Image Analysis Tools for the Assessment of Carbon Anodes

A thesis presented in fulfilment of the requirements

for the degree

of Master of Technology

in Manufacturing and Industrial Technology

at Massey University

Alisdair Hamblyn

1993

Abstract

The energy efficiency and performance of an aluminium smelter depends critically on the quality and consistency of properties of the carbon electrodes that are consumed during the normal operation of the electrolytic cells or "pots". Unfortunately, although a small number of experts are able to assess anode quality by examining 10x images of samples, no objective method exists for making quality determinations. This thesis is about a project that has the goal of developing such an objective method.

This thesis describes methods that have been developed for the characterization of the microstructure of carbon anodes. As a result of the process by which they are manufactured, carbon anodes contain pores or voids caused by out-gasing. In this continuing project we have concentrated on developing means for characterising the size and spatial distributions of these voids. Some of the methods used to characterise the spatial distribution include order neighbour analysis (a method used in geographical studies), and statistical texture analysis. These methods and the analysis described in this thesis are of general application.

Acknowledgments

I would like to thank my supervisors, Professor Hodgson and Dr Bailey, for their support and guidance throughout my masterate. Their help in motivating and keeping me on the correct path was appreciated. I would also like to thank Wyatt Page and Ralph Pugmire for their invaluable help with my Pascal programming and putting up with my constant interruptions.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The quality of the anodes used in the smelting process is of critical importance to aluminium manufacture. The Bluff smelter, operated by New Zealand Aluminium Smelters, typically consumes up to 400 one tonne carbon anodes every day, and as the smelter produces their own carbon anodes on-site it has direct control over the anode quality.

The anodes are made up of a mixture of aggregate (85%) and pitch (15%). The aggregate is composed of the butts from old anodes, coarse coke, and crushed fines and is bound together with the pitch. The quality of the anodes is critically dependent on the amount of pitch added when the anodes are being manufactured.

The aggregate and pitch are mixed together at 160 °C and formed with a vibration press into one tonne anodes at 150 °C. These "green" anodes are then baked for approximately 20 days at 1100-1200 °C in a large kiln. The baking process drives off the volatile fraction of the pitch and carbonises the remainder into coke. This release of volatiles forms fine voids in the binder (pitch)/fines matrix.

Too much pitch will inevitably increase the number of voids formed, hence weakening the overall anode structure. This weakened structure can cause an anode to crack and pieces fall into the pot during smelting, which can then put the pot out of action. Also, the increased number of voids present in the anode increases the air permeability and air reactivity. This increases the rate at which the anodes burn away. This overpitching also causes "strings" of voids to form around the larger particles (butts, coke) in the anode, weakening the structure.

Too little pitch in the mixture will cause the anodes to lack binding, and be weak and brittle, leading to similar problems to overpitching. Underpitching also causes many larger voids to form throughout the anode.

An optimum pitching level exists where anodes have a maximum density and a fine porous structure.

Presently, physical properties such as the air permeability, compressive strength, resistivity, carbon dioxide reactivity of the anodes are measured for quality control. Comalco Aluminium Limited have developed a subjective optical macroscopy technique for assessing the pitching level and degree of compaction in baked anodes [1]. This method relies on the subjective visual appraisal by an expert of 10x macrographs of anode samples.

The following points arose from further consultation [2] with experts in visual quality assessment.

- Most of the information needed to distinguish between anodes of varying pitching level is thought to lie within the void structure.
- Not only is the void size histogram of interest, but the relative spacing between voids is also important.
- It is important to analyse a large number of images of each sample to get an idea of the overall structure of the anode. A single image may be misleading. It is assumed that a single anode core sample is sufficiently representative of the complete anode.
- There may be slight differences in structure and texture resulting from different forming processes, and different aggregate batches.

From the above points it can be seen that it is desirable to find methods of characterising the size and spatial distributions of the voids within the anode samples. This project outlines some quantitative image processing methods developed to determine if differences in these distributions exist between anodes of different pitching levels and forming conditions (ie forming temperature and time). This was undertaken as a step to overcoming the subjectivity of methods relying on human vision.

It is believed that most of the void information lies in voids of the size range between 0.05 and 0.7 mm in diameter as this is the void size range the visual experts use to make discriminations. If two pixels represent 0.05 mm then a spatial resolution of 40 pixels/mm is needed. For the purposes of the project two spatial resolutions were concentrated on: 10 pixels/mm and 40 pixels/mm. The 10 pixels/mm was also chosen to determine if there was any gross texture pattern present.

2 BACKGROUND

2.1 Background Literature Search

It was decided to undertake a literature search to find if there were any previous similar studies to this work and to investigate ways of characterising the size and spatial distributions of the voids. The literature search found no previous work on carbon anode quality assessment using image analysis. The search covered statistical texture analysis [3,4,5,6] as a means of characterising the anode quality as this has been successfully applied to industrial problems [7,8] in the past. The search also concentrated on methods of characterising size and spatial distributions. A method found for characterising spatial distributions was order neighbour analysis [9], which is widely used in geographical studies.

These methods are described below.

2.2 Statistical Texture Analysis

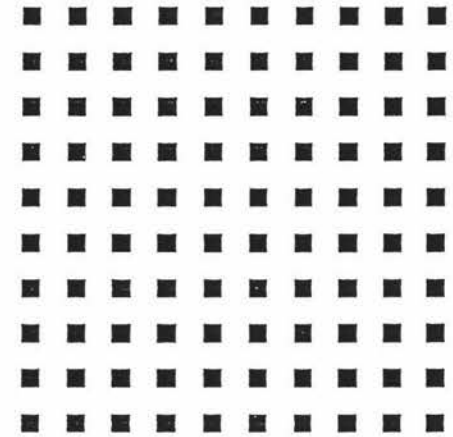
Statistical texture analysis is a powerful technique that is used to characterise texture features in an image in a quantitative, consistent and objective way. The texture of an image is concerned with the spatial distribution of the gray levels in the image. This distribution can be deterministic or stochastic in the extreme as shown in figure 2.1.

Deterministic textures are best analysed using structural methods such as placement rules and tree diagrams while stochastic textures such as carbon anodes are best analysed statistically. These stochastic textures can be analysed statistically using four different intermediate matrix methods [3,4,5,6]:

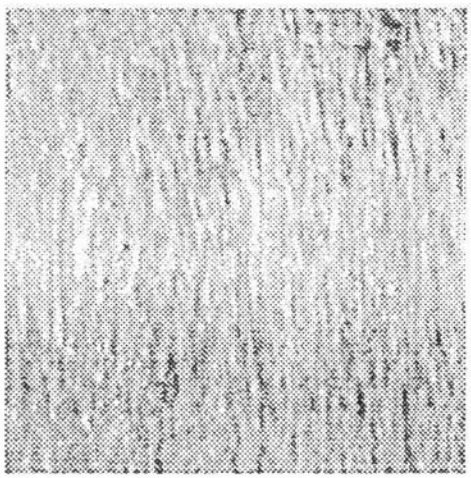
1. Spatial gray level dependency method, SGLDM [3]
2. Gray level difference method, GLDM [4]
3. Gray level run length method, GLRLM [5]
4. Neighbouring gray level dependency method, NGLDM [6]

The intermediate matrices calculated from each of the above methods describe in coded form the spatial relationships between the gray levels in the image. These intermediate matrices allow the calculation of texture features to be made which in turn attempt to describe the texture in a meaningful way. For the purposes of this project we have concentrated on the most popular of the intermediate texture matrix based methods, the spatial gray level dependency method (SGLDM), because it has finer discriminating

power than the other statistical methods. The other three methods have been included here for completeness.



Deterministic



Stochastic

Figure 2.1 - Texture Types

2.2.1 Spatial Gray Level Dependency Method

The spatial gray level dependency method [3] is the most widely used method and is based on the estimation of the second order joint conditional probability density functions *f(i,j,d,a)* where ***d*** is the intersample spacing and ***a*** is the direction *(ie 0°, 45°, 90°, 135°)*. This is illustrated in figure 2.2.

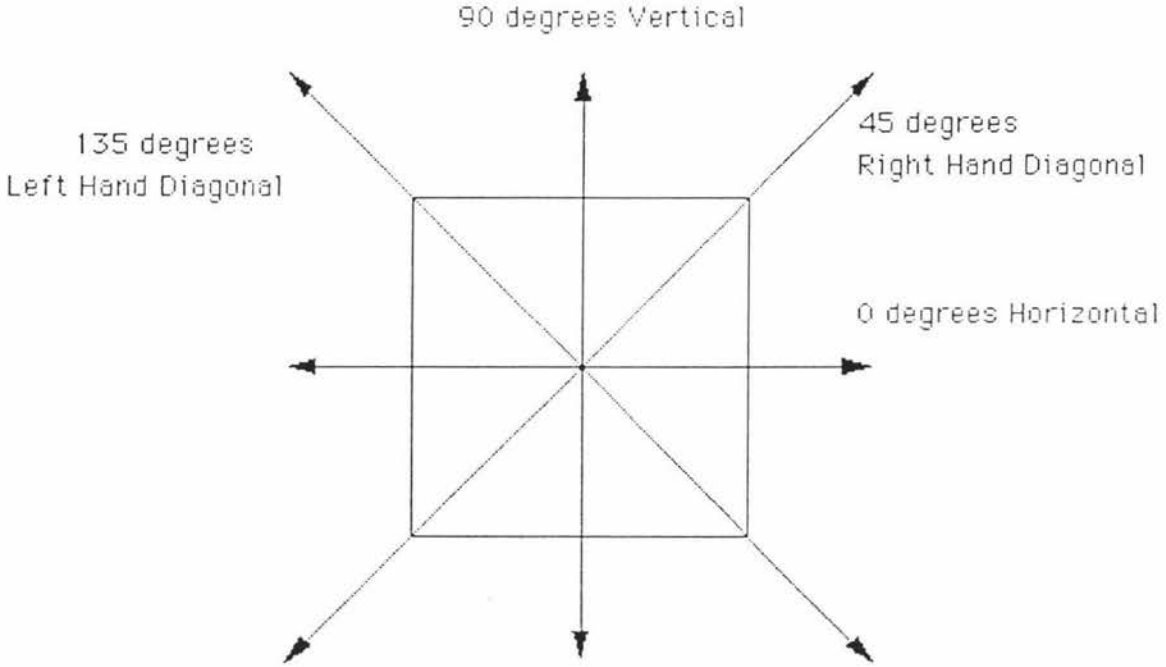


Figure 2.2 - Directions

f(i,j,d,a) is the probability of going from gray level *i* to gray level *j*, in distance ***d*** between the two, and in direction ***a*** *(0°, 45°, 90°, and 135°)*. If there are *Ng* gray levels in the image then the intermediate matrix *P(i,j,d,a)* is *Ng* × *Ng* in size.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	4

Figure 2.3 - Simple Image

The above method calculates four *P(i,j,d,a)* intermediate matrices, one for each direction ***a***. For example, the simple image in figure 2.3 which has 5 gray levels (0-4), will result in four 5 × 5 matrices, one for each direction ***a***. The resulting intermediate matrix for the horizontal direction is shown in figure 2.4.

4	2	1	0	0
2	4	0	0	0
1	0	6	1	0
0	0	1	0	1
0	0	0	1	0

Figure 2.4 - $P(i,j,1,0)$: $0°$

The underscored 4 in the $P(i,j,1,0°)$ matrix above is the number of times a gray level 0 is next to the same level in the image in the horizontal direction.

The underscored 1 in the $P(i,j,1,0°)$ matrix above is the number of times gray level 0 is next to gray level 2 in the image in the horizontal direction.

The four $P(i,j,d,a)$ matrices are normalised by a normalising function dependant on the size of the Region of Interest (ROI) in the chosen image to give the four intermediate matrices: $f(i,j,d,0°)$, $f(i,j,d,45°)$, $f(i,j,d,90°)$, and $f(i,j,d,135°)$.

Haralick et al. [3] proposed various features that can be calculated from the these $f(i,j,d,a)$ matrices. These features are energy, entropy, inertia, homogeneity and correlation. The equations for these features can be found in Appendix D. Analysis of these features may allow discrimination between anode images.

- Energy is a measure of the homogeneity of the image. In a homogeneous image the energy will be high and for a nonhomogeneous image the energy will be low.
- Entropy is a measure of the complexity of an image. The more complex an image is the higher the entropy is.
- Inertia is a measure of the amount of local variations present in the image. The more local variations (contrast) present the higher the inertia.
- Homogeneity is a measure of the degree with which similar gray levels tend to be neighbours.
- Correlation is a measure of gray level linear dependencies.

2.2.2 Gray Level Difference Method

The gray level difference method [4] works on the gray level differences between two adjacent pixels separated by a distance d. Let $f(x,y)$ be the digital image and $f'(x,y) = |f(x,y)-f(x+\Delta x, y+\Delta y)|$ where Δx and Δy are integers giving the displacement d.

Let P' be the probability density function of f'. If there are Ng gray levels in the image then P' has the form of a Ng dimensional vector whose ith component is the probability that $f'(x,y)$ will have value i. It is simple to compute $P'(i)$ from f by counting the number of times each value of $f'(x,y)$ occurs.

The above method is calculated for each of the four basic directions **a** as shown in figure 2.2.

Weszka et al. [4] proposed various features that can be calculated from the four $P'(i)$ matrices. These features are contrast, angular second moment, entropy, mean and inverse different moment. The equations for these features can be found in Appendix D. These features attempt to describe the texture numerically.

- The contrast is the second moment about $P'(i)$. This is greatest when the visual contrast in the image is large.
- The angular second moment is smallest when $P'(i)$ are all as equal as possible and large when some values are high and some low.
- The entropy is largest for equal $P'(i)$ and smallest when they are very unequal.
- The mean is smallest when $P'(i)$ are concentrated near the origin and largest when they are far from the origin.

2.2.3 Gray Level Run Length Method

The gray level run length method [5] is based on calculating the number of gray level runs of various lengths in the four basic directions a as shown in figure 2.2. A gray level run is a set of consecutive co-linear pixels of the same gray level. The length of the run is the number of pixels in the run.

The intermediate matrices $P(i,j)$ specify the number of times the image contains a run of length j, in the given direction, consisting of pixels of gray level i. Let Ng be the number of gray levels and Nr be the number of different possible run lengths.

