which can be seen from these plots.

The reason for choosing the inertia measure as the feature to detect periodicity and its ability to guage the unit patterns is clearly demonstrated. Hence these plots definitely show a better classification for the images which exhibit periodicity in texture pattern.

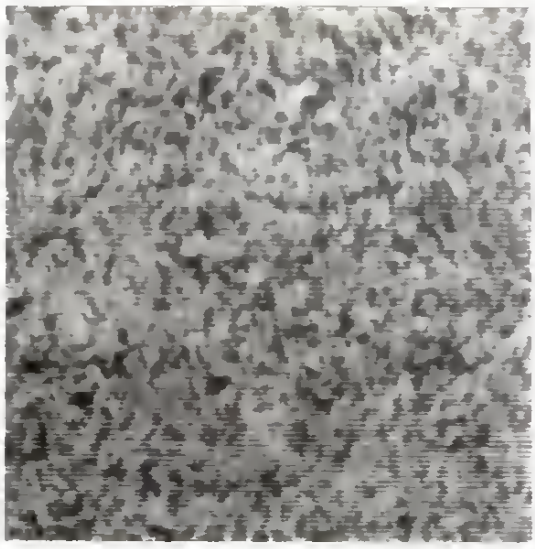


a. raffia

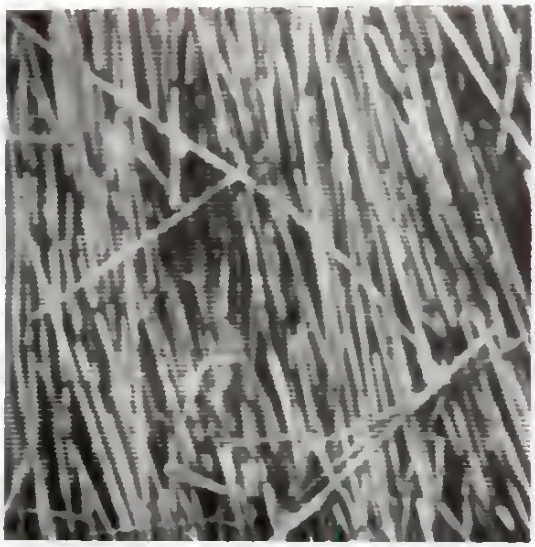


b. French canvas

Figure 4.3-1 Texture samples



a. Grass



b. Straw

Figure 4.3-2 Texture samples

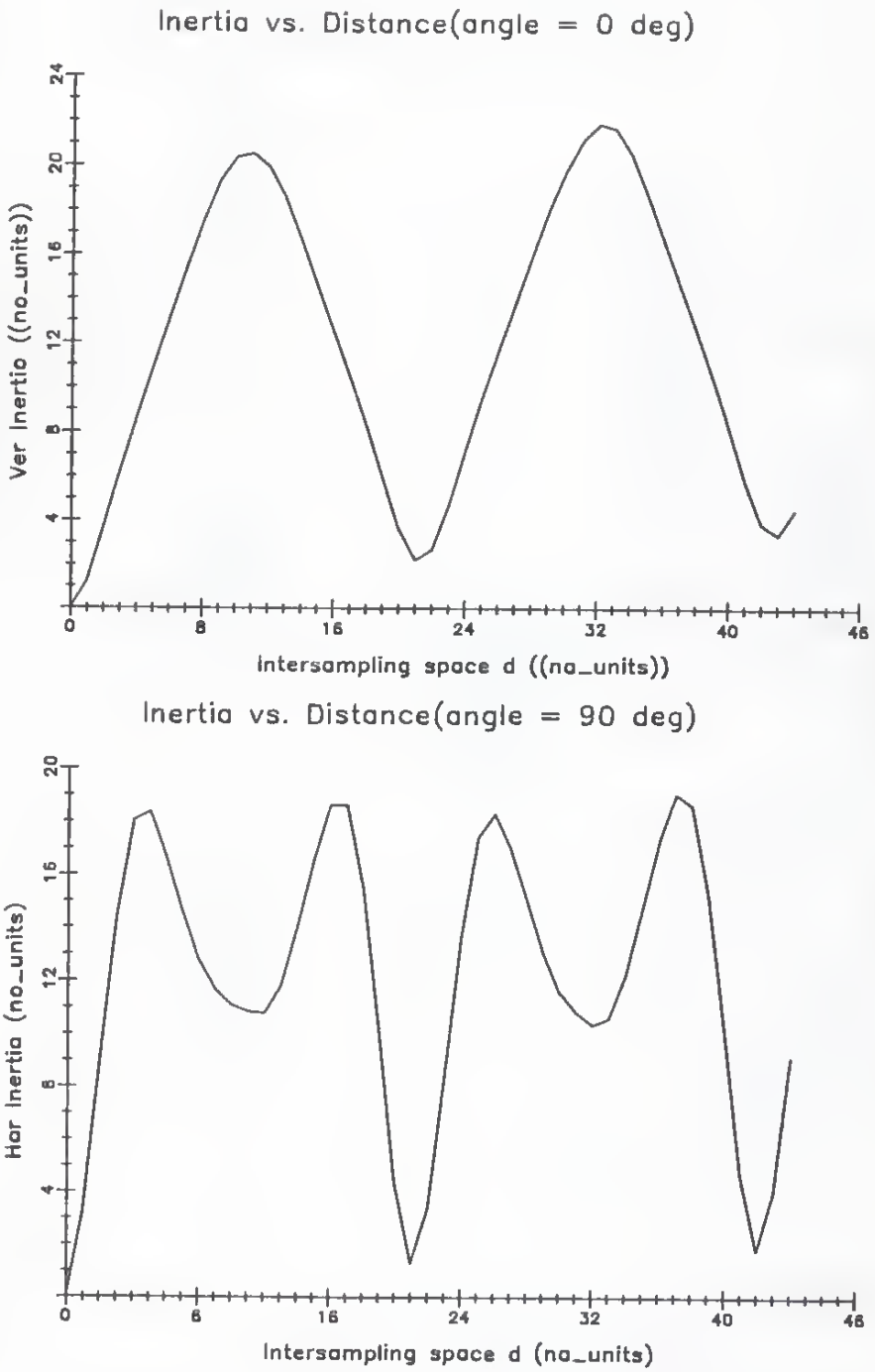


Figure 4.3-3 Inertia plots along 0° and 90°

for image french canvas

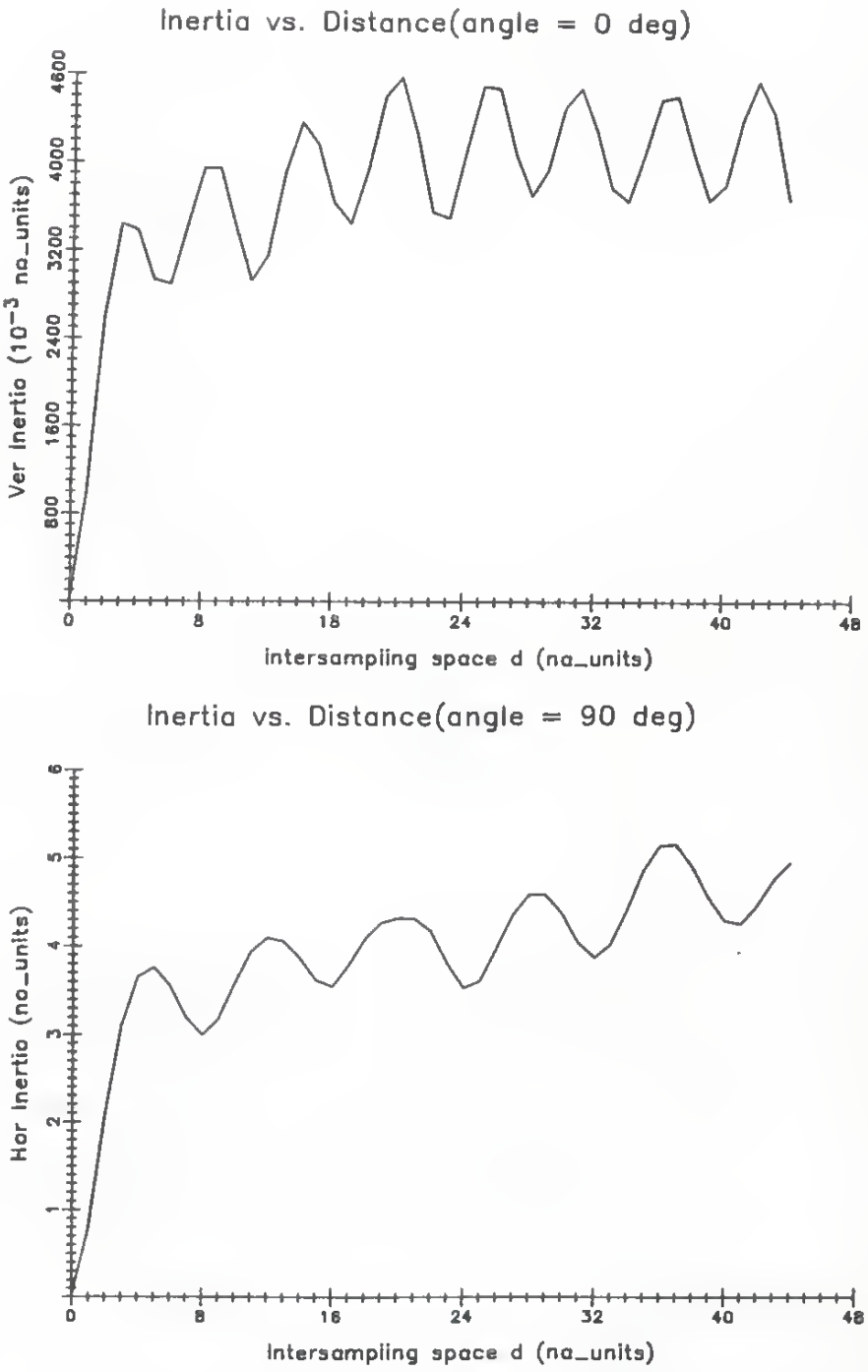


Figure 4.3-4 Inertia plots along 0° and 90°

for the image raffia

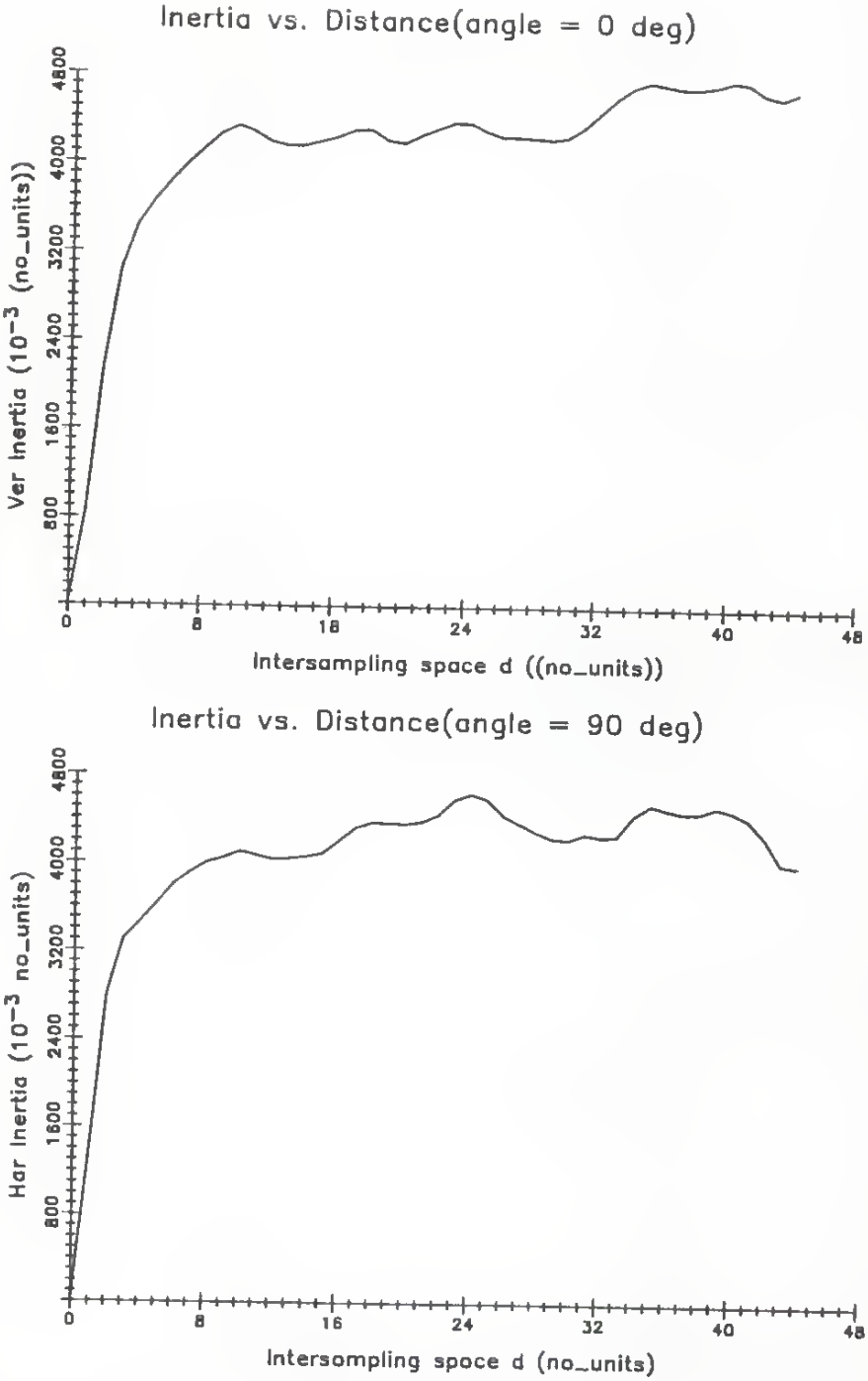


Figure 4.3-5 Inertia plots along 0^{o} and 90^{o} for the image grass

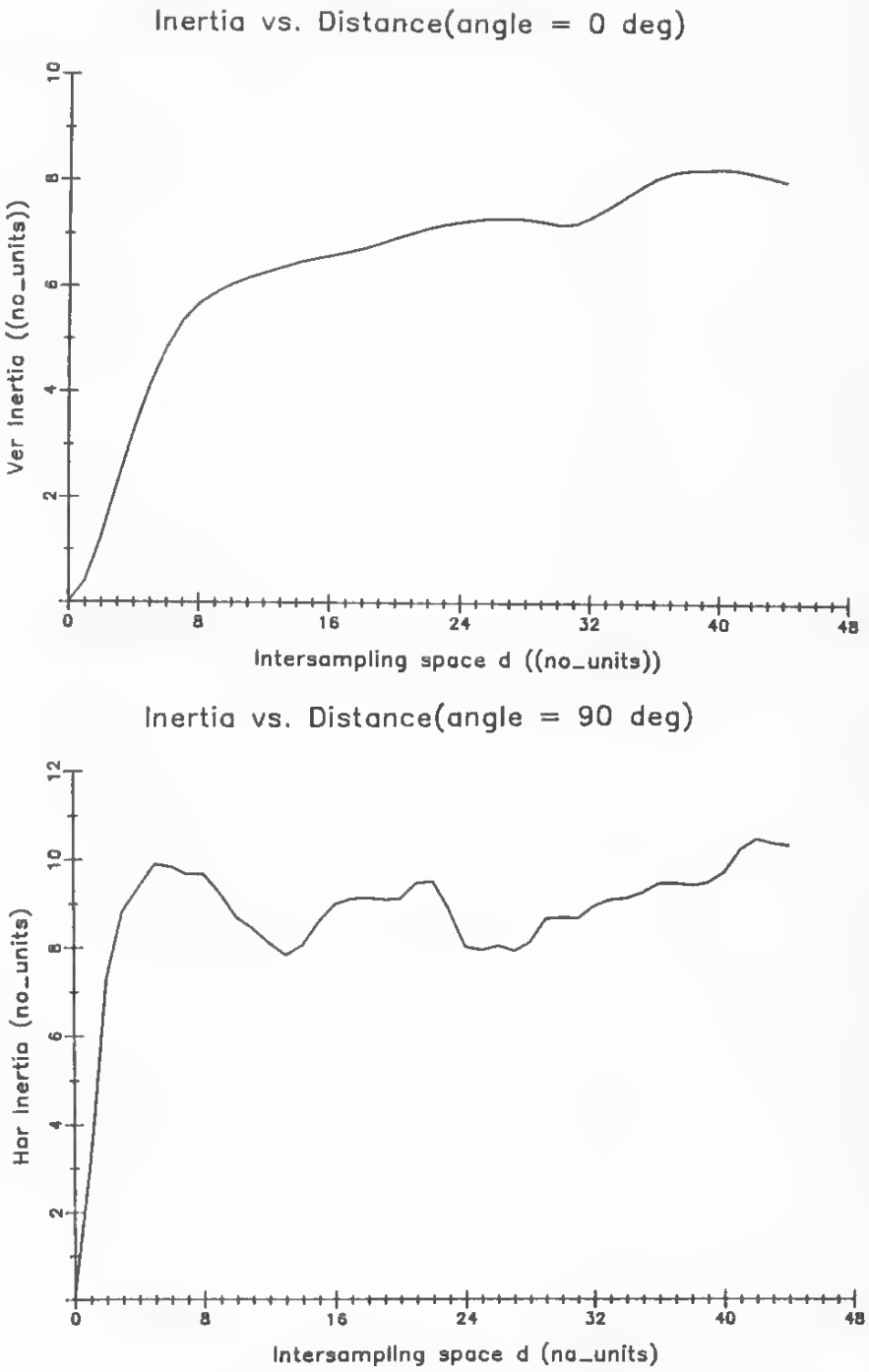


Figure 4.3-6 Inertia plots along 0^{o} and 90^{o} for the image straw

4.4 LINEAR CLASSIFIER FOR IMAGE CLASSIFICATION

Pattern recognition consists of classification of a set of processes. The classification of images or patterns is based on the measurements taken from the selected features like inertia, entropy, homogeneity, correlation and contrast. A pattern recognition system consists of a feature extractor and a classifier. The feature extractor extracts the feature measurements from the input patterns and a classifier performs the function of classification [6].