Using the image in figure 2.5, the $P(i,j)$ matrix for $0°$ is shown in figure 2.6.

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

Figure 2.5 - The Image

	Run Length			
	1	2	3	4
Gray 0	4	0	0	0
Level 1	1	0	1	0
2	3	0	0	0
3	3	1	0	0

Figure 2.6 - $P(i,j)$ for $0°$

The underscored 1 means the gray level 1 has one run of length 3 in the image.

Galloway [5] proposed various features that can be calculated from the four $P(i,j)$ matrices. These features are long run emphasis, short run emphasis, gray level nonuniformity, run length nonuniformity and run percentage. The equations for these features can be found in Appendix D. These features attempt to describe the texture numerically.

- The long run emphasis gives greater weight to long runs of any gray level.
- The short run emphasis gives greater weight to short runs of any gray level.
- When runs are equally distributed throughout the gray levels the gray level nonuniformity is smallest.
- When runs are equally distributed throughout the run lengths the run length nonuniformity is smallest.
- The run percentage is the lower for images with the greatest linear structure.

2.2.4 Neighbouring Gray Level Dependency Method

The neighbouring gray level dependency method [6] is directionally independent. The intermediate matrix $Q(k,s)$ is calculated by considering the relationship between a pixel and all its neighbouring pixels, at a distance less than or equal to d, at one time instead of in one direction at a time.

4	4	6	5	4	3
4	4	5	3	0	1
3	3	5	0	0	1
2	0	7	3	3	2
0	0	7	7	3	3
0	1	6	6	2	2

Figure 2.7 - The Image

For example consider the image in figure 2.7 which has 8 gray levels 0-7.

An intermediate intermediate matrix *P(i,j,d,a)* is calculated. For the above example this is calculated on pixel *3,3*, which has a gray level of 5, in a neighbourhood of ***d=1*** around it, with the difference factor, ***a=zero***. There is only one pixel in the neighbourhood of distance 1 ***(d=1)*** with a gray level of 5 equal ***(a=0)*** to that of pixel *3,3*. Therefore *P(3,3,1,0)=(5,1)* where 5 is the gray level and 1 is the NGLDM number for the pixel *3,3*.

The complete *P(i,j,1,0)* for the image is shown in figure 2.8.

(4,3)	(5,2)	(3,0)	(0,2)
(3,1)	(5,1)	(0,2)	(0,2)
(0,2)	(7,2)	(3,2)	(3,3)
(0,3)	(7,2)	(7,2)	(3,3)

Figure 2.8 - *P(i,j,1,0)*

Q(k,s) is the intermediate matrix for NGLDM and is the total number of entries in *P* that have gray level ***k*** and NGLDM number *s*.

eg. *Q(7,2)=3* because there are 3 entries of *(7,2)* in the *P* matrix in figure 2.8. Therefore the *Q* matrix in figure 2.9 can be considered as frequency counts of the greyness variation of an image. It is similar to the histogram of the image.

		NGLDM Number *s*							
		0	1	2	3	4	5	6	7
	0	0	0	4	1	0	0	0	0
	1	0	0	0	0	0	0	0	0
Gray	2	0	0	0	0	0	0	0	0
Level	3	1	1	1	2	0	0	0	0
k	4	0	0	0	1	0	0	0	0
	5	0	1	1	0	0	0	0	0
	6	0	0	0	0	0	0	0	0
	7	0	0	3	0	0	0	0	0

Figure 2.9 - *Q(k,s)*

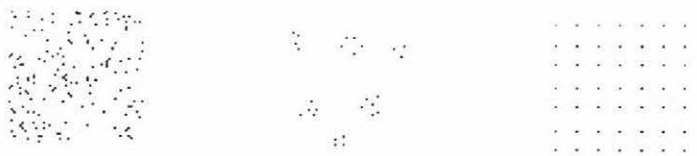
As shown above *Q(7,2) = 3*.

Sun and Wee [6] proposed various features that can be calculated from the *Q(k,s)* matrix. These features are small number emphasis, large number emphasis, number nonuniformity, second moment and entropy. The equations for these features can be found in Appendix D. These features attempt to describe the texture numerically.

- The small number emphasis is a measure of the fineness of the image. The finer the image is the larger the small number emphasis is.
- The large number emphasis is a measure of the coarseness of the image. The coarser the image is the larger the large number emphasis is.
- The number nonuniformity is related to the coarseness of the image.
- The second moment is a measure of the homogeneity of the image. The larger the second moment is the more homogeneous the image is.
- The entropy is related to the coarseness of the image.

2.3 Order Neighbour Analysis

Order neighbour analysis [9] is a method used in geographical studies to analyse point patterns in order to characterise the spatial distribution. Order neighbour analysis recognises three types of point pattern: clustered, random and dispersed. These are shown in figure 2.10.



Random **Clustered** **Dispersed**

Figure 2.10 - Point Pattern Types

The distance to the nearest neighbour is calculated for each of the points in a pattern and from this data an R statistic can be calculated for the nearest neighbour level. It is assumed that there is no boundary around the point pattern as this will restrict the directions in which distance measurements can be made. This R statistic gives a statistical indication of the randomness of the spatial distribution of the pattern and the standard deviation for each R value can also be calculated. The equations for the R statistics and its standard deviation for the nearest neighbour level and the boundless case are shown below in equations 1 to 3 inclusive.

$$R(1) = \frac{\bar{r}(1)}{\rho(1)} \qquad (1)$$

where $\bar{r}(1)$ is the average nearest neighbour distance for the point pattern

$\rho(1)$ is the mean point to point distance for a random pattern

$$\rho(1) = 0.5\left(\frac{N}{A}\right)^{-0.5} \qquad (2)$$

where N is the number of points in the pattern

A is the area covered

$$\sigma[R(1)] = 0.5228N^{-0.5} \qquad (3)$$

Adjustments can be made to the R statistic and its standard deviation to take in to account a boundary being placed around the points.

An R value of 1.0 (within the statistical bounds) indicates that the image is random. If the calculated R statistic is below the lower statistical bound then the image tends towards a clustered pattern and if the calculated R statistic is above the upper statistical bound then the image tends towards a dispersed or regular pattern.

Aplin also suggests that two point patterns can be statistically distinguished from one another by applying a two-sample Kolgomorov-Smirnov (KS) [10] goodness of fit test to the cumulative distributions of the nearest neighbour distances. The KS test is described below.

2.3.1 The Two-sample Kolgomorov-Smirnov Goodness of Fit Test

The two-sample KS test is a test of whether two independent samples have been drawn from the same population (or from populations with the same distribution). The two-tailed test is sensitive to any kind of difference such as a change in the mean or variance in the distributions. The KS test is concerned with the agreement between two cumulative distributions. If the two samples have been drawn from the same population distribution, then the cumulative distributions of both samples may be expected to be similar allowing for random fluctuations. If the two cumulative distributions are too far apart at any point then the samples may have come from different populations, allowing for the level of confidence.

To apply the KS test, cumulative frequency distributions are constructed for both of the point patterns to be tested, using the same intervals for both distributions. For each interval, one of the distributions is subtracted from the other and the largest absolute deviation is used in the KS test. This largest absolute deviation is then compared to the critical KS value which takes into account the number of points in each pattern and the desired level of confidence. If the largest absolute deviation is greater than the critical KS value then the two distributions are statistically different. These points are illustrated in the example below.

2.3.1.1 *KS Test Example*

Sample Intervals (mm)	Pattern 1 Cumulative Distribution	Pattern 2 Cumulative Distribution
< 0.04	0	0
0.04 - 0.095	35	40
0.095 - 0.15	250	176
0.15 - 0.205	811	602
0.205 - 0.26	1388	1073
0.26 - 0.315	1953	1575
0.315 - 0.37	2372	1970
0.37 - 0.425	2725	2262
0.425 - 0.48	2897	2454
0.48 - 0.535	3012	2586
0.535 - 0.59	3086	2665
0.59 - 0.645	3161	2718
0.645 - 0.7	3193	2754
0.7 - 0.755	3205	2773
0.755 - 0.81	3214	2787
0.81 - 0.865	3224	2792
0.865 - 0.92	3226	2802
0.92 - 0.975	3226	2806
0.975 - 1.03	3227	2806
1.03 - 1.085	3227	2806
1.085 - 1.14	3228	2806

Figure 2.11 - Cumulative Distributions

Pattern 1 Cumulative Frequency Distribution	Pattern 2 Cumulative Frequency Distribution	Absolute Differences
0	0	0
0.0108	0.0142	0.0034
0.0774	0.0627	0.0147
0.2512	0.2145	0.0367
0.4299	0.3823	**0.0476**
0.6050	0.5612	0.0437
0.7348	0.7020	0.0327
0.8441	0.8061	0.0380
0.8974	0.8745	0.0229
0.9330	0.9215	0.0114
0.9560	0.9497	0.0062
0.9792	0.9686	0.0106
0.9891	0.9814	0.0076
0.9929	0.9882	0.0046
0.9956	0.9932	0.0024
0.9987	0.9950	0.0037
0.9993	0.9985	0.0008
0.9993	1	0.0006
0.9997	1	0.0003
0.9997	1	0.0003
1	1	0

Figure 2.12 - Differences

Figure 2.11 shows a table of the cumulative distributions of the nearest neighbour distances for two different point patterns.

The next step is to normalise the cumulative distribution with respect to the total number of points in each pattern to give the cumulative frequency distributions. For the first pattern the total number of voids is 3228, and for the second pattern it is 2806.

The KS test is performed on the cumulative frequency distributions firstly by finding the largest absolute difference between the two distributions and comparing this value to the critical KS value. Figure 2.12 shows the cumulative frequency distributions and the resulting absolute differences. The largest absolute difference is shown in bold type.

The largest absolute deviation between the two cumulative frequency distributions is 0.0476. This value is compared to the critical KS value which takes into account the number of observations (points in each pattern) and the desired level of confidence. If the critical KS value is less than 0.0476 then the two patterns are statistically different (ie the largest absolute deviation is greater than the critical KS value). For the 99% level of confidence the critical KS value was calculated as shown in equation 4 [10]:

$$KS_{crit} = 1.63\sqrt{\frac{n_1 + n_2}{n_1 n_2}} \qquad (4)$$

Where n_1 and n_2 are the total number of points in patterns 1 and 2 respectively.

For this example the critical KS value is 0.04207, which is less than the largest absolute deviation. This means the two point pattern spatial distributions are different at the 99% level of confidence. Another useful measure is to calculate the KS ratio which is the largest absolute deviation divided by the critical KS value. In this case the KS ratio is 1.13. As the KS ratio is greater than 1.0 then we can say the two point pattern spatial distributions are different. A KS ratio less than 1.0 would indicate that the spatial distributions were the same. This measure gives an indication of the amount of difference.

The sensitivity of the KS test was tested using gaussian curves based on equation 5:

$$y(x) = \frac{1}{\sigma\sqrt{(2\pi)}} e^{-\left[\frac{(x-\mu)^2}{2\sigma^2}\right]} \qquad (5)$$

where σ is the standard deviation and μ is the mean.

Two gaussian curves with a mean of 0 and a standard deviation of 1 were created and the standard deviation of one of the curves was increased. The KS ratio was calculated at a 99% level of confidence as the standard deviation increased. The standard deviation at which the KS ratio became 1 was found to be 1.019 or a 1.9% increase.

The process was then repeated for a decreasing standard deviation at a 99% level of confidence. The resulting standard deviation at which the two distributions became significantly different was found to be 0.981 or a 1.9% decrease.

The process was then repeated for a changing mean at a 99% level of confidence where the standard deviation was kept constant. The resulting mean at which the two distributions begin to differ was 0.06167.

From the above sensitivity tests it was concluded that the KS test is quite sensitive to small variations in the mean and standard deviations of the distributions being tested.

3 TOOLS USED

This section describes the hardware and software used in the project. The preparation of the sample anodes and the image capture process is also described. The final part of the section describes the image processing routines implemented in the project.

3.1 Hardware Used

The computer hardware used in the project consisted of three main components:

- A host computer
- A frame-grabber
- A solid-state camera

Each of these will be discussed in more detail below.

3.1.1 The Host Computer

The host computer used for this project was a Macintosh Quadra 900 running Operating System 7.0.1 with 8 Mb of RAM. The "Image" [11,12] and "VIPS" [13] image processing software were installed on this computer. The frame-grabber card and associated utility software (QuickCapture™) were also installed on the Quadra and as the "Image" software supported the frame-grabber directly, the images were captured within "Image".

3.1.2 The Frame-grabber

The frame-grabber used was an 8-bit Data Translation (DT2255-50Hz) QuickCapture™ card which captures 512 lines x 768 pixels.

3.1.3 The Camera

The camera used was an Ikegami ICD-44DC black and white solid-state video camera. A black and white camera was chosen because a colour camera requires the colour subcarrier signal to be filtered out before being used with a monochrome frame-grabber. This camera incorporates a 1/2 inch CCD image sensor with 512 horizontal and 582 vertical picture elements. The lens used to capture the images had a focal length of 25mm.

3.2 Sample Preparation

An anode sample is normally in the form of a flat slice about 50 mm^2 in area. The voids of interest are the cavities formed by this slice through the 3-D matrix of the original anode as shown in figure 3.1.