Pattern classification is basically a separation of feature space. A set of N features extracted from a pattern is called a feature vector which is shown in the Figure 4.4-1.

$$Y = \begin{vmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{vmatrix}$$

Figure 4.4-1 Feature vector

Classification of an image is to assign each feature vector a proper pattern class, which can be mathematically formulated in terms of discriminant functions. Let 1,2,..,m be the m texture classes to be separated. The discriminant functions associated with a pattern K is given by a linear

combination of feature vector Y multiplied by its appropriate weights as indicated by the formula,

$$D(Y) = \sum_{k=1}^{N} W_k Y_k + W_{n+1} \qquad (4.4\text{-}1)$$

A linear four class classifier is shown in the Figure 4.4-2. Classification of patterns is done by an adaptive classifier. The feature measurements for four images are impressed on the terminals which are multiplied by a set of weights. The sum is then passed through four threshold units.

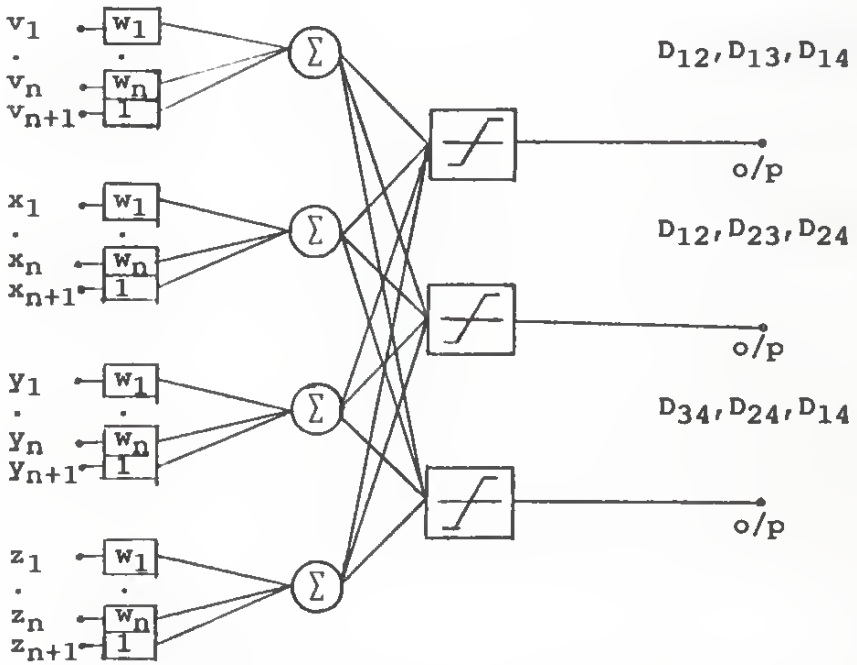


Figure 4.4-2 A linear four class classifier

Let $v_1,..,v_n, x_1,..,x_n, y_1,..,y_n$ and $z_1,..,z_n$ be the feature vectors for four images. Classification of the

images can be done by finding the different discriminant functions and satisfying a few conditions such as follows:

If $D_{12}>Th_1$ & $D_{13}>Th_1$ & $D_{14}>Th_1$, then V belongs to image 1.
If $D_{12}<Th_1$ & $D_{23}>Th_2$ & $D_{24}<Th_2$, then X belongs to image 2.
If $D_{23}<Th_2$ & $D_{34}>Th_3$ & $D_{13}<Th_3$, then Y belongs to image 3.
If $D_{34}<Th_3$ & $D_{24}<Th_3$ & $D_{14}<Th_3$, then Z belongs to image 4.

where,

D_{12} is the discriminant function between the images 1 and 2. Similarly D_{24} is the discriminant function between the images 2 and 4 and so on. Th_1, Th_2, and Th_3 are the thresholds between the images 1, 2, 3 and 4 as shown in the Figure 4.4-3. A threshold of 0.0 was chosen between straw and grass, a threshold of -1.0 was chosen between grass and raffia and finally a threshold of -2.0 was chosen between raffia and paper.

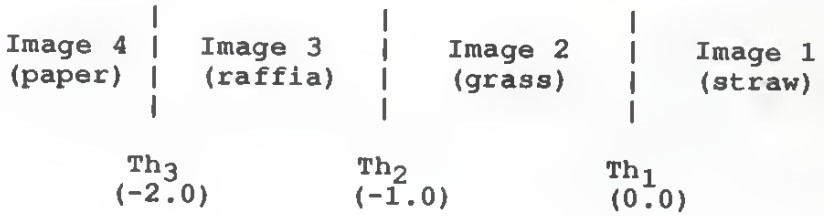


Figure 4.4-3 Different images and their threshold values

For the number of texture patterns greater than two, threshold devices that may be required are given by $2^M > m$,

where M is the number of threshold devices and m is the number of patterns. For example, to classify eight pattern classes three threshold devices may be sufficient.

4.5 TRAINING IN LINEAR CLASSIFIERS

If two textural patterns are linearly separable, then by choosing the correct values of the weights a perfect recognition can be achieved. Choosing the correct weights can be done by training the classifier. Training a classifier is done by adjusting the weights with a known texture classification.

Let $Y = y_1, y_2, \ldots, y_n$ be a feature vector of a texture pattern. Augmented feature vector X and weight vector W are shown in Figure 4.5-1.

$$X = \begin{vmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \\ 1 \end{vmatrix} \qquad W = \begin{vmatrix} w_1 \\ w_2 \\ . \\ . \\ w_n \\ w_{n+1} \end{vmatrix}$$

Figure 4.5-1 Augmented feature vector and the weight vector.

If 1 and 2 are the pattern classes which are linearly separable, then a solution weight vector exists such that if $X^T W\ (D_{12}) > Th_1$, then X belongs to 1 and if $X^T W\ (D_{12}) < Th_1$, then X belongs to 2.

The training is done by the following procedure. A

data set with the known classification is fed to the classifier. If there is no error in classification, then for the image belonging to 1, the product X^TW should be greater than Th_1. If $X^TW < Th_1$ for the image belonging to 1, the weight vector is adjusted such that

$$W_{new} = W_{old} + (alpha * X) \qquad (4.5\text{-}1)$$

On the other hand, if the product $X^TW > Th_1$ for the image belonging to two, then the new weight vector would be

$$W_{new} = W_{old} - (alpha * X) \qquad (4.5\text{-}2)$$

where alpha is any fixed positive number.

In this study an alpha value of 0.1 was chosen and the classifier was trained for threshold values of 0.0, -1.0, and -2.0.

4.6 IMAGE CLASSIFICATION

A four class classifier shown in the Figure 4.5-2 was used to classify raffia, grass, straw and paper. The input to the classifier consisted of five textural features computed from distance 1 spatial gray level dependence matrices. The data sets were obtained by creating 4 windows of size 32x32 from original 256x256 images. The average set computed from four data sets was used as a training set and the four data sets were used as test samples. The contingency table or confusion matrix for the classification of test samples is shown in the Fig 4.6-1.

The straw and grass showed 75% accuracy in classification which was better compared to the paper and raffia which showed 50% and 25% respectively. Overall classification accuracy on the data set which was obtained by adding all the diagonal elements of the contingency table and dividing by the total number of samples was found to be approximately 57%.

		Assigned category				
		straw	grass	raffia	paper	total
True category	straw	3	1	0	0	4
	grass	1	3	0	0	4
	raffia	1	1	1	1	4
	paper	1	1	0	2	4
	total	6	6	1	3	16

Figure 4.6-1 Contingency table for the classification of data set.

The obvious reason for a moderate degree of accuracy is that the classifier was trained with only one data set, which resulted in improper weights. By choosing more training sets, a classifier can be trained better. Hence the conclusion is that scatter plots, inertia plots and the classifier were useful in classifying only certain images and do not give good classification for all images.

is that the classifier was trained with only one data set, which resulted in improper weights. By choosing more training sets, a classifier can be trained better. Hence the conclusion is that scatter plots, inertia plots and the classifier were useful in classifying only certain images and do not give good classification for all images.

CHAPTER 5: SUMMARY AND CONCLUSIONS

The primary purpose of this project was to compute and test the classification power of the cooccurrence matrices using the popular technique, Spatial Gray Level Dependence Method (SGLDM). Various textural features like inertia, entropy, homogeneity, contrast and correlation were computed using cooccurrence matrices. The textures used for this study included raffia, wood, wire, french canvas, paper, grass, cork, and straw. The cooccurrence matrices and textural features were computed from a reduced image.

Programs were written in "C" to compute cooccurrence matrices and textural features. Scatter plots and inertia plots were plotted and a linear classifier was used to classify the textures.