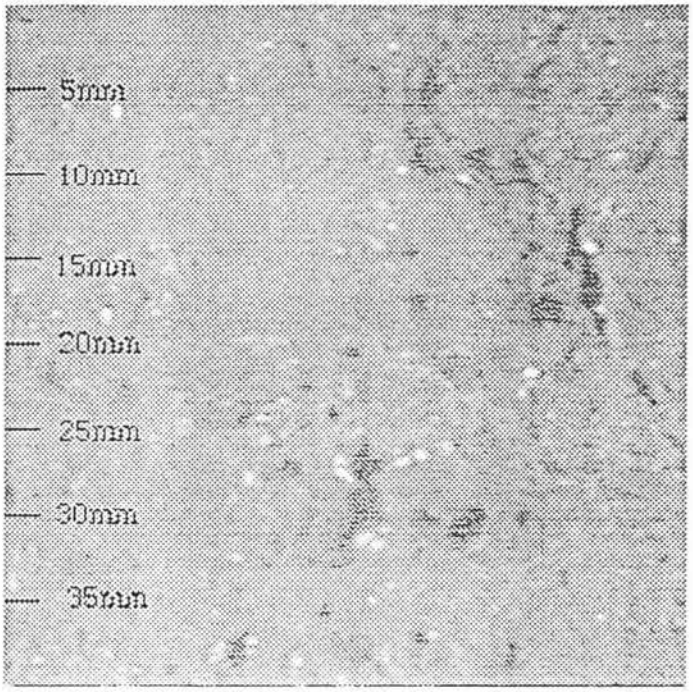


Figure 3.1 - Anode Sample at a 40 pixel/mm Resolution

Since we are looking at the size and spatial distributions of the voids, the voids have to be distinguished from the background in some way. Two methods were tried:

- Direct/Side lighting
- Void filling

In the first method, images of the same scene were captured with direct lighting and then with side lighting. When the images are subtracted the resulting image should show the voids. This method was abandoned because we could not get adequate direct lighting onto the anode samples at high magnifications because of the small camera to sample distances.

The second method involves filling the voids with a white substance to contrast the voids to the gray background. Firstly chalk was used to fill the voids by rubbing a stick of chalk over the polished anode surface. This was not very successful as the chalk would not fill the largest voids and since the voids were filled with dust, the dust sometimes came out of the voids. This problem was rectified by using plaster of paris instead of chalk. A wet plaster mixture was spread over the anode surface using a wall paper stripper to fill the voids, then the wet surface was scraped to remove the excess plaster. The plaster was left to dry and given a light rub with a rag to remove the plaster from the anode surface. This method resulted in the plaster filled voids having a high contrast to the background as shown in figure 3.2.

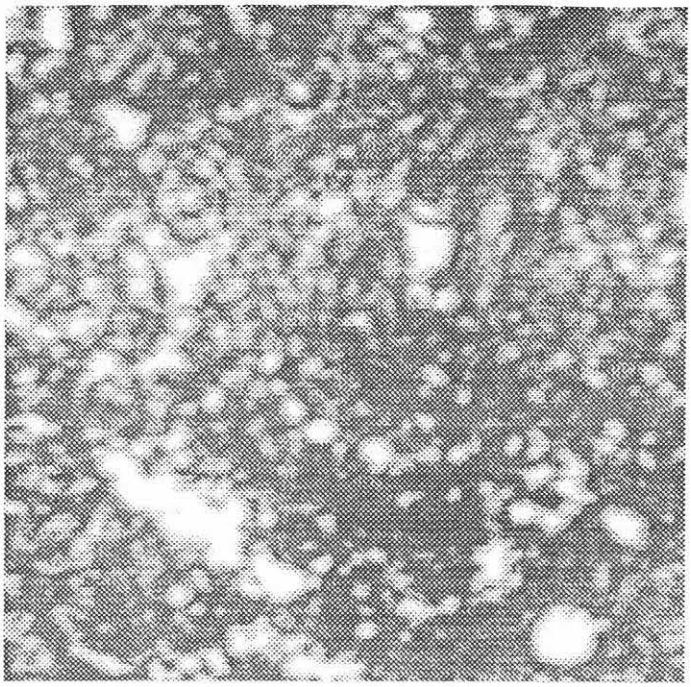


Figure 3.2 - Another Sample at 40 Pixels/mm after Plaster Applied

Once the voids have been filled using plaster the image can then be thresholded to identify the voids as shown in figure 3.3. Note for convenience the images have been inverted; the black areas correspond to the voids and the white area corresponds to the background. From these binary images further analyses can be performed to characterise the size and spatial distributions.

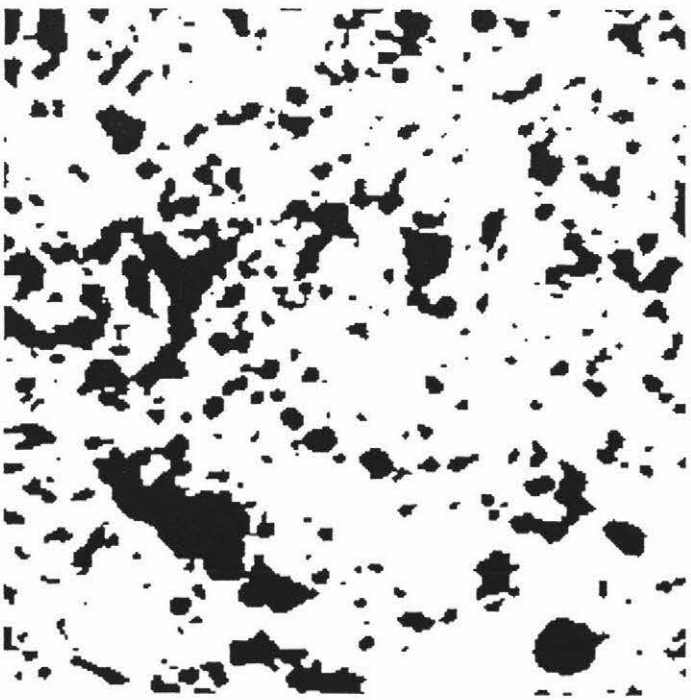


Figure 3.3 - Thresholded Result

Once images of the required spatial resolution have been captured in "Image" the resolution of 10 or 40 pixels/mm can be set into "Image" to calibrate the pixel measurements into SI units.

The next step in the process was to capture images of the anode samples.

3.3 Image Capture

Images were captured using the arrangement shown in figure 3.4. This set-up was inside a booth with curtains to keep out background light.

The camera was mounted on an adjustable mount which allowed the user to set the height of the camera above the sample. The lighting was provided by four moveable lamps which were adjusted to give an even lighting over the sample. The sample was placed inside an X-Y table to move the sample in the horizontal and vertical directions normal to the camera.

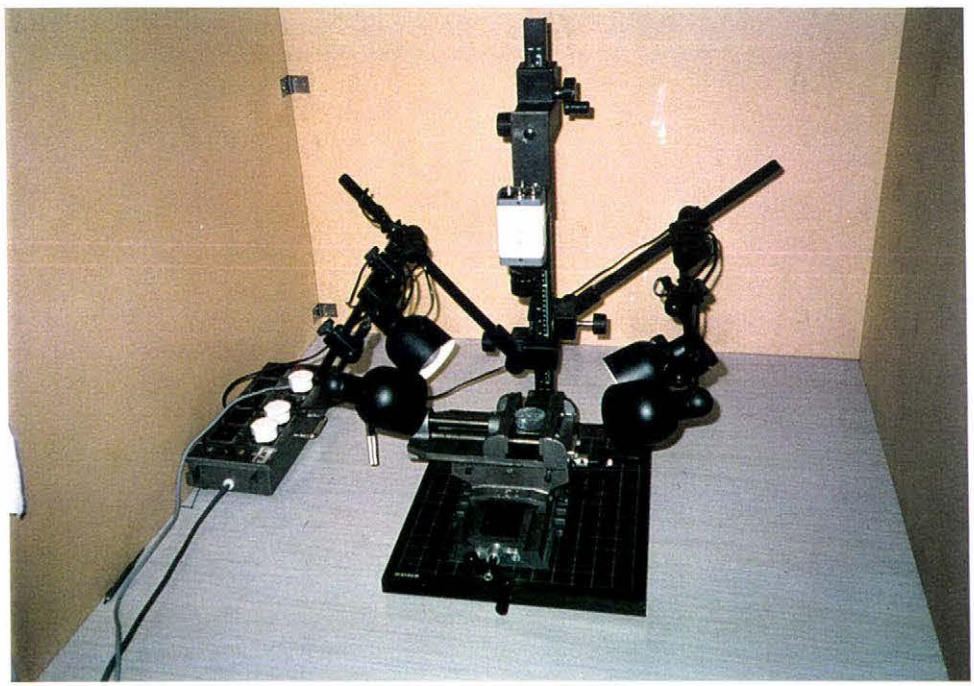


Figure 3.4 - Capture Set-up

To capture an image, the prepared sample was placed in the X-Y table directly below the camera and lens arrangement. The "Image" software was run and the live capture command was chosen. The camera lens and the height were adjusted to give a sharply focused image. The lighting was then adjusted to give uniform illumination. The lighting was checked by constructing plot profiles of the illumination in the vertical and horizontal directions on the captured images in "Image". If there was a significant slope in the gray scales with respect to the number of pixels then the lighting was uneven and the lighting adjusted accordingly.

To calibrate the spatial resolution, a scaled ruler was placed under the camera on top of the sample and a focused image of the ruler was captured. Using "Image", the number of pixels between two points on the ruler was measured. Since the actual physical distance and the number of pixels between two points was known, the spatial calibration in pixels/unit distance could be calculated. "Image" allows this spatial calibration to be entered in SI units and after calibration, all subsequent calculations are output in the SI unit specified rather than pixels.

3.4 Software Used

The major software applications that were used in the practical part of the project were "Image" [11,12] and "Excel".

"Image" is a public domain image processing and analysis program for the Macintosh, written by the American National Institute of Health. It supports all major image file formats such as TIFF and PICT and any measurements made can be saved as "Excel" text files. It supports many standard image processing functions, including histogram equalisation, contrast enhancement, density profiling, smoothing, sharpening, edge detection, median filtering, and spatial convolutions. Image can also be used to measure the areas, positions, and perimeters of binary blobs (voids) in an image. The major reason "Image" was chosen was because of the Pascal-like macro language for automating repetitive tasks, and the availability of the complete Pascal source code. The structure of the source code allowed the easy addition of user defined image processing routines at the source code level. This approach was used throughout the project.

"Excel" was used mainly because it is the spreadsheet used on the Macintosh computers in the Department, and the compatibility with "Image" was coincidental. "Excel" was used to process the raw measurement data from "Image" and to perform the KS tests.

3.5 Image Processing Routines

As mentioned earlier, "Image" allowed the easy addition of extra image processing routines. The image processing routines added for this project were:

- Statistical Texture Analysis [3,4,5,6]
- Rank and Range Filters [14]
- Area/Distance Factors
- Weighted Area
- Nearest Neighbour Analysis (Spatial Distribution) [9]
- Circularity
- Connectivity
- Texture Primitives [15]

A user manual outlining the above routines can be found in Appendix A. Of the above routines, the area/distance factors, the weighted area, the circularity, and the connectivity were features created especially for this project, whereas the other routines were written by others. All the above routines except the statistical texture analysis and the rank/range filters are specifically designed to be run on binary images which show the voids as black and the background as white.

3.5.1 Statistical Texture Analysis

Statistical texture analysis techniques were described in Section 2.2. The four major statistical methods (SGLDM, GLDM, GLRLM, and NGLDM) were implemented in "Image" and the source code can be found in Appendix B.

3.5.2 Range and Rank Filters

As "Image" did not have a rank or range filter [14] available, it was decided to add them for future use and completeness. A rank filter is a local operator filter which uses the pixel values contained in a window around and including a particular pixel. For this project a window of 3 x 3 was used.

The rank filter works by sorting the 9 pixel values in the 3 x 3 window into numerical order from 0 (white) to 255 (black). Note that Macintosh computers regard a 0 pixel value as white, and not black as most other systems. For a rank filter of rank 1 (a MIN filter), the central pixel in the 3 x 3 window is replaced by the lowest value of the sorted pixels in the window. Similarly for a rank filter of rank 9 (a MAX filter), the central pixel in the 3 x 3 window is replaced by the highest value of the sorted pixels in the window. A rank filter of rank 5 acts as a median filter and replaces the central pixel by the median of the sorted pixels. The MIN filter will remove layers of pixels from the black voids in binary images, and the MAX filter will add layers of pixels to the black voids. When the rank

filter command is run in "Image", the user is asked the rank (from 1 to 9 inclusive) to be used.

The output of a range filter is the difference of two rank filtered images. This subtraction means the range filter is useful as an edge detector. When the range filter command is run in "Image", the user is asked the ranks (from 1 to 9 inclusive) of the two rank filters to be subtracted. The value of the second rank entered is subtracted from the first.

A problem with window filters such as the rank and range filters is that the outside layer of pixels around the image does not have an output. Other image processing systems such as VIPS [13] extend the second layer of pixels out to the outermost layer, and this approach was implemented in "Image" as the EXTEND command. When the rank or range filter is run in "Image" on any image, the outermost layer of pixels is set to white. This will remove any void pixels (black) touching the edges of a binary image when using the rank filter as a MIN filter. The EXTEND command may be used to extend grayscale images if desired.

The source code for the rank and range filter routines can be found in Appendix B.

3.5.3 Area/Distance Factors

The area factor (AF) measures the spatial and size distribution from the areas and distances between all the voids in an image. For an image, the AF is calculated as follows:

$$AF = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{N}\frac{A_j}{\left(d_{ij}\right)^2}\right] \qquad i \neq j \tag{1}$$

where N is the total number of voids in the image

A_j is the area of the *jth* void

d_{ij} is the distance between the *ith* and *jth* void

The distance factor (DF) quantifies only the spatial distribution of the voids. This measure is the same as the area factor except that it is not weighted by the areas of the voids.

$$DF = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{N}\frac{1}{\left(d_{ij}\right)^2}\right] \qquad i \neq j \tag{2}$$

When "Image" runs the area and distance factor routines, the result is a column of area or distance factors which shows the individual area or distance factors for each void. In order to get the area or distance factor for the image as a whole the resulting column above is averaged to give the equations above.

The source code for the area and distance factors routines can be found in Appendix B.

3.5.4 Weighted Area

This is a measure of the volume of the voids in a sample. It is assumed that the larger the cross-sectional area of a void in our image of a sample, the larger the volume of the void in the original material will be. Each pixel within a void is weighted in proportion to its distance from the edge. The weighted area (WA) is calculated by using a MIN filter to remove the outermost layer of pixels from all of the voids in the image. The change in total area is given a weighting of 1 for the first layer removed, a weighting of 2 for the second layer etc, as shown in figure 3.5. The process is repeated until all the voids in the image have disappeared, that is eroded to nothing. The area changes for each layer removed, along with the respective weightings, are summed to give a weighted area for the image as a whole.