Two-dimensional scatter plots, which are the plots of one feature against the other were made use of in this study. A good clustering of some of the images occurred, which resulted in a good classification of those images. Some of the images like raffia, grass, straw and paper could not be very well classified. The scatter plots showed a fairly good classification power and could classify four to five textures successfully.

Since the images like raffia, grass and straw could not be very well classified, plots of horizontal and vertical inertia against intersampling distance d were obtained. Horizontal and vertical inertia were computed along 0 and 90 degrees with the intersampling distance d = 1 to 40. Results showed that the inertia plots could be used to detect periodic property of the textures. The textures raffia and canvas which exhibited different types of periodicities were very well distinguished from grass and straw which did not exhibit any periodic property.

The final phase of the study included texture classification using a linear classifier which makes use of all the features. The test samples used for this study were raffia, grass, straw and paper. Straw and grass showed a fairly good classification with an accuracy of 75%. Overall classification accuracy on the entire data set was found to be 57 per cent.

CHAPTER 6: REFERENCES

1. Connors, R.W., and Harlow, C.A. [1979]. "Towards a Structural Textural Analyser based on Statistical Methods". Remote Sensing and Image Processing Laboratory, Louisiana State University, Louisiana.

2. Connors, R.W., Trivedi, M.M. and Harlow, C.A. [1984]. "Segmentation of a High-Resolution Urban Scene Using Texture Operators." Computer Vision, Graphics and Image Processing, vol.25, pp.273-310.

3. Davis, L.S., Johns, S.A. and Aggarwal, J.K. [1979]. "Texture Analysis Using Generalized Cooccurrence Matrices." I.E.E.E. Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 3, pp. 251-259.

4. De Souza, P. [1982]. "Texture Recognition via Auto Regression." Pattern Recognition, vol.6, pp.471-475

5. Galloway, M.M., [1975]. "Texture Analysis Using Gray Level Run Lengths." Computer Graphics and Image Processing, vol. 4, pp. 172-179.

6. Fu, K.S. [1982]. "Applications of Pattern Recognition." CRC Press Inc., Florida.

7. Haralick, R.M., Shanmugam, K. and Dinstein, I. [1973]. "Textural Analysis for Image Classification." I.E.E.E. Transactions on Systems, Man and Cybernetics, vol.6, pp.610-621.

8. Jain, A.K. [1989]. "Fundamentals of Digital Image Processing." Prentice-Hall, Inc., New Jersey.

9. Kaneko, H. and Yodogawa, E. [1982]. "A Markov Random Field Application to Texture Classification." Proc. Pattern Recognition and Image Processing. pp.221-225.

10. Kashyap, R.L., Chellappa, R. and Khotanzad, A. [1982]. "Texture Classification Using Features Derived from Random Field Models." Pattern Recognition Lett.1, pp.43-50.

11. Sun, C., and Wee, W.G. [1983]. "Neighbouring Gray Level Dependence Matrix for Texture Classification." Computer Vision, Graphics and Image Processing, vol.23, pp.341-352.

12. Van Gool, L., Dewaele, P. and Oosterlinck, A. [1985]. "Survey, Texture Analysis Anno 1983." Computer Vision, Graphics and Image Processing, vol.29, pp.336-357.

APPENDIX A: EQUIPMENT LIST

1. Digital Equipment Corporation VAX 11/750 digital computer
2. Grinnell GMR 270 Series Image Processing System
3. Mitsubishi Model #C3922 LPK Color Television Monitor
4. Hewlett-Packard 7475A plotter
5. Olympus 35mm Camera
6. Ilford 35mm film, ASA 100

APPENDIX B: Computer Listings

This appendix includes the computer listings of some of the important routines written for this study. These routines are written to compute cooccurrence matrices, textural features and also includes a classifier program for textural classification. These computer listings include only the routines which are relevant to the report and plotting routines which plotted scatter plots and inertia plots are excluded.

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *              Kansas State University
 *          VAX C Source file name : co_occ_matrix.c
 *****************************************************************
 *
 *
 *   FUNCTION:              main()
 *
 *
 *   DESCRIPTION:           The following program opens a image
 *                          file and reads the image. The
 *                          cooccurrence matrices are computed
 *                          from the window of user's choice and
 *                          the different features like inertia,
 *                          entropy, contrast,correlation and
 *                          homogeneity are computed.
 *
 *
 *  DOCUMENTATION           none.
 *  FILES:
 *
 *
 *  ARGUMENTS:              none.
 *
 *
 *  RETURN:                 none.
 *
 *
 *  FUNCTIONS
 *  CALLED:                 homo_measure();
 *                          corr();
 *                          inertia();
 *                          entropy();
 *                          contrast()
 *
 *
 *  AUTHOR:                 D. K. Durga
 *
 *
 *  DATE CREATED:           September 20, 1988      version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
*****************************************************************/

#include <stdio.h>

#define     cr          0x0d
#define     space       0x20
#define     SIZE        32
#define     MAX         256
```

```
#define    MAXIMUM   1024

main ()
{

extern double homo_measure(),contrast(), entropy(),corr(),
            inertia();

/*-------------------------------------------------------------*
 *  Variable declarations.                                     *
 *-------------------------------------------------------------*/

int   i,                        /* General purpose loop counters */
      j,
      k,
     val1,                      /* First gray level from matrix  */
     val2,                      /* Second gray level from matrix */
     l,                         /* Dummy variables.              */
     m,
     c,
     dummy,
     window,                    /* Number of the window.         */
     init,                      /* Window first x_coordinate     */
     final,                     /* Window first y_coordinate     */
     first,                     /* Window last x_coordinate      */
     last;                      /* Window last y_coordinate.     */

/*-------------------------------------------------------------*
 *  Array declarations.                                        *
 *-------------------------------------------------------------*/

int gray_level [MAXIMUM];       /* 1_d image array.              */
    image_array [MAX][MAX];     /* 2_d image array.              */
    co_occ_matrix [SIZE][SIZE],/* Coooccurrence matrix           */
    occ_matrix [SIZE][SIZE];

double homogen,                 /* Homogeneity of the image      */
       ent,                     /* Entropy of the image.         */
       con,                     /* Contrast of the image.        */
       cor,                     /* Correlation of the image.     */
       iner;                    /* Inertia of the image.         */

char file_name [SIZE];
char fname[20];

FILE *fpr;                      /* Pointer to the file           */
FILE *fp;

/*-------------------------------------------------------------*
 *  Prompt the user to enter the file name and read the file   *
 *  name.                                                      *
 *-------------------------------------------------------------*/
   printf ("\n Please enter the file name ");
   scanf ("%s", file_name);
```

```
/*-------------------------------------------------------------*
 *  Open the file of user's choice and if any error occurs     *
 *  while opening the file print the error message and exit.   *
 *-------------------------------------------------------------*/
    if (( fpr = fopen (file_name, "r")) == NULL)
        {
        printf ("\n ERROR opening the file %s", file_name);
        exit (0);
        }
/*-------------------------------------------------------------*
 * Prompt the user to enter the number of the window to be     *
 * processed. The switch statement chooses the size of the     *
 * window.                                                     *
 *-------------------------------------------------------------*/
    printf("Please enter the number(size) of the window");
    scanf("%d", &window);

    switch (window)
        {
        case 1 : init = 32;
                 final = 64;
                 first = 32;
                 last = 64;
                 break;
        case 2 : init = 32;
                 final = 64;
                 first = 192;
                 last = 224;
                 break;
        case 3 : init = 192;
                 final = 224;
                 first = 32;
                 last = 64;
                 break;
        case 4 : init = 192;
                 final = 224;
                 first = 192;
                 last = 224;
                 break;
        }
/*-------------------------------------------------------------*
 *  Initialize the image array to zeros.                       *
 *-------------------------------------------------------------*/
    for (i = 0; i < MAX; ++i)
        {
        for (j = 0; j < MAX; ++j)
            {
            image_array [i][j] = 0;
            }
        }
/*-------------------------------------------------------------*
 *  Read the integers of the file into the image array one     *
 *  after the other leaving out all the newline characters     *
 *  and carriage return characters.                            *
```

```
/*-------------------------------------------------------------------*/
    k = 3;
    for (i = 0; i < MAX; ++i)
        {
        for (j = 0; j < MAX; ++j)
            {
            while (k != 0)
                {
                c = fgetc (fptr);
                if (( c != '\n') && (c != cr))
                    {
                    if (c == space)
                        c -= 0x20;
                    else
                        c -= 0x30;
                    if (k != 1)
                        {
                        dummy = k-1;
                        while (dummy != 0)
                            {
                            c = c * 10;
                            dummy--;
                            }
                        }
                        image_array [i][j] += c;
                        k--;
                        }
                    }
                k = 3;
            }
        }

/*-------------------------------------------------------------------*
 * Set all the elements of occ matrix and cooccurence matrix         *
 * equal to zero.                                                    *
 *-------------------------------------------------------------------*/
   for (i =0; i < SIZE; ++i)
        {
        for (j = 0; j < SIZE; ++j)
            {
            occ_matrix[i][j] = 0;
            co_occ_matrix[i][j] = 0;
            }
        }
/*-------------------------------------------------------------------*
 * Initilize gray_level matrix.                                      *
 *-------------------------------------------------------------------*/
   for (i = 0; i < MAXIMUM; ++i)
        {
        gray_level [i] = 0;
        }
/*-------------------------------------------------------------------*
 *  Read the gray levels of two dimensional array into one           *
 *  dimensional gray level array and scale the gray levels.          *
```

```
/*-------- - ---------------------------- - ----------------------------*/
   k = 0;
   for (i = init; i <= final; ++i)
       {
       for (j = first; j < last; ++j)
           {
           gray_level[k] = image_array [i][j]/8;
           k++;
           }
       }
/*-------------------------------------------------------------------*
 *  Read in the values of two neighbouring pixels and if any         *
 *  two pixels have same neighbouring values then increase the       *
 *  corresponding element of occurrence matrix by one.               *
 *-------------------------------------------------------------------*/
   for (i = 0; i < SIZE; ++i)
      {
      for (j = 0; j < SIZE; ++j)
         {
         l = 0;
         for (m = 0; m < MAXIMUM; m = m + 32)
           {
           for (k = m; k < (m + 31); ++k)
              {
               val1 = gray_level[l];
               l++;
               val2 = gray_level[l];
               if ((i == val1) && (j == val2))
                  {
                  occ_matrix[i][j] = occ_matrix[i][j] + 1;
                   }
                if ((i == val2) && (j == val1))
                   {
                   occ_matrix[i][j] = occ_matrix[i][j] + 1;
                    }
                }
               l++;
             }
         }
       }

/*-------------------------------------------------------------------*
 *  Transfer all the elements of occ_matrix into cooccurrence        *
 *  matrix.                                                          *
 *-------------------------------------------------------------------*/
  for (i = 0; i < SIZE; ++i)
       {
       for (j = 0; j < SIZE; ++j)
           {
           co_occ_matrix[i][j] = occ_matrix[i][j];
           }
       }
/*-------------------------------------------------------------------*
 * Call the function homo_measure to measure the homogeneity        *
```

```
 * of the image.                                                    *
 *------ ----- --- - ---------- ------------------------------------*/
   homosen =  homo_measure(co_occ_matrix);
   printf("The homoseneity is = %lf", homosen);
/*-------------------------------------- ---------- ----------*
 * Call the function entropy to measure the entropy of the      *
 * image.                                                       *
 *------ - --- ------------------------------------------------*/
   ent = entropy(co_occ_matrix);
   printf ("The entropy is = %lf", ent);
/*-------------------------------------------------------------*
 * Call the function inertia to measure inertia of the image.   *
 *-------------------------------------------------------------*/
   iner = inertia(co_occ_matrix);
   printf("The inertia of the image = %lf",iner);
/*------ ---- ---------- - - ----------*
 * Call the function contrast to measure contrast.              *
 *------- - - ---------- ---------------*/
   con = contrast(co_occ_matrix);
   printf("The contrast of the image = %lf", con);
/*-------------------------------------------------------------*
 * Call the function corr to measure the correlation.           *
 *---------- ---------- ---------- ---------------------------*/
   cor = corr(co_occ_matrix);
   printf("The correlation of the image = %lf", cor);
/* ------ - - - ---------- ---------- ---------- ---------------*
 * End of program.                                              *
 *------ ---- ---------- - - ----------------------------------*/
   return(0);
}
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *               Kansas State University
 *          VAX C Source file name : More.iner.c
 *****************************************************************
 *
 *
 *  FUNCTION:             main()
 *
 *
 *  DESCRIPTION:          The following function computes inertia
 *                        for different intersampling distances
 *                        d = 1 to 45 and uses simple plot to
 *                        plot inertia vs. distance.
 *
 *
 *  DOCUMENTATION         none
 *  FILES:
 *
 *
 *  ARGUMENTS:            none
 *
 *
 *  RETURN:               none
 *
 *
 *  FUNCTIONS             simple_plot()
 *  CALLED:
 *
 *
 *  AUTHOR:               D. K. Durga
 *
 *
 *  DATE CREATED:         November 10, 1988      version 1.00
 *
 *
 *  REVISIONS:            none
 *
 *
 *****************************************************************/

#include <stdio.h>

#define    cr         0x0d
#define    space      0x20
#define    SIZE       45
#define    MAX        256
#define    MAXIMUM    4050

main ()
{
 extern simple_plot();
/*-------------------------------------------------------------*/
/*  Variable declarations.                                     */
```

```
/*-------------------------------------------------------------*/

int  i,                          /* General purpose loop counters  */
     j,
     k,
     plot_type = 1,              /* Line plot for cmplx_plot.      */
     length = 90,                /* Image size.                    */
     value,                      /* Dummy variable                 */
     val1,                       /* First gray level of the matrix */
     val2,                       /* Second gray level of the matrix*/
     l,                          /* Dummy variables                */
     m,
     c,
     dummy,
     window,                     /* Number of the window.          */
     init,                       /* Window first x_coordinate.     */
     final,                      /* Window first y_coordinate.     */
     first,                      /* Window last x_coordinate.      */
     last,                       /* Window last Y_coordinate.      */
     n= 1 : 15,                  /* Total number of gray levels    */
     dist;                       /* Intersampling distance.        */
/*-------------------------------------------------------------*
 #  Array declarations.                                        *
 *-------------------------------------------------------------*/
int gray_level [MAXINUM],        /* 1-d image array.               */
    image_array [MAX][MAX],      /* 2-d image array                */
    co_occ_matrix[SIZE][SIZE],/* Cooccurrence matrix.              */
    occ_matrix [SIZE][SIZE];

double inertia[SIZE],            /* Inertia array                  */
       distance[SIZE];           /* Intersampling distance array   */

double iner,                     /* Inertia to be computed.        */
       r,                        /* Normalization constant         */
       nx = 45,                  /* Resolution cells in x_dir      */
       ny = 45;                  /* Resolution cells in y_dir.     */
char file_name [SIZE];           /* File to be opened.             */
char fname[20];
char xtitle[MAX],                /* X title array                  */
     ytitle[MAX],                /* Y title array                  */
     xunits[MAX],                /* X axis units array             */
     yunits[MAX],                /* Y axis units array             */
     plot_title[MAX];            /* Plot title array.              */

FILE *fpr;                       /* Pointer to file.               */
FILE *fp;

/*-------------------------------------------------------------*
 * Promrt the user to enter the file name and read the file    *
 * name.                                                       *
 *-------------------------------------------------------------*/
   printf ("\n Please enter the file name ");
   scanf ("%s", file_name);
/*-------------------------------------------------------------*
```

```
    * Open the file of user's choice and if any error occurs      *
    * while opening the file print the error message and exit.    *
    *-------------------------------------------------------------*/
      if (( fpr = fopen (file_name, "r")) == NULL)
         {
          printf ("\n ERROR opening the file %s", file_name);
          exit (0);
          }
    /*-------------------------------------------------------------*
     * Set the size of the window to 45 x 90.                      *
     *-------------------------------------------------------------*/
      init = 0;
      final = 44;
      first = 0;
      last = 89;
    /*-------------------------------------------------------------*
     * Initialize the image array to zeros.                        *
     *-------------------------------------------------------------*/
      for (i = 0; i < MAX; ++i)
          {
          for (j = 0; j < MAX; ++j)
              {
              image_array [i][j] = 0;
              }
          }
    /*-------------------------------------------------------------*
     * Read the integers of the file into the image array one      *
     * after the other leaving out all the newline characters      *
     * and carride return characters.                              *
     *-------------------------------------------------------------*/
      k = 3;
       for (i = 0; i < MAX; ++i)
           {
           for (j = 0; j < MAX; ++j)
               {
               while (k != 0)
                   {
                   c = fgetc (fpr);
                   if (( c != '\n') && (c != cr))
                       {
                       if (c == space)
                           c -= 0x20;
                       else
                           c -= 0x30;
                       if (k != 1)
                           {
                           dummy = k-1;
                           while (dummy != 0)
                               {
                               c = c * 10;
                               dummy--;
                               }
                           }
                           image_array [i][j] += c;
```

```
                        k--;
                        }
                    }
                k = 3;
            }
        }

/*-------------------------------------------------------------------*
 * Set all the elements of occ matrix, cooccurrence matrix,          *
 * inertia and distance arrays equal to zero.                        *
 *-------------------------------------------------------------------*/
   for (i =0; i < SIZE; ++i)
       {
       for (j = 0; j < SIZE; ++j)
           {
           occ_matrix[i][j] = 0;
           cu occ_matrix[i][j] = 0;
           }
       }
    for (i=0; i< SIZE;++i)
        {
        inertia[i] = 0.0;
        distance[i]=0.0;
        }
/*-------------------------------------------------------------------*
 * Initilize gray level matrix.                                      *
 *-------------------------------------------------------------------*/
   for (i = 0; i < MAXIMUM; ++i)
       {
       gray_level [i] = 0;
       }
/*-------------------------------------------------------------------*
 * Read the gray levels of two dimensional array into one            *
 * dimensional gray level array and scale the gray levels.           *
 *-------------------------------------------------------------------*/
    k = 0;
    for (i = init; i < final; ++i)
        {
        for (j = first; j < last; ++j)
            {
            gray_level[k] = (int)image.array [i][j]/5.60888888;
            k++;
            }
        }
/*-------------------------------------------------------------------*
 * Computation of r.                                                 *
 *-------------------------------------------------------------------*/
      r = 2 * nx * (ny - 1);
/*-------------------------------------------------------------------*
 * Read in the values of two neighbouring pixels and if any          *
 * two pixels have same neighbouring values then increase the        *
 * corresponding element of occurrence matrix by one and repeat      *
 * this process for d = 1 to 45.                                     *
 *-------------------------------------------------------------------*/
```