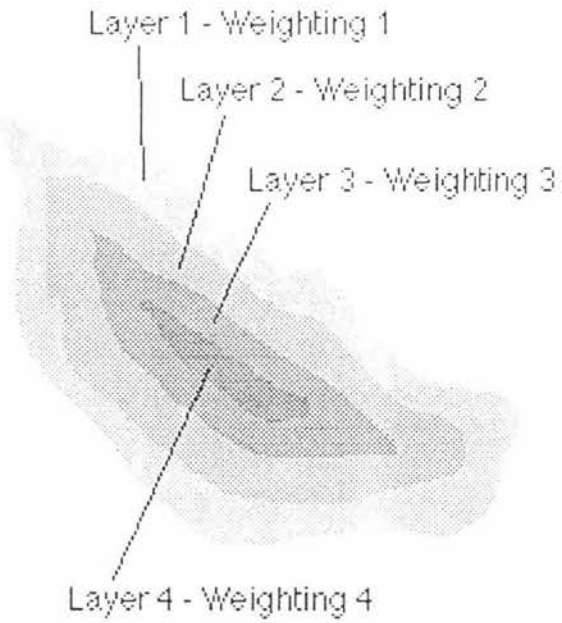


Figure 3.5 - WA Four Layers

The weighted area can be summarised in the following equation:

$$WA = \sum_{i=1}^{N} iA_i \qquad (3)$$

Where N is the number of layers in the void

A_i is the area of the layer

The source code for the weighted area routine can be found in Appendix B.

3.5.5 Nearest Neighbour

In this routine the nearest neighbour distances are calculated for characterising the spatial distribution. The data from this routine is analysed in "Excel" for the KS tests. "Image" calculates the distance to the nearest eight neighbours for each void in the image. If the distance to the nearest boundary is closer to a void than the distance to its first nearest visible neighbour, then the first nearest neighbour distance for that void is not used. This logic is used because what lies outside the boundary is unknown. If a void has no first nearest neighbour then it cannot possibly have a second or third and so on. This process can be repeated for higher neighbouring levels. Therefore, if the distance to the nearest boundary is closer to a void than the distance to its second nearest visible neighbour, then the second nearest neighbour distance for that void is not used. This logic is again applied because what lies outside the boundary is unknown. If a void has no second nearest neighbour then it cannot possibly have a third or fourth and so on. It must be remembered that the project concentrated on the (first) nearest neighbour distances for use in the KS tests. The average of the nearest neighbour distances is also used as a feature for making discriminations between anodes (see section 3.6). The higher neighbour levels were included for completeness and potential use in more complex KS testing in the future.

Each time the neighbour level increased, the effective area over which the neighbours were analysed grew smaller. This occurs because the outer voids were being ignored as they were closer to the boundaries of the image than their nearest visible neighbour. These area changes are noted by "Image" and recorded for use in calculating the R statistics in "Excel". These R statistics can be assumed to have no boundary as only voids which have visible nearest neighbours closer than the nearest boundary are accounted for. For this reason the equations shown in section 2.3 can be used for the nearest neighbour level without making any adjustments for a boundary.

The source code for the neighbour analysis routine can be found in Appendix B.

3.5.6 Circularity

This routine measures the lengths of the major and minor axes of the best fitting ellipse of a void in an image. The circularity is the length of the minor axis divided by the the length of the major axis. Therefore a circularity of 1.0 would indicate a perfectly circular void. The lower the circularity, the longer and thinner the void in question. The circularity measure works best on voids with a large number of pixels as the circularity measure is distorted when measuring extremely small voids (<20 pixels).

The source code for the circularity routine can be found in Appendix B.

3.5.7 Connectivity

During investigations into effects of the MAX filter on the void images, it was decided to look at the changes in the number of voids in the image as successive layers of pixels were added to the voids. This change in the void count as layers of pixels were added was called connectivity. The final result of the connectivity analysis is one single blob where all the original voids in the image have been connected together. The result of this analysis is a plot of the number of voids as a function of the number of layers added. This plot is shown in figure 3.6.

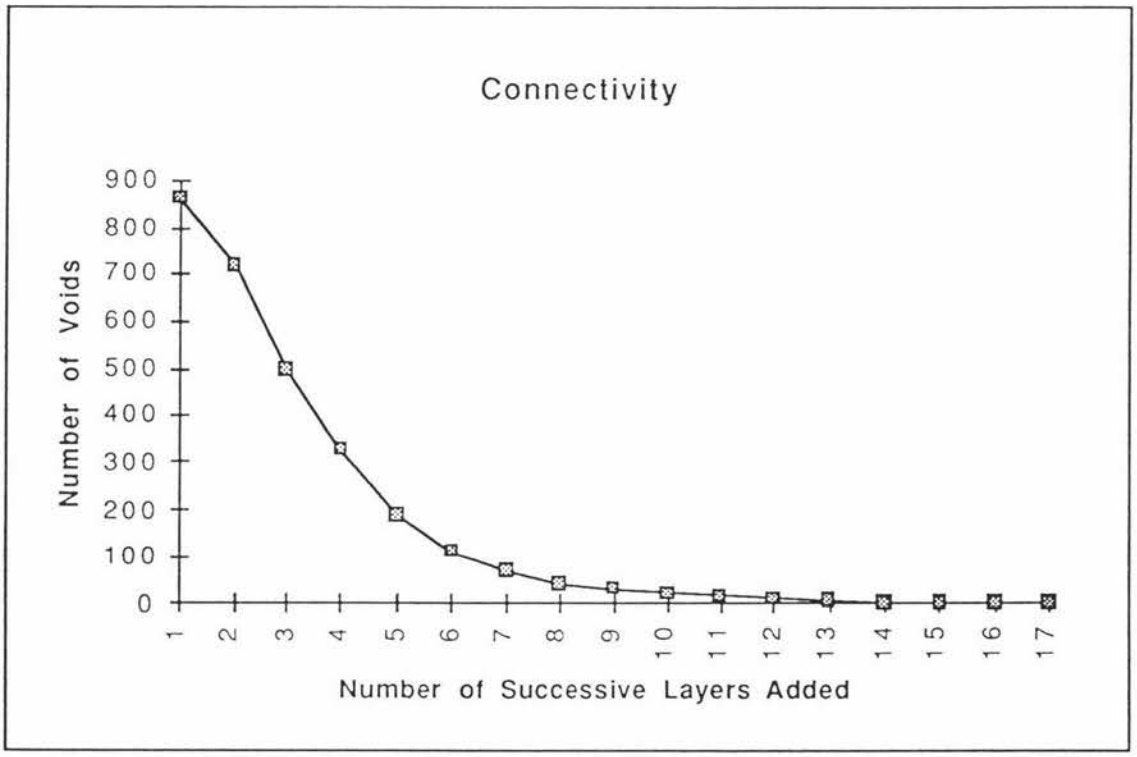


Figure 3.6 - Connectivity Results

The connectivity plot shows the changes in the void count as successive layers of pixels were added to the voids in the image. The plots of different images could be compared by plotting on the same graph and visually inspecting them.

Another way to compare connectivity plots from different images would be to use the KS test as these connectivity plots could be viewed as cumulative plots. A problem with this method is the different number of layers added in different images. This is a problem because the KS test requires the x axis for all the plots being compared having the same number of points or bins. Normalisation of the x axis to a single number would be difficult as this would require complex interpolation of the data.

When the connectivity routine is run in "Image" the number of voids in the original image is counted and recorded. Next a layer of pixels is added to the voids and the count is recorded. This process is repeated until there is only one void left and the data is

displayed for saving by the user. To avoid a limitation inherent in the void counting algorithm used by "Image", a new void counting algorithm was written for the connectivity routine to overcome this. This new void counting algorithm was a modified blob counting routine from VIPS [13] called BLOB.

3.5.8 Texture Primitives

This routine uses various attributes of the texture primitives [15] (in this case the voids) such as the area, perimeter, compactness, eccentricity, and direction to analyse the images. The area and perimeter of a void are self-explanatory. The compactness is the ratio of the square of the perimeter to the area. The eccentricity is the inverse of the circularity, and the direction is the angle the major axis makes with the horizontal.

The steps "Image" takes to calculate the texture primitives are as follows:

- The attributes (area, perimeter, compactness, eccentricity, and direction) are calculated for each void in the image
- For each void the four nearest neighbours are found
- Primitive attribute co-occurence matrices are constructed for each attribute
- Second-order statistics are calculated from these co-occurence matrices

The first two steps have been discussed earlier. The final two steps resemble statistical texture analysis.

For each of the attributes, the values are divided into N=16 intervals. These intervals form an N x N co-occurence matrix for each attribute. To construct the co-occurence matrix P for the area attribute say, each void and it four neighbours are looked at. If the area for a particular void is in interval i and the nearest neighbour's area is in interval j, then we add 1 to $P(i,j)$, the $(i,j$th entry of the area co-occurence matrix. Finally, the matrix is normalised by dividing each entry of P by:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} P(i,j) \qquad (4)$$

From the co-occurence matrix for each attribute, the following second-order textural statistics are calculated:

- Angular Second Moment (ASM)

$$ASM = \sum_{i=1}^{N}\sum_{j=1}^{N}\{P(i,j)\}^2 \qquad (5)$$

- Entropy (ENT)

$$ENT = -\sum_{i=1}^{N}\sum_{j=1}^{N} P(i,j)\log(P(i,j)) \tag{6}$$

- Inverse Different Moment (IDM)

$$IDM = \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{P(i,j)}{1+(i-j)^2} \tag{7}$$

- Contrast (CON)

$$CON = \sum_{i=1}^{N}\sum_{j=1}^{N} \left\{(i-j)^2 P(i,j)\right\} \tag{8}$$

When the texture primitive routine is run, "Image" calculates a table of 20 texture primitives, ie four columns of second-order statistics and five rows of attributes. These second order statistics allow the discrimination of images with different textures.

3.6 Features Used

For the purposes of the project it was decided to concentrate on the following ten features.

1. Energy
2. Entropy
3. Inertia
4. Homogeneity
5. Correlation
6. Area Factor
7. Distance Factor
8. Average Area of the voids present in an image
9. Weighted Area
10. The Average Nearest Neighbour Distance

The first five features are the second-order statistics from the spatial gray level dependency method, and the last five have been described above.

4 RESULTS AND DISCUSSION

4.1 Introduction

As discussed in earlier sections, the size and spatial distributions of the voids in an anode sample are important for assessing the effect of pitching level on the quality of the anode. For this reason, it was decided to investigate the changes or trends in the area and nearest neighbour distance distributions and the image features described in the previous chapter with respect to the pitching level and the forming conditions of the anodes. This will characterise the area and spatial distribution's sensitivity to changes in pitching level and forming conditions.

The ten image features that were used were the five SGLDM texture analysis features, namely: energy, entropy, inertia, homogeneity, and correlation; and the area factor (AF), distance factor (DF), the average area of the voids, the weighted area (WA) and the average nearest neighbour distance (NN).

The pitching levels looked at were classified by Comalco as: underpitched (UP), good, slightly overpitched (SOP), and overpitched (OP). It must be noted that these pitching levels are not measured values, but are assessments made by visual inspection of the anode samples by an expert (See Appendix F).

The forming condition parameters considered were: the time the green paste spent in the vibration former, and the temperature. Anode samples formed under six different conditions were compared:

1. 25 seconds in the vibration former at a temperature of 147 °C (25@147).
2. 25 seconds in the vibration former at a temperature of 149 °C (25@149).
3. 45 seconds in the vibration former at a temperature of 153 °C (45@153).
4. 45 seconds in the vibration former at a temperature of 155 °C (45@155).
5. 75 seconds in the vibration former at a temperature of 145 °C (75@145).
6. 75 seconds in the vibration former at a temperature of 151 °C (75@151).

For both the pitching levels and forming conditions, analysis was performed at 10 pixels/mm and 40 pixels/mm resolution for reasons stated earlier.

For all the above conditions, the image features were graphed against the changes in pitching level and forming conditions. Visual trends and high statistical correlations between image features and the pitching level or forming conditions were looked for.

Kolgomorov-Smirnov (KS) tests were performed on the area and nearest neighbour distance distributions to look for differences (or similarities) between anode samples.

4.2 Changes in Pitching Level

4.2.1 10 pixels/mm

4.2.1.1 *Image Features*

Images of a resolution of 10 pixels/mm were captured for each of the prepared samples of different pitching levels. At this resolution only one image could be obtained from each sample. These images were 256 x 256 in size and covered an area of 27.23 mm x 27.23 mm (741.5 mm^2). This provided four images to work with, one of each pitching level.

For each image, the image features were calculated using "Image" and "Excel", and the area and nearest neighbour distance distributions were also noted. Graphs of the image features as a function of the pitching level are shown in Figures 4.1 to 4.10 below.