```
    value = 45;
    for (dist = 1; dist <=44; ++dist)
     {
     for (i = 0; i < SIZE; ++i)
        {
        for (j = 0; j < SIZE; ++j)
           {
           l = 0;
           for (m = 0; m < MAXIMUM; m = m + length)
              {
              for (k = m; k < (m +value); ++k)
                 {
                  val1 = gray_level[l];
                  l = (l+dist);
                  val2 = gray_level[l];
                  if ((i == val1) && (j == val2))
                     {
                      occ_matrix[i][j] = occ_matrix[i][j] + 1;
                      }
                   if ((i == val2) && (j == val1))
                      {
                      occ_matrix[i][j] = occ_matrix[i][j] + 1;
                      }
                    l = ((l -dist) + 1);
                    }
                    l=l+(length -value);
              }
           }
        }
/*-------------------------- ------------------------------------*
 *  Transfer all the elements of occ_matrix into co_occurance   *
 *  matrix.                                                     *
 *------ -- ---------------------------------------------------*/
     for (i = 0; i < SIZE; ++i)
        {
        for (j = 0; j < SIZE; ++j)
           {
           co_occ_matrix[i][j] = occ_matrix[i][j];
            }
         }
/*-------------------------------------------------------------*
 * Compute the inertia for each of the sampling distances and  *
 * fill the inertia array.                                     *
 * ----------------------------------------------------------*/
     iner = 0.0;
     for (i = 0; i < ng_1; ++i)
         {
         for (j = 0; j < ng_1; ++j)
             {
             iner += ((co_occ_matrix[i][j]/r) * (i-j) *
                      (i-j));
              }
          }
     inertia[dist] = iner/10.0;
```

```
      distance[dist] = dist;
/*-------------------------------------------------------------*
 * Reinitialize occurrence and concurrence matrices.           *
 *-------------------------------------------------------------*/
       for (i=0; i <SIZE;++i)
           {
           for(j = 0; j <SIZE; ++j)
               {
                co_occ_matrix[i][j] =0;
                occ_matrix[i][j] = 0;
                }
            }
    }
/*-------------------------------------------------------------*
 * Call the simple plot to plot inertia vs. intersampling      *
 * distance.                                                   *
 *-------------------------------------------------------------*/
   strcpy (xtitle, 'Intersampling space d');
   strcpy(ytitle, 'Var Inertia');
   strcpy(xunits, 'no_units');
   strcpy(yunits, 'no_units');
   strcpy(plot_title, 'Inertia vs. Distance(angle = 0 deg)');
   simple_plot(15,distance,inertia,xtitle,ytitle,xunits,yunits,
               plot_type, plot_title);
/*-------------------------------------------------------------*
 * End of program.                                             *
 *-------------------------------------------------------------*/
   return(0);
 }
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                Kansas State University
 *           VAX C Source file name : homogeneity.c
 *****************************************************************
 *
 *
 *  FUNCTION:               homo_measure ()
 *
 *
 *  DESCRIPTION:            The following function computes
 *                          homogeneity from the cooccurrence
 *                          matrices.
 *
 *
 *  DOCUMENTATION           none
 *  FILES:
 *
 *
 *  ARGUMENTS:
 *      co_occ_matrix       (input) integer
 *                          cooccurrence matrix
 *
 *
 *  RETURN:
 *     homo                 (output) double
 *                          This function returns computed
 *                          value of homogeneity to the main.
 *
 *
 *  AUTHOR:                 D. K. Durga
 *
 *
 *  DATE CREATED:           September 30, 1988      version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
*****************************************************************/

#include <stdio.h>
#include <math.h>

#define    SIZE    32


double homo_measure(co_occ_matrix)

int co_occ_matrix[SIZE][SIZE];

{

int ng_1 = 32;              /* Total number of grey levels.   */
```

```
int i,                          /* General purpose loop counters. */
    j;

double val = 2.0,               /* Power of the pow function.     */
       homo,                    /* Texture feature to be computed */
       r,                       /* Value for the normalization of */
                                /* matrix.                        */
       nx = 32.0,               /* No. of gray levels in x dir.   */
       ny = 32.0;               /* No. of gray levels in y dir.   */

double pow ();

/*-------------------------------------------------------------*
 * The computation of R for normalizating the co_occ_matrix. *
 *-------------------------------------------------------------*/
       r = 2 * nx * (ny - 1);
/*-------------------------------------------------------------*
 * Calculation of homoseneity measure.                         *
 *-------------------------------------------------------------*/
       homo = 0.0;
       for (i = 0; i < ng_1; ++i)
           {
           for (j = 0; j < ng_1; ++j)
               {
                homo += pow ((double) co_occ_matrix[i][j]/r,
                             val);
               }
           }
/*-------------------------------------------------------------*
 * Return homoseneity to the main function.                    *
 *-------------------------------------------------------------*/

       return (homo);
  }
```

```
/*************************************************************************
 *      Department of Electrical and Computer Engineering
 *                Kansas State University
 *           VAX C Source file name : Contrast.c
 *************************************************************************
 *
 *
 *  FUNCTION:               contrast ()
 *
 *
 *  DESCRIPTION:            The following function computes
 *                          contrast from the cooccurrence matrices.
 *
 *
 *  DOCUMENTATION           none.
 *  FILES:
 *
 *
 *  ARGUMENTS:
 *      co_occ_matrix       (input)  double
 *                          cooccurrence matrix.
 *
 *
 *  RETURN:
 *      contrast            (output) double
 *                          This function returns computed value
 *                          of contrast to the main.
 *
 *
 *  AUTHOR:                 D. N. Duraa
 *
 *
 *  DATE CREATED:           September 12, 1988    version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
*************************************************************************/

#include <stdio.h>
#include <math.h>

#define   SIZE  32
#define   MAX   10000

double contrast(co_occ_matrix)

int co_occ_matrix[SIZE][SIZE];

{
int nd_1 = 32;                  /* Total number of gray levels   */

int i;                          /* General purpose loop counters */
```

```
    j;
    k;

double contrast;              /* Feature to be computed          */
       sum;                   /* Sum of coocurrence matrix       */
       r;                     /* Value for the normalization of*/
                              /* matrix.                         */
       nx = 32.0;             /* No of gray values in x dir.     */
       ny = 32.0;             /* No of gray values in y dir.     */

/*-------------------------------------------------------------*
 * Computation of r for normalization of co_occ_matrix.       *
 *-------------------- ------------------ --------------------*/

    r = 2.0 * nx * (ny - 1);

/*-------------------------------- ---------------------------*
 * Calculation of contrast.                                   *
 *-------------------------------------- -- ------ ----------*/

    contrast = 0.0;

    for (i = 0; i < ng_1; ++i)
        {
        for (j = 0; j < ng_1; ++j)
            {
            contrast += (abs(i-j) * co_occ_matrix[i][j]/r);
            }
        }

    return (contrast);
}
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *                Kansas State University
 *           VAX C Source file name : Inertia.c
 *****************************************************************
 *
 *
 *  FUNCTION:               inertia ()
 *
 *
 *  DESCRIPTION:            The following function computes
 *                          inertia from the cooccurrence
 *                          matrices.
 *
 *
 *  DOCUMENTATION           none.
 *  FILES:
 *
 *
 *  ARGUMENTS:
 *      co_occ_matrix       (input)  integer
 *
 *
 *  RETURN:
 *      inertia             (output) double
 *                          This function returns the computed
 *                          value of inertia to the main.
 *
 *
 *  AUTHOR:                 D. K. Durga
 *
 *
 *  DATE CREATED:           october 17, 1988      version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
*****************************************************************/

#include <stdio.h>
#include <math.h>

#define    SIZE    32

double inertia(co_occ_matrix)

int co_occ_matrix[SIZE][SIZE];

{

int no_1 = 32,              /* Total number of gray levels.   */
    i,                      /* General purpose loop counters  */
    j;
```

```
double val = 2.0,              /* Power of the pow function.      */
       inertia,                /* Texture feature to be computed */
       r,                      /* Value to normalize the matrix. */
       nx = 32.0,              /* No. of gray levels in the x dir*/
       ny= 32.0;               /* No. of gray levels in the y dir*/

double pow ();