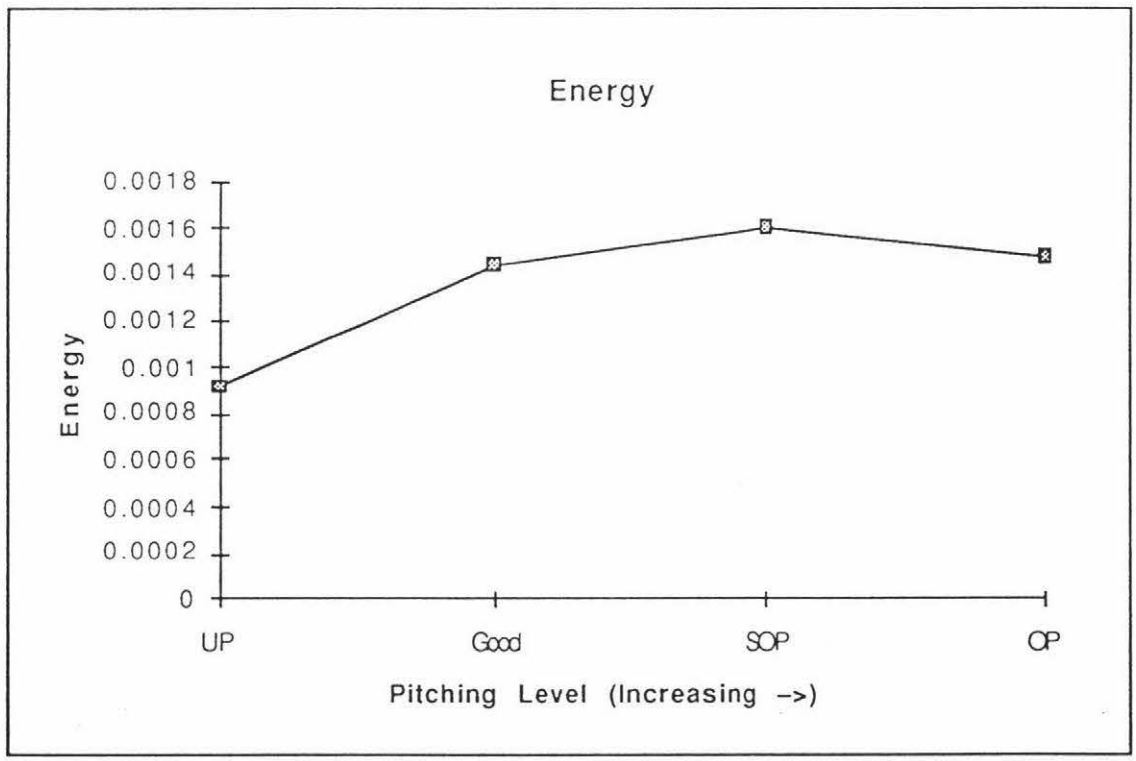


Figure 4.1 - Energy vs Pitching Level

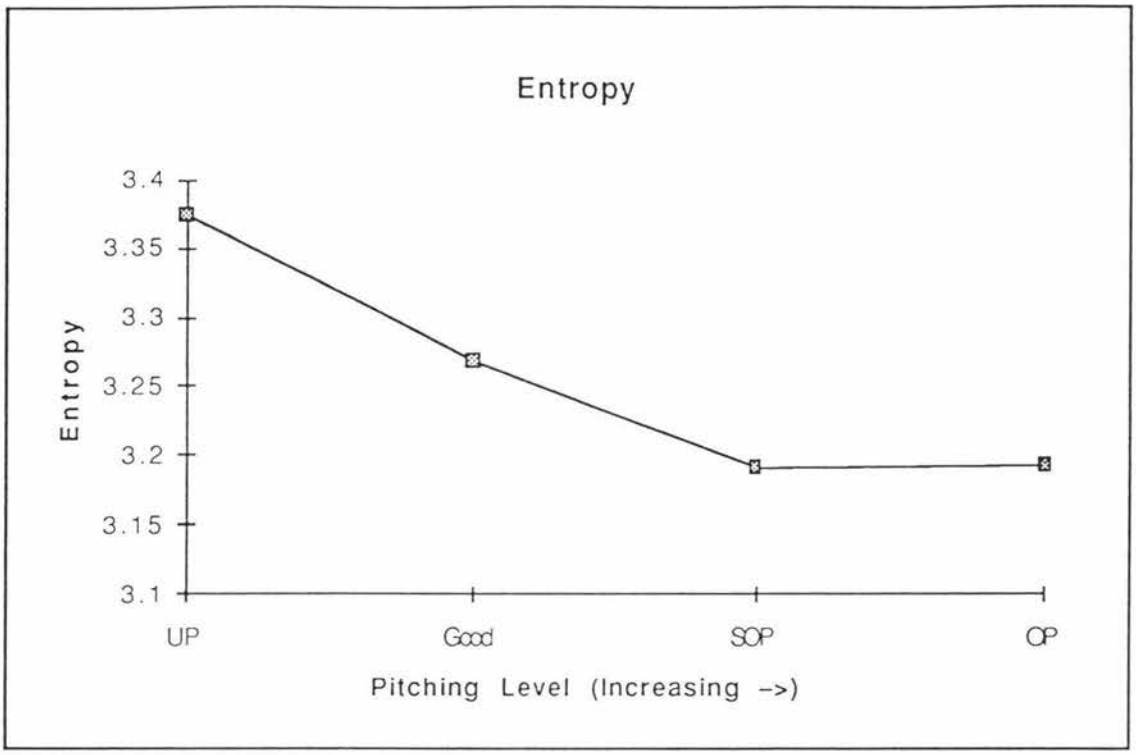


Figure 4.2 - Entropy vs Pitching Level

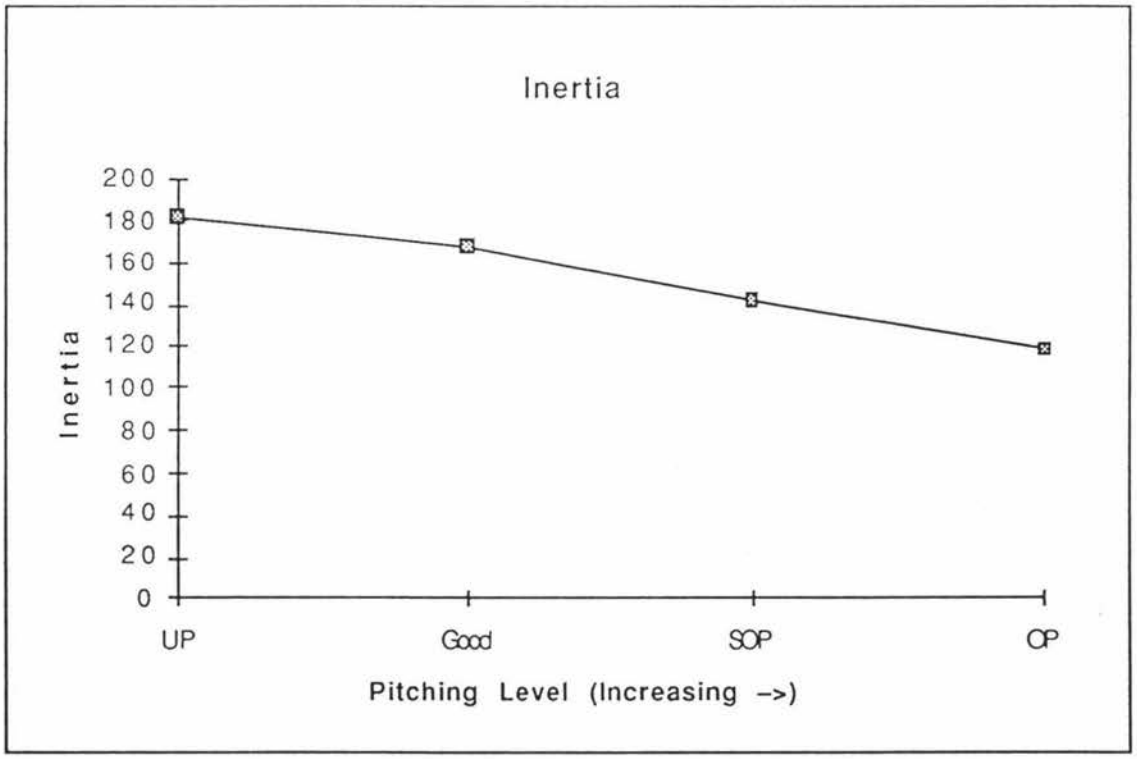


Figure 4.3 - Inertia vs Pitching Level

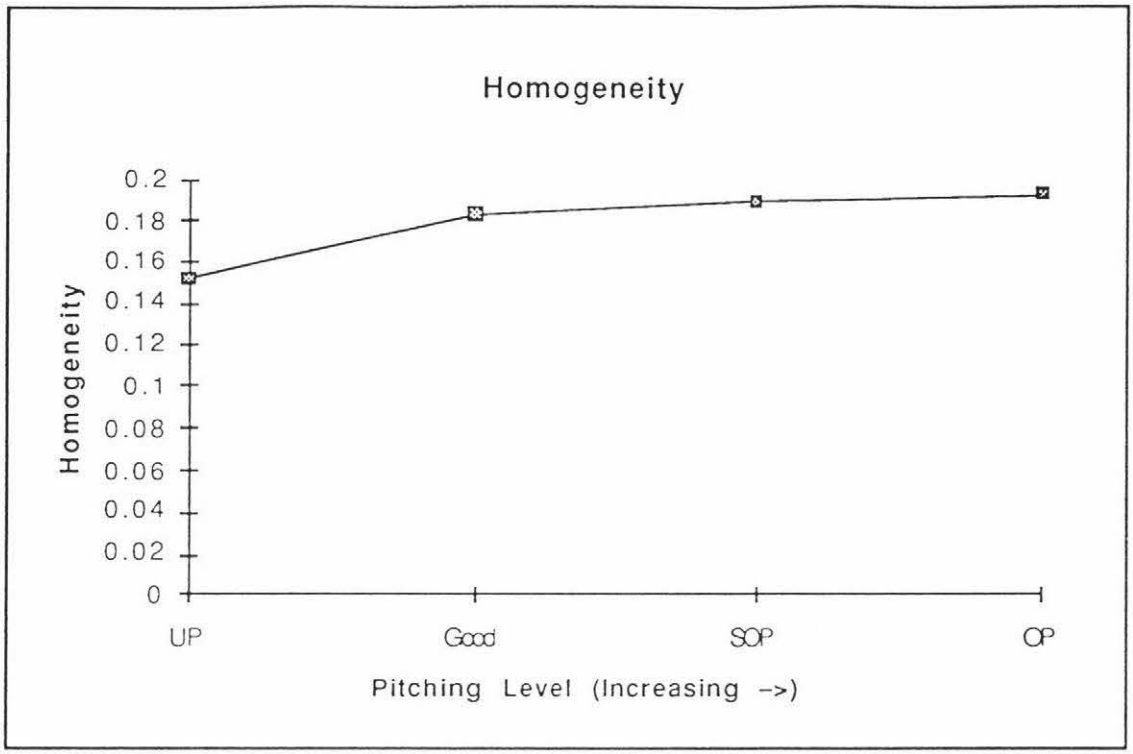


Figure 4.4 - Homogeneity vs Pitching Level

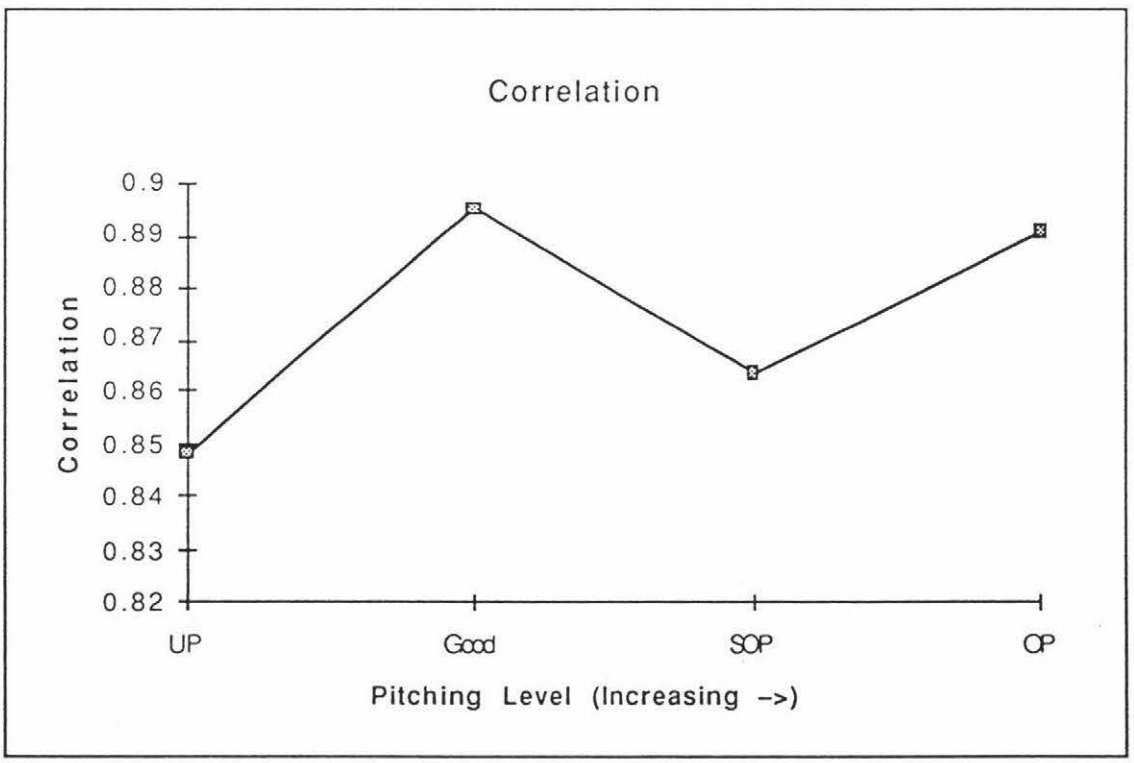


Figure 4.5 - Correlation vs Pitching Level

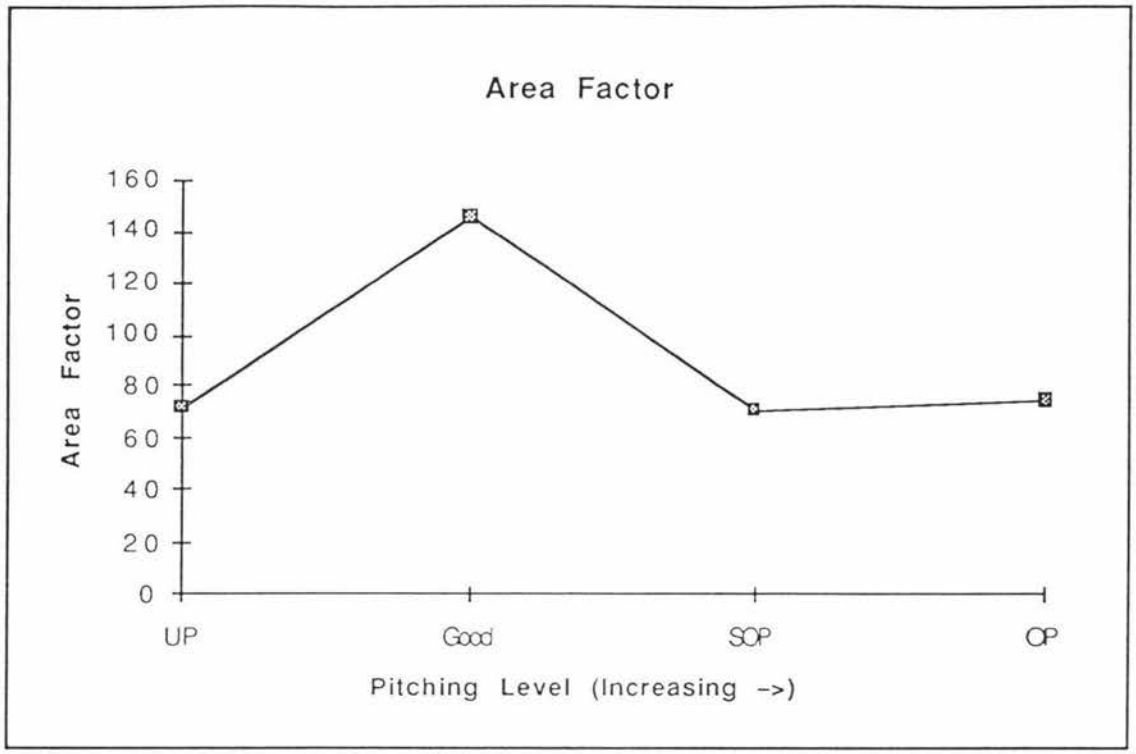


Figure 4.6 - Area Factor vs Pitching Level

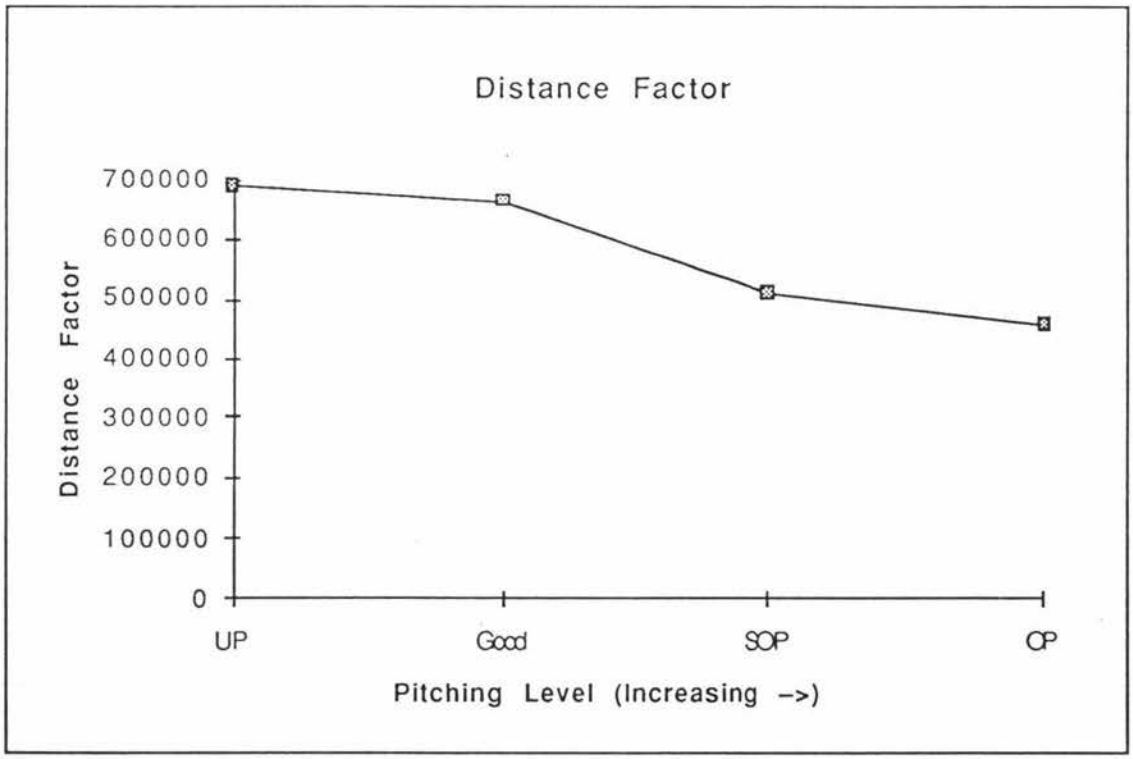


Figure 4.7 - Distance Factor vs Pitching Level

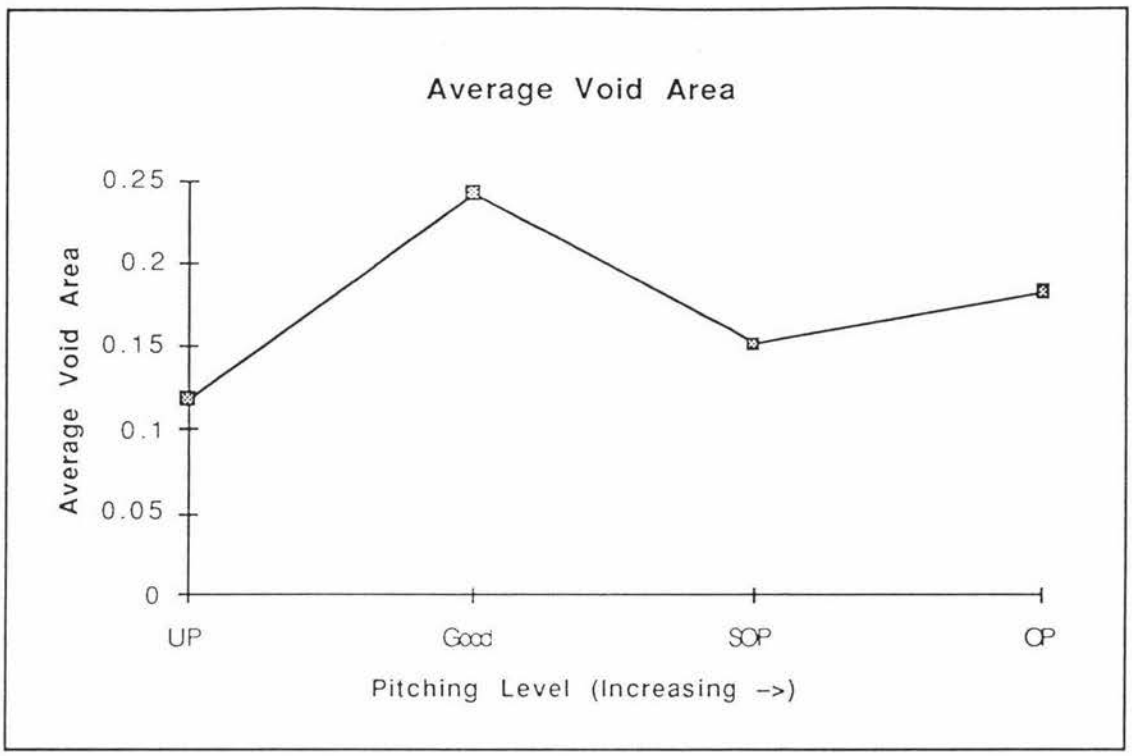


Figure 4.8 - Average Void Area vs Pitching Level

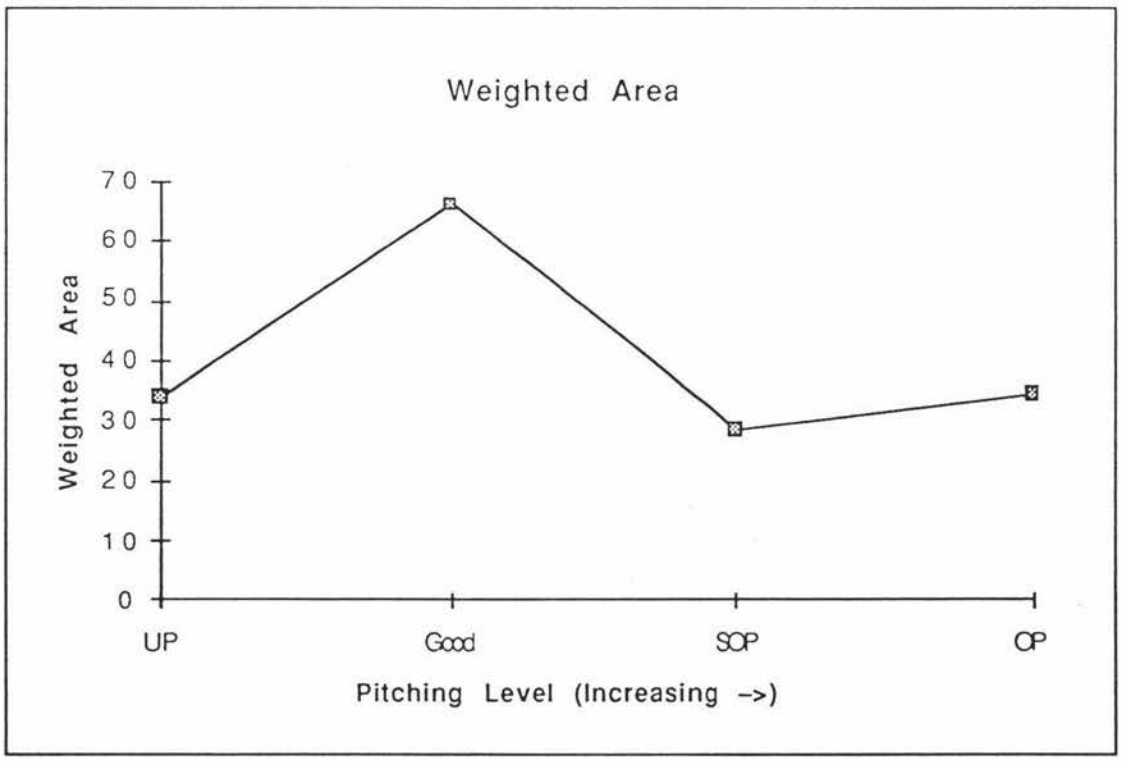


Figure 4.9 - Weighted Area vs Pitching Level

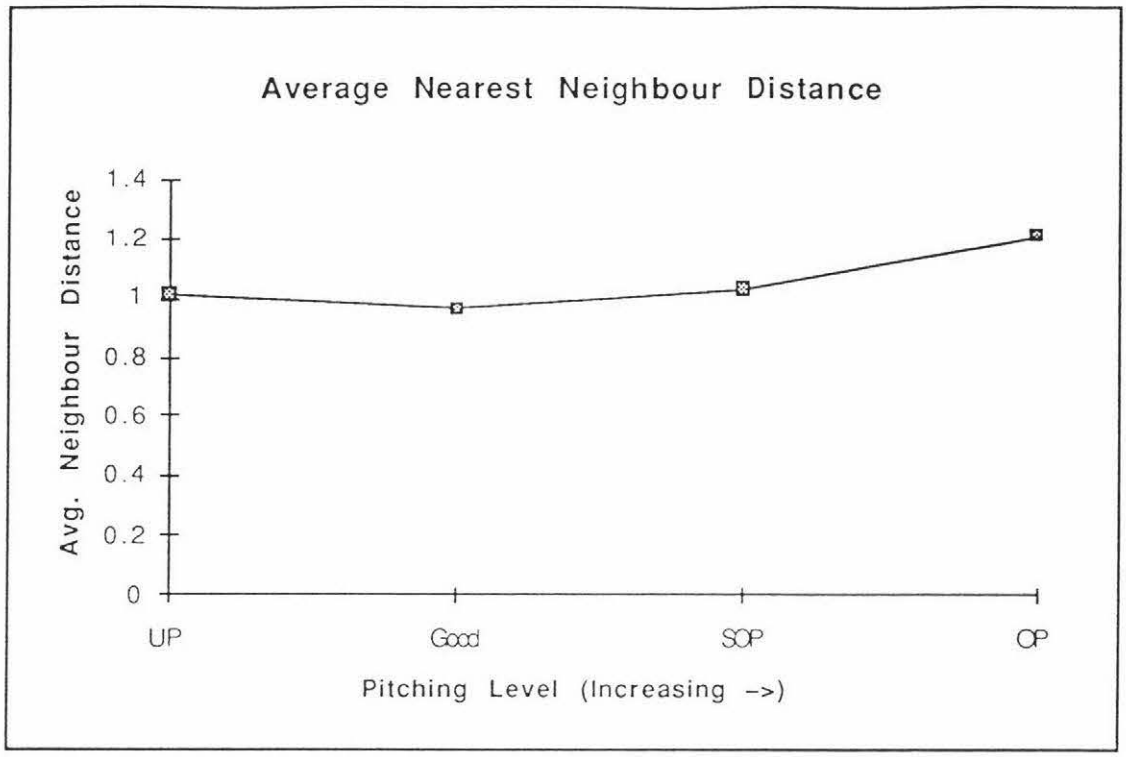


Figure 4.10 - Average Neighbour Distance vs Pitching Level

From the above figures it can be seen that some of the image features vary systematically according to the pitching level. These features are inertia, homogeneity and distance factor. It remains to be seen if these trends are confirmed for samples with accurately known pitching levels.

4.2.1.2 *Area and Spatial Distributions*

For each of the anode samples the area and nearest neighbour distance distributions were measured. A cumulative histogram was constructed using "Excel" for each area distribution using the log of the area since there are many more small voids than large voids. A cumulative histogram was also constructed for the nearest neighbour distance distribution.

Kolgomorov-Smirnov (KS) tests were used to determine if the area and nearest neighbour distance distributions for each of the pitching levels are different from each other, that is the pattern of voids in the two samples is different.

For the pitching level data the cumulative log area histograms are plotted in figure 4.11. The area distributions have been normalised according to the number of voids detected in the image.

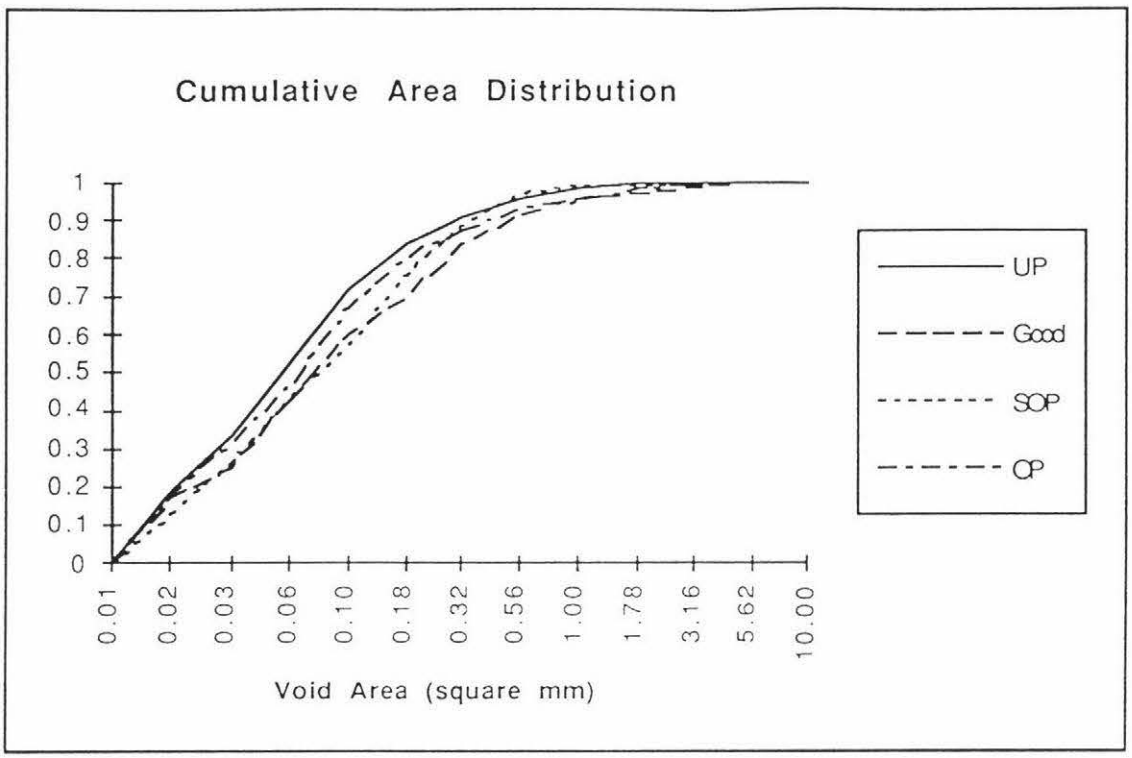


Figure 4.11 - Cumulative Area Distributions

KS tests were performed at a 99% level of confidence between the four pitching levels for the area distribution and results are shown in figure 4.12.

	UP	Good	SOP	OP
UP	Same	Same	Same	Same
Good	Same	Same	Same	Same
SOP	Same	Same	Same	Same
OP	Same	Same	Same	Same

Figure 4.12 - KS Results for the Area Distribution at a Confidence Level of 99%

This table shows that the area distributions for all pitching levels at a resolution of 10 pixels/mm are the same. The KS tests were repeated for a confidence level of 95% and the underpitched anode was found have a different area distribution than the good anode and the slightly overpitched anode. The underpitched anode was the same statistically as the overpitched anode. This is shown in figure 4.13.

	UP	Good	SOP	OP
UP	Same	Different	Different	Same
Good	Different	Same	Same	Same
SOP	Different	Same	Same	Same
OP	Same	Same	Same	Same

Figure 4.13 - KS Results for the Area Distribution at a Confidence Level of 95%

The KS tests themselves did not have to be repeated for the 95% confidence level as the KS ratios for the 99% confidence level tests could have been multiplied by a factor of 1.63/1.36 (≈1.2). This adjustment factor is the ratio of the KS critical value for the 99% confidence level divided by the KS critical value for the 95% confidence level (See Appendix C). Any KS ratios now over 1.0 indicate that particular KS test has now become significant.

For the pitching level data the cumulative nearest neighbour distance histograms are plotted in Figure 4.14. The nearest neighbour distance distributions have been normalised according to the number of nearest neighbour measurements made in the image.

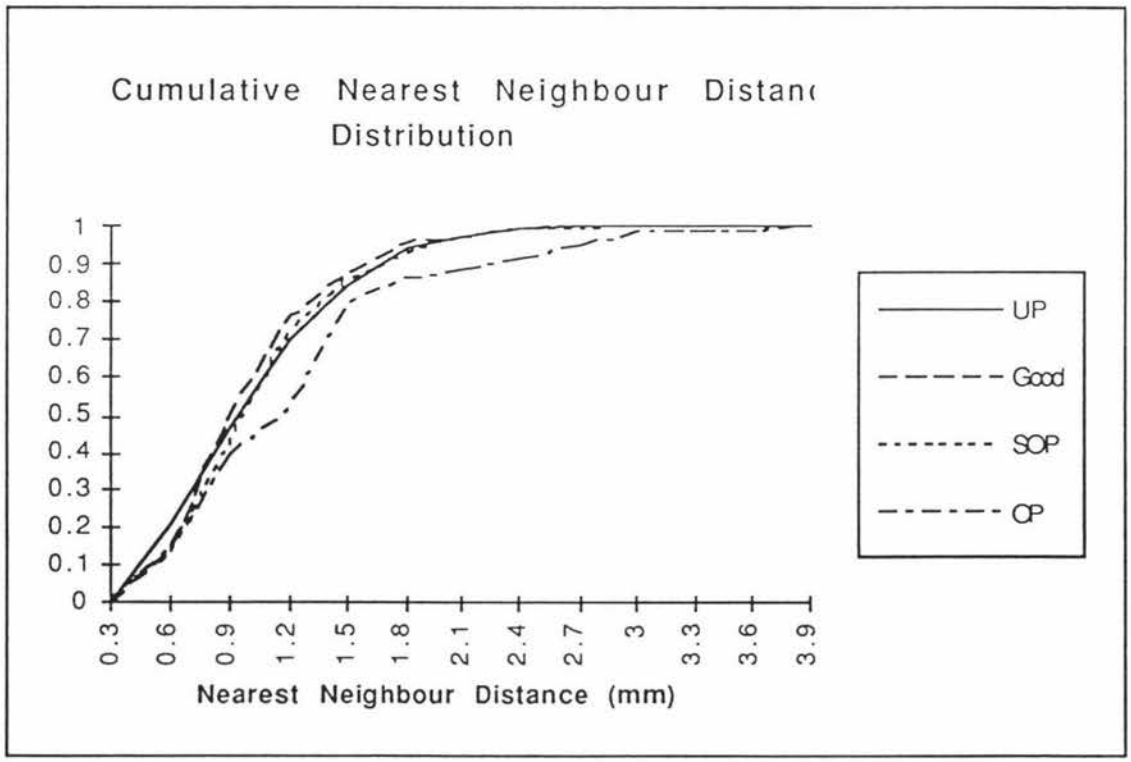


Figure 4.14 - Cumulative Nearest Neighbour Distance Distribution

KS tests were performed between the four pitching levels for the nearest neighbour distance distribution and the following table resulted in figure 4.15.

	UP	Good	SOP	OP
UP	Same	Same	Same	Same
Good	Same	Same	Same	Different
SOP	Same	Same	Same	Same
OP	Same	Different	Same	Same

Figure 4.15 - KS Results for the Spatial Distribution at a Confidence Level of 99%

The good anode has a significantly different spatial distribution from the overpitched one at a 99% level of confidence. The KS ratio for this test was 1.05 so the spatial distributions were only marginally different. The KS tests were then repeated for a confidence level of 95% and the overpitched anode was also found to be different from the good and the slightly overpitched anodes. The KS ratio for the overpitched/underpitched test was 0.98 so if an even lower confidence level had been used then the spatial distribution of the overpitched anode would have been significantly different from all the other pitching levels. This would allow discrimination of the overpitched anode from the others. This is shown in figure 4.16.

	UP	Good	SOP	OP
UP	Same	Same	Same	Same(0.98)
Good	Same	Same	Same	Different
SOP	Same	Same	Same	Different
OP	Same(0.98)	Different	Different	Same

Figure 4.16 - KS Results for the Spatial Distribution at a Confidence Level of 95%

4.2.2 40 pixels/mm

4.2.2.1 *Image Features*

Five images of a resolution of 40 pixels/mm were captured of each of the samples of different pitching levels. These images were all 512 x 768 in size and covered an area of 242.4 mm^2. This provided five images at each of the four pitching levels to work with.

For each of the images the image features were calculated using "Image" and "Excel" and the area and nearest neighbour distance distributions were also noted. Graphs of the image features as a function of pitching level are shown in Figures 4.17 to 4.26 below. These graphs show the mean and upper and lower bounds (ie ±3 standard deviations) of the five features after the largest and smallest feature values for the five images have been discarded. This removes the outliers.

This analysis will indicate the variability of each feature. It will also show if there are any statistical differences between the features for different pitching levels.