/*-------------------------------------------------------------*
 * The computation of R for normalizating the co_occ_matrix.  *
 *-------------------------------------------------------------*/
     r = 2 * nx * (ny - 1);
/*-------------------------------------------------------------*
 * Calculation of inertia measure.                             *
 *-------------------------------------------------------------*/
     inertia = 0.0;

     for (i = 0; i < ng_1; ++i)
         {
         for (j = 0; j < ng_1; ++j)
             {
             inertia +=((co_occ_matrix[i][j]/r) * ((i-j)
                        * (i-j)));
             }
         }
/*-------------------------------------------------------------*
 * Return inertia to the main function.                        *
 *-------------------------------------------------------------*/
     return (inertia);
 }
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *              Kansas State University
 *          VAX C Source file name : Entropy.c
 *****************************************************************
 *
 *
 *  FUNCTION:           entropy ()
 *
 *
 *  DESCRIPTION:        The following function computes entropy
 *                      from the cooccurrence matrices.
 *
 *
 *  DOCUMENTATION       none.
 *  FILES:
 *
 *
 *  ARGUMENTS:
 *      co_occ_matrix   (input)  int
 *
 *
 *  RETURN:
 *      ent             (output)  double
 *                      This function returns computed value
 *                      of entropy to the main.
 *
 *
 *  AUTHOR:              D. K. Durga
 *
 *
 *  DATE CREATED:        September 13, 1988     version 1.00
 *
 *
 *  REVISIONS:           none.
 *
 *
*****************************************************************/

#include <stdio.h>
#include <math.h>

#define   SIZE  32
#define   MAX   10000

double entropy(co_occ matrix)

int co_occ_matrix[SIZE][SIZE];

{
int ng_1 = 32,                  /* Total number of gray levels  */
    i,                          /* General purpose loop counters*/
    j,
    k;
```

```
double entropy,                 /*Texture feature to be computed*/
       sum,                     /* Sum of cooccurrence matrix    */
       ent,                     /* Negative value of entropy.    */
       r,                       /* Value for the normalization   */
                                /* of matrix.                    */
       nx = 32.0,               /* No. of gray levels in x dir. */
       ny = 32.0;               /* No. of gray levels in y dir. */

/*-------- ---------- ---- ----------*
 * Computation of r for normalization of co_occ_matrix.      *
 *---------- ---------- ----------*/

       r = 2.0 * nx * (ny - 1);

/*---------- ---------- ----------*
 * Calculation of Entropy.                                   *
 *---------- ---------- ----------*/

       entropy = 0.0;

       for (i = 0; i < ns_1; ++i)
           {
           for (j = 0; j < ns_1; ++j)
               {
                entropy += ((co_occ_matrix[i][j]/r)*
                            log10(co_occ_matrix[i][j]/r));
               }
            }

        ent = ((-1) * entropy);
        return (ent);
}
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *              Kansas State University
 *          VAX C Source file name : Correlation.c
 *****************************************************************
 *
 *
 *  FUNCTION:               corr()
 *
 *
 *  DESCRIPTION:            The fallowing program computes one of
 *                          the features, correlation from the
 *                          cooccurrence matrices.
 *
 *
 *  DOCUMENTATION           none.
 *  FILES:
 *
 *
 *  AGUMENTS:
 *      co_occ matrix       (input)  integer
 *                          cooccurrence matrix.
 *
 *
 *  RETURN:
 *       corr               (output)  double
 *                          This function returns computed value
 *                          of correlation to the main.
 *
 *
 *  AUTHOR:                 D. K. Durga
 *
 *
 *  DATE CREATED:           September 20, 1988    version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
 *****************************************************************/

#include <stdio.h>
#include <math.h>

#define    SIZE    32

double corr(co_occ_matrix)

int co_occ_matrix[SIZE][SIZE];

{
extern double power();

int ng_1 = 32;              /* Total number of gray levels.   */
```

```
int i,                          /* General purpose loop counters. */
    j;

double corr,                    /* Texture feature to be computed */
       r,                       /* Value for normalization of the */
                                /* matrix.                        */
       nx = 32.0,               /* No of gray levels in x dir.    */
       ny= 32.0,                /* No of gray levels in y dir.    */
       x_mean,                  /* Mean value of x.               */
       y_mean,                  /* Mean value of y.               */
       x_var,                   /* Variance of x.                 */
       y_var,                   /* Variance of y.                 */
       total,                   /* Total of mean deviations.      */
       xy_sigma,                /* Product of sqrt of variances.  */
       sum;                     /* Sum of cooccurrence matrix.    */

/*-------------------------------------------------------------*
 * The computation of R for normalization of the co_occ.matrix.*
 *-------------------------------------------------------------*/
    r = 2 * nx * (ny - 1);
/*-------------------------------------------------------------*
 * Calculation of x_mean.                                      *
 *-------------------------------------------------------------*/
    x_mean = 0.0;
    sum = 0.0;
    for (i = 0; i < ng_1; ++i)
        {
        for (j = 0; j < ng_1; ++j)
            {
            sum += co_occ_matrix[i][j]/r;
            }
         x_mean += i * sum;
        }
/*-------------------------------------------------------------*
 * Calculation of y_mean.                                      *
 *-------------------------------------------------------------*/
    y_mean = 0.0;
    sum = 0.0;
    for (j = 0; j < ng_1; ++j)
        {
        for (i = 0; i < ng_1; ++i)
            {
            sum += co_occ_matrix[i][j]/r;
            }
         y_mean += j * sum;
        }
/*-------------------------------------------------------------*
 * Calculation of x_var.                                       *
 *-------------------------------------------------------------*/
    x_var = 0.0;
    sum = 0.0;
    for (i = 0; i < ng_1; ++i)
        {
```

```
            for (j = 0; j < ng_1; ++j)
                {
                sum += co_occ_matrix[i][j]/r;
                }
             x_var += (i - x_mean) * (j - y_mean) * sum;
            }
/*-------------------------------------------------------------*
 * Calculation of y_var.                                       *
 *-------------------------------------------------------------*/
       y_var = 0.0;
       sum = 0.0;
       for (j = 0; j < ng_1; ++j)
            {
            for (i = 0; i < ng_1; ++i)
                {
                sum += co_occ_matrix[i][j] / r;
                }
            y_var += (j - y_mean)*(j - y_mean) * sum;
            }
/*-------------------------------------------------------------*
 * Calculation of correlation.                                 *
 *-------------------------------------------------------------*/
     total = 0.0;
     for (i = 0; i < ng_1; ++i)
         {
         for (j = 0; j < ng_1; ++j)
             {
             total += ((i - x_mean) * (j - y_mean) *
                         co_occ_matrix[i][j]/r);
             }
         }
     xy_sigma = sqrt(x_var) * sqrt(y_var);
     corr = total / xy_sigma;
/*-------------------------------------------------------------*
 * Return correlation to the main function.                    *
 *-------------------------------------------------------------*/

     return (corr);

 }
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *               Kansas State University
 *          VAX C Source file name : Welsh.c
 *****************************************************************
 *
 *
 *  FUNCTION:                main ()
 *
 *
 *  DESCRIPTION:             The following program computes the
 *                           appropriate weights for different
 *                           features.
 *
 *
 *  DOCUMENTATION
 *  FILES:                   none.
 *
 *
 *  ARGUMENTS:               none.
 *
 *
 *  RETURN:                  none.
 *
 *
 *  FUNCTIONS
 *  CALLED:                  none.
 *
 *
 *  AUTHOR:                  B. K. Durga
 *
 *
 *  DATE CREATED:            November 23, 1988   version 1.00
 *
 *
 *  REVISIONS:               none.
 *
 *
*****************************************************************/

#include <stdio.h>

#define  MAX  40
#define  NUM  6
#define  LEN  24
#define  YES  1
#define  NO   0

main ()
{
/*-------------------------------------------------------------*
 * Declaration of variables.                                   *
 *-------------------------------------------------------------*/
 int m=0;                    /* General purpose loop counters.  */
```

```
    i,
    j,
    l,
    l=0,
    answer=1;                   /* Boolean variable for weights.   */

 double ws1,                    /* Sum of the feature vectors for  */
        ws2,                    /* images one and two.             */
        alpha =0.1,             /* Alpha for adjustment of weights */
        sum1[NUM],              /* Sum of the feature vectors for  */
        sum2[NUM],              /* images one and two.             */
        weight[NUM];            /* Weight vector.                  */
        straw[LEN],             /* Feature vector for straw.       */
        raffia[LEN],            /* Feature vector for raffia.      */
        grass[LEN],             /* Feature vector for grass.       */
        paper[LEN];             /* Feature vector for paper.       */

/*-------------------------------------------------------------------*
 * Average values of the features are assigned to their             *
 * respective locations.                                            *
 *------------------------------------------------------------------*/
   straw[0]   0.01425;straw[1] = 1.41331;straw[2] = 0.110844;
   straw[3] = 3.95792;straw[4] = 1.99139;straw[5] = 1.0;

   grass[0] = 0.01195;grass[1] = 1.63886;grass[2] = 0.109533;
   grass[3]= 4.4168345;grass[4]=1.977756;grass[5]=1.0;

   raffia[0]=0.0135;raffia[1]=1.68699;raffia[2]=0.12396;
   raffia[3]=4.87349;raffia[4]=1.96076;raffia[5]=1.0;

   paper[0]=0.02185;paper[1]=1.57132;paper[2]=0.11341;
   paper[3]=4.1439;paper[4]=1.81228;paper[5]=1.0;

/*-------------------------------------------------------------------*
 * Initialization of sum and weight arrays to zero and one.         *
 *------------------------------------------------------------------*/
   for (i = 0; i < NUM; ++i){
       sum1[i] = 0.0;
       sum2[i] = 0.0;
       }

   weight[0] = 1.0;
   weight[1] = 1.0;
   weight[2] = 1.0;
   weight[3] = 1.0;
   weight[4] = 1.0;
   weight[5] = 1.0;
/*-------------------------------------------------------------------*
 * The following for loop computes the sum of two feature           *
 * vectors for two images after multiplying with their              *
 * respective weights.                                              *
 *------------------------------------------------------------------*/

  while (answer == YES)
```

```
  {
  for (i = 0; i < 6; ++i)
        {
        sum1[m] += (grass[i] * weight[i]);
        sum2[m] += (paper[i] * weight[i]);
        l = l + 1;
        }
    wg1 = sum1[m];
    wg2 = sum2[m];
    l = 0;
    sum1[0] = 0.0;
    sum2[0] = 0.0;

/*-------------------------------------------------------------*
 * The weights of the feature vectors are adjusted by a value *
 * alpha until the thresholds of the images one and two are   *
 * satisfied and weight vector converges to an appropriate one*
 *-------------------------------------------------------------*/

    if (wg1 >-1.0){
       if(wg2 <-2.0){
           for (j = 0;j < 6; ++j)
             printf("weights are %lf",weight[j]);
             answer = NO;
             }
        else
             {
             for (i=0; i < 6;i++)
               weight[i] = weight[i] - (alpha * paper[i]);
               answer = YES;
             }
             }
     else
             {
             for (i = 0; i < 6; ++i)
                 weight[i] = weight[i] + (alpha * grass[i]);
             }
   }
/*-------------------------------------------------------------*
 * End of the program.                                         *
 *-------------------------------------------------------------*/
  }
```

```
/*****************************************************************
 *      Department of Electrical and Computer Engineering
 *              Kansas State University
 *          VAX C Source file name :  Pattern.c
 *****************************************************************
 *
 *
 *  FUNCTION:               main()
 *
 *
 *  DESCRIPTION:            The following program computes various
 *                          discriminant functions from the weights
 *                          computed earlier and classifies the
 *                          unknown images based on discriminant
 *                          functions.
 *
 *
 *  DOCUMENTATION
 *  FILES:                  none.
 *
 *
 *  ARGUMENTS:              none.
 *
 *
 *  RETURN:                 none.
 *
 *
 *  FUNCTIONS
 *  CALLED:                 none.
 *
 *
 *  AUTHOR:                 D. K. Durga
 *
 *
 *  DATE CREATED:           December 20, 1988    version 1.00
 *
 *
 *  REVISIONS:              none.
 *
 *
*****************************************************************/

#include <stdio.h>

#define  MAX  40
#define  NUM  6
#define  LEN  24
#define  SIZE 4

main ()
{
/*-------------------------------------------------------------*
 * Declaration of variables.                                   *
 *-------------------------- ---------- ----------------------*/
```

```
 int i,                          /* General purpose loop counters */
     j,
     k=0,
     l=0;

 double w12[NUM], d12[SIZE],    /* Different weights and the      */
        w13[NUM], d13[SIZE],    /* discriminant functions.       */
        w14[NUM], d14[SIZE],
        w23[NUM], d23[SIZE],
        w24[NUM], d24[SIZE],
        w34[NUM], d34[SIZE],
        unknown[EN],            /* Unknown image texture features */
        value;                  /* Texture feature value          */
/*-------------------------------------------------------------*
 * Different weights to calculate discriminant functions.      *
 *-------------------------------------------------------------*/
   w12[0]=0.997565;w12[1]=0.393567;w12[2]=0.967796;
   w12[3]=-0.554508;w12[4]=0.41349;w12[5]=0.7;
   w13[0]=1.009;w13[1]=0.361132;w13[2]=0.986627;
   w13[3]=-1.44534;w13[4]=1.931328;w13[5]=1.4;
   w14[0]=0.78682;w14[1]=-2.985364;w14[2]=0.970437;
   w14[3]=-2.773056;w14[4]=6.954924;w14[5]=1.4;
   w23[0]=0.99502;w23[1]=0.465025;w23[2]=0.954156;
   w23[3]=-0.73604;w23[4]=0.42197;w23[5]=0.7;
   w24[0]=0.76273;w24[1]=-4.866274;w24[2]=-0.780787;
   w24[3]=0.235581;w24[4]=7.817616;w24[5]=-9.4;
   w34[0]=0.874985;w34[1]=-0.828808;w34[2]=0.890021;
   w34[3]=0.422490;w34[4]=-0.958532;w34[5]=-0.9;
/*-------------------------------------------------------------*
 * Prompt the user to enter the texture features of the unknown *
 * image and the image is classified into one of the four images*
 * based on different discriminant functions.                  *
 *-------------------------------------------------------------*/

   for(i = 0; i < 24; ++i)
      {
      printf("Please enter the texture feature \n");
      scanf("%lf",&value);
      unknown[i] = value;
      }
/*-------------------------------------------------------------*
 * Initialization of discriminant function arrays.             *
 *-------------------------------------------------------------*/
   for (i = 0;i < 4; ++i)
       {
       d12[i] = 0.0;
       d13[i] = 0.0;
       d14[i] = 0.0;
       d23[i] = 0.0;
       d24[i] = 0.0;
       d34[i] = 0.0;
       }
/*-------------------------------------------------------------*
 * Linear combination of feature vectors after multiplying     *
```

```
* with their respective weights.                                   *
*------------------------------------------------------------------*/

  for (j = 0;j < 24;j = j + 6)
   {
   for (i = j; i < (j+6); ++i)
       {
       d12[k] += (unknown[i] * w12[l]);
       d13[k] += (unknown[i] * w13[l]);
       d23[k] += (unknown[i] * w23[l]);
       d14[k] += (unknown[i] * w14[l]);
       d24[k] += (unknown[i] * w24[l]);
       d34[k] += (unknown[i] * w34[l]);
       l = l + 1;
       }

 /*--------------------------------------------------------------*
  * The unknown image is classified into one of the four images *
  * depending on the value of the discriminant function .       *
  *--------------------------------------------------------------*/
    if ((d12[k] > 0.0) && (d13[k] >0.0) && (d14[k] > 0.0))
        printf('image belongs to straw');
       else if ((d12[k] < 0.0)  &&  (d23[k] > 1.0) &&
                (d24[k] >-1.0))
                printf('image belongs to grass');
       else if ((d23[k] < -1.0) && (d34[k] > -2.0) &&
                (d13[k] < -2.0))
                 printf('image belongs to raffia');
       else if ((d34[k] < -2.0) && (d24[k] < -2.0) &&
                (d14[k] < -2.0))
                printf('image belongs to paper');
   else if (((d12[k]>0.0) && (d13[k]>0.0)) ||((d14[k]>0.0)&&
             (d12[k]>0.0))||((d13[k]>0.0)&&(d14[k]>0.0)))
             printf('image belongs to straw');
   else if (((d12[k]<0.0) && (d23[k]>-1.0))||((d12[k]<0.0)&&
            (d24[k]>-1.0))||((d23[k]>-1.0)&&(d24[k]>-1.0)))
            printf('image belongs to grass');
  else if (((d34[k]<-2.0)&&(d24[k]<-2.0))||((d34[k]<-2.0)&&
            (d14[k]<-2.0))||((d24[k]<-2.0)&&(d14[k]<-2.0)))
            printf('image belongs to paper');
  else if (((d23[k]<-1.0)&&(d34[k]>-2.0))||((d23[k]<-1.0)&&
            (d13[k]<-2.0))||((d34[k]>-2.0)&&(d13[k]<-2.0)))
            printf('image belongs to raffia');
   k = k + 1;
   l = 0;
   }

/*-----------------------------------------------------------------*
 * End of the program.                                             *
 *-----------------------------------------------------------------*/
}
```

TEXTURE ANALYSIS USING COOCCURRENCE MATRICES

by

DURUVASULA KANAKA DURGA

B.S., University of Mysore, India, 1984

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

A popular technique called Spatial Gray Level Dependence Method (SGLDM) was used to compute and test the classification power of cooccurrence matrices. The SGLDM was particularly chosen because of its powerfulness and simplicity. Various textural features such as entropy, correlation, homogeneity, contrast and inertia were computed from cooccurrence matrices which measure the various visual qualities of textural patterns.

A brief discussion of various classification approaches is included as a background. Details of spatial gray level dependence method and textural features are included .

Final phase of this project consists of classification of various textures using textural measures. Two-dimensional scatter plots and inertia plots were plotted and images were classified to a certain extent. A linear classifier which made use of more than two textural features could classify the entire data with an accuracy of 57 percent.