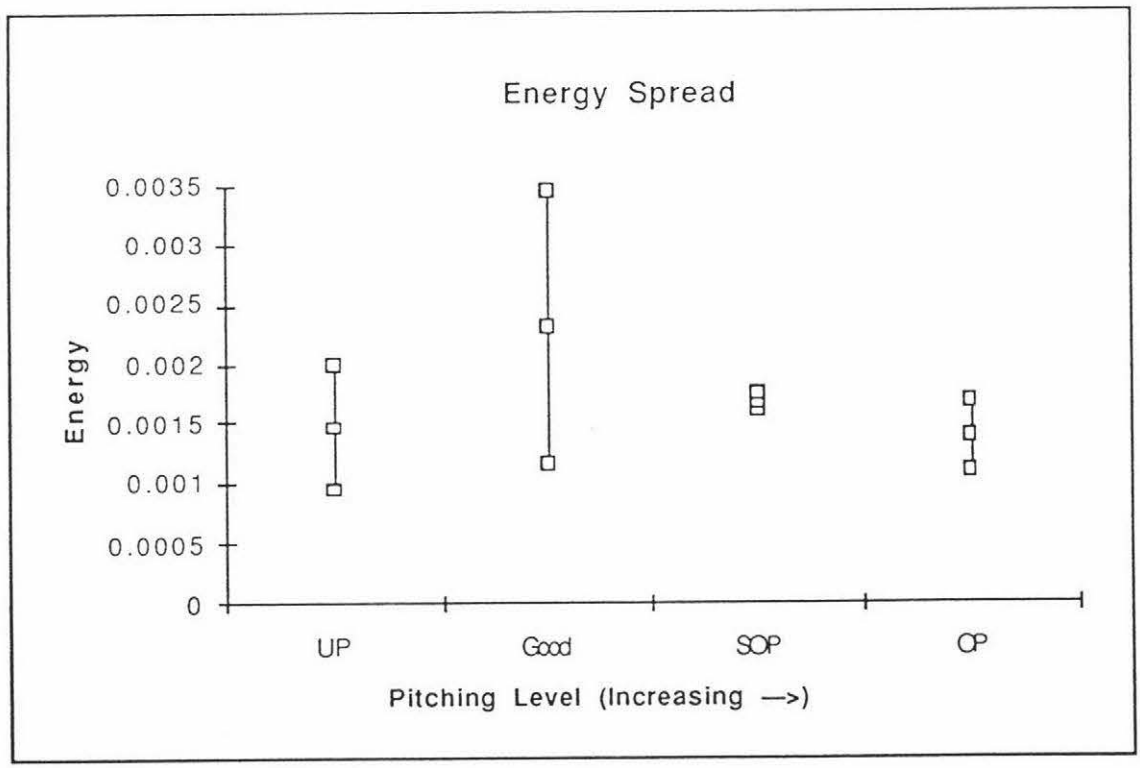


Figure 4.17 - Energy Spread vs Pitching Level

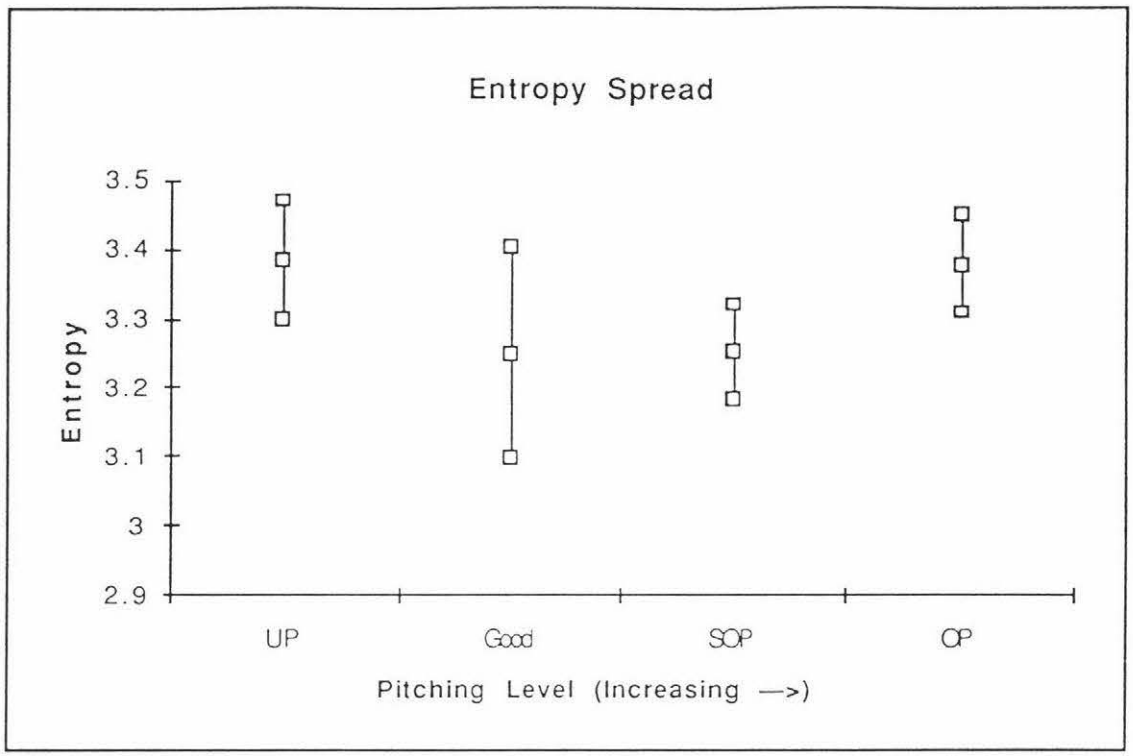


Figure 4.18 - Entropy Spread vs Pitching Level

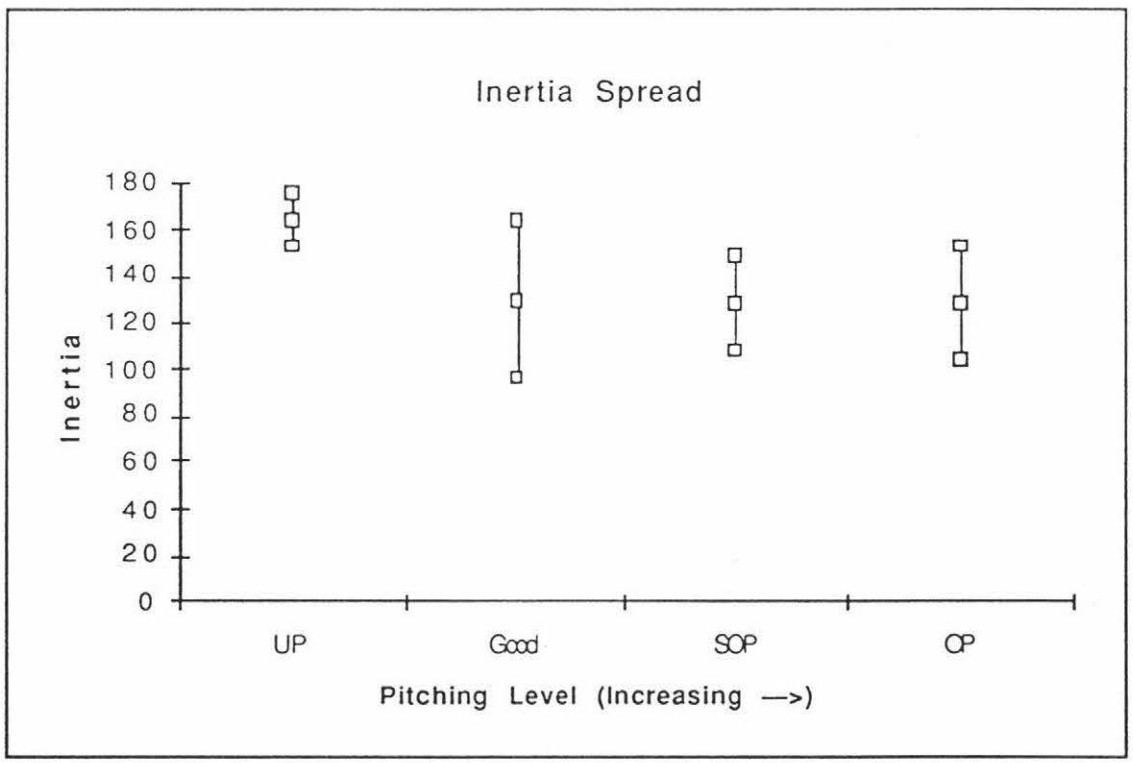


Figure 4.19 - Inertia Spread vs Pitching Level

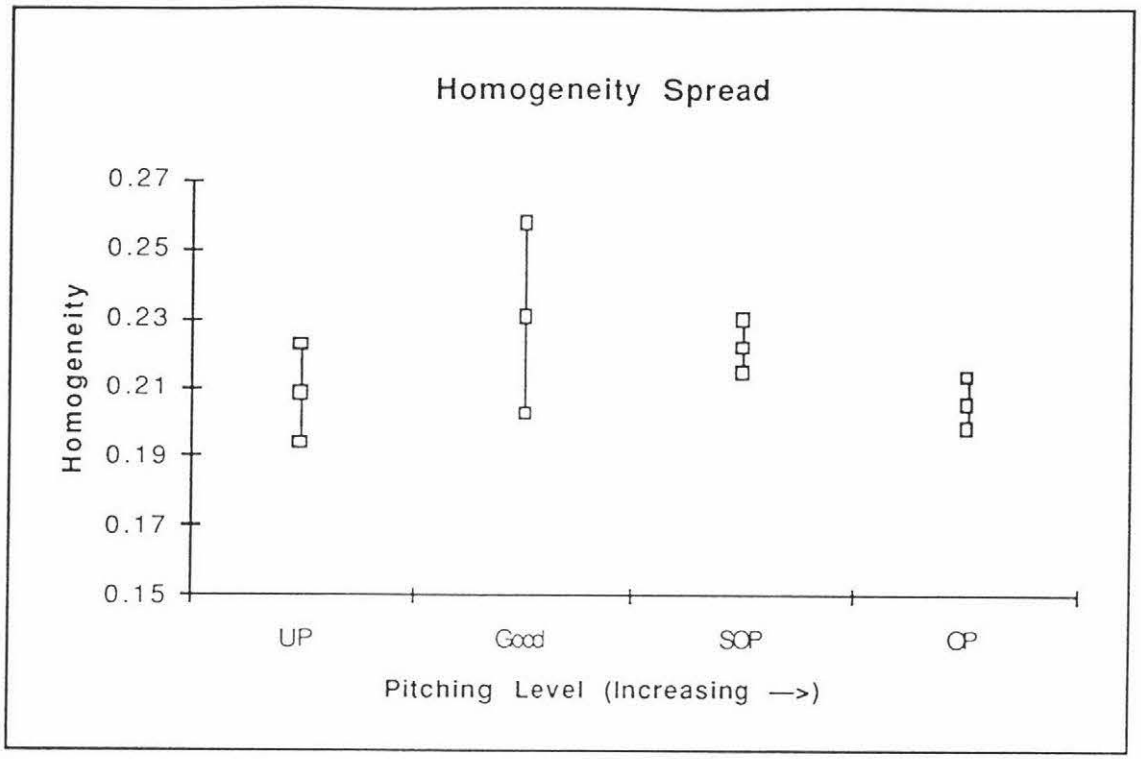


Figure 4.20 - Homogeneity Spread vs Pitching Level

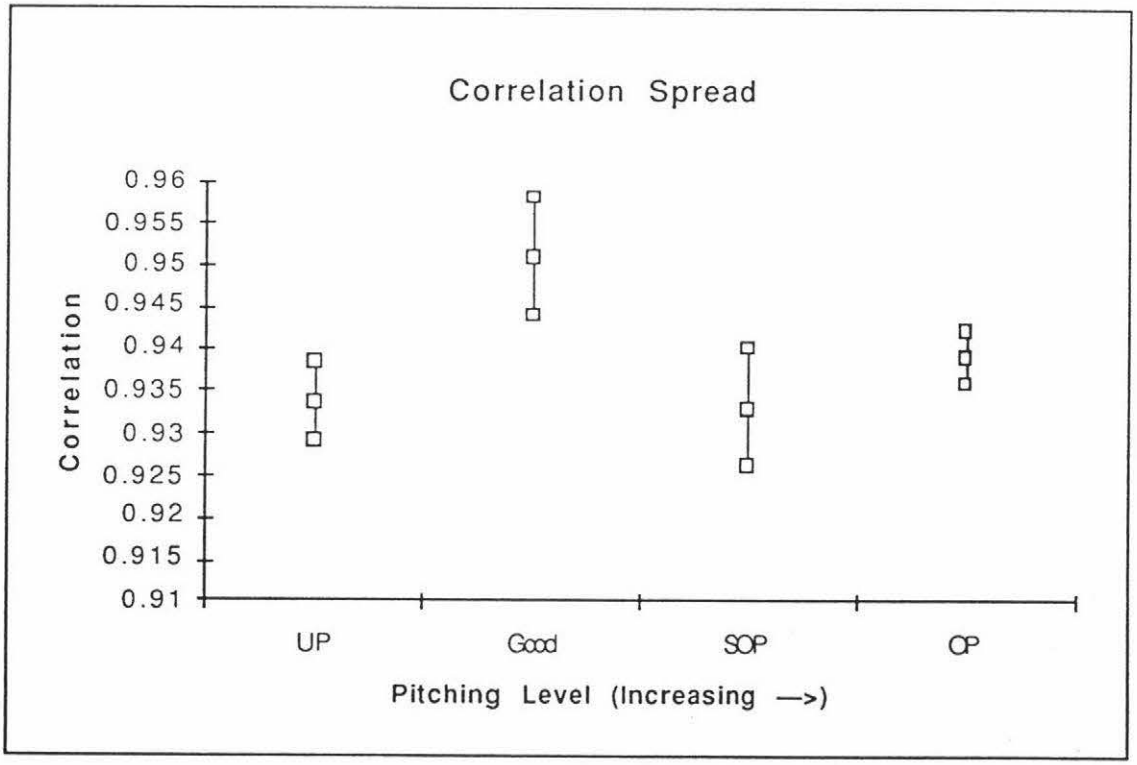


Figure 4.21 - Correlation Spread vs Pitching Level

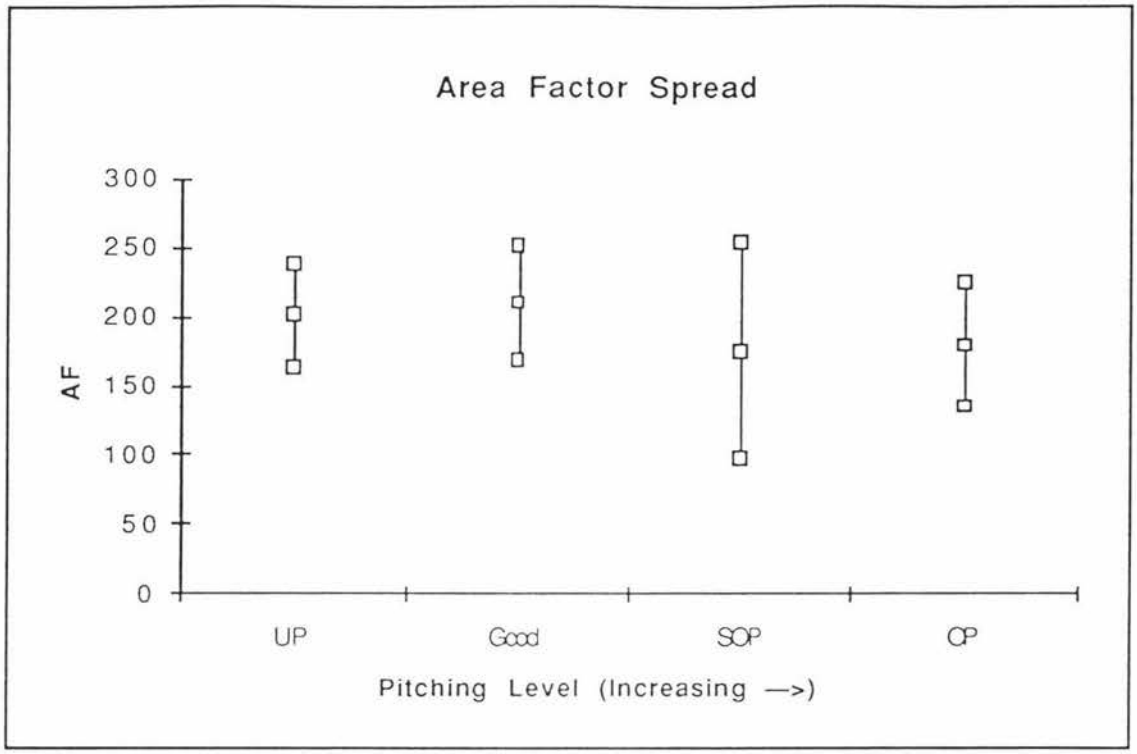


Figure 4.22 - Area Factor Spread vs Pitching Level

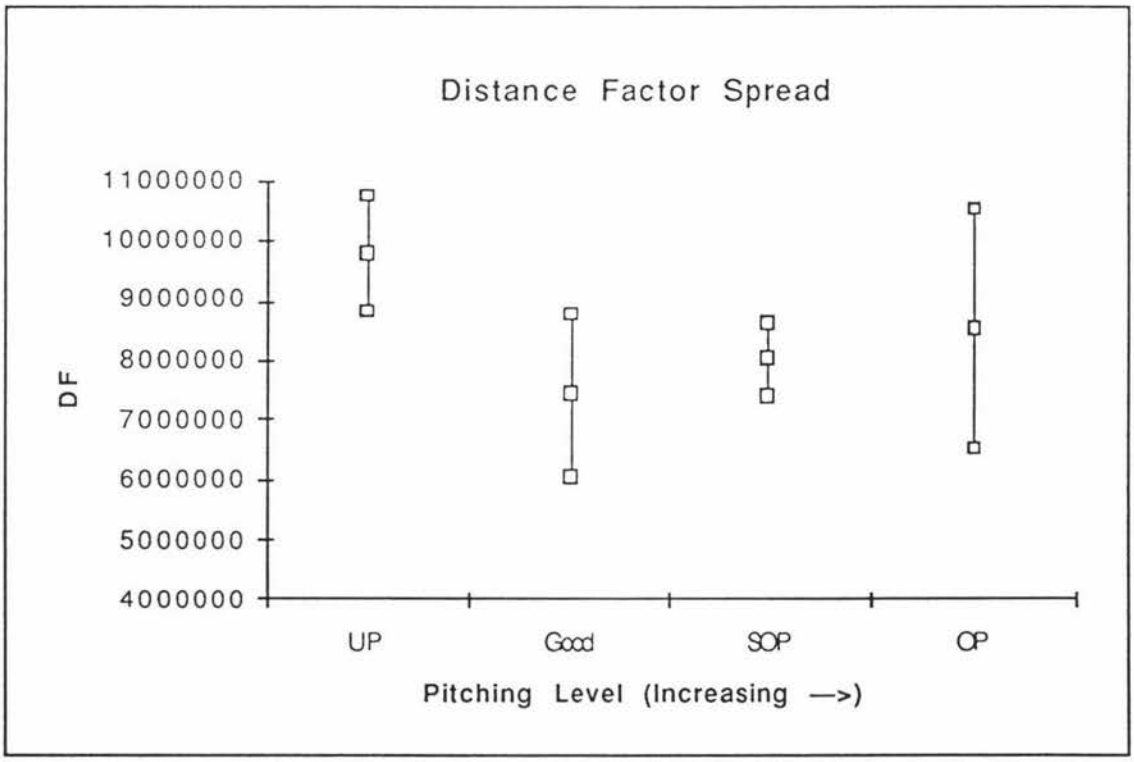


Figure 4.23 - Distance Factor Spread vs Pitching Level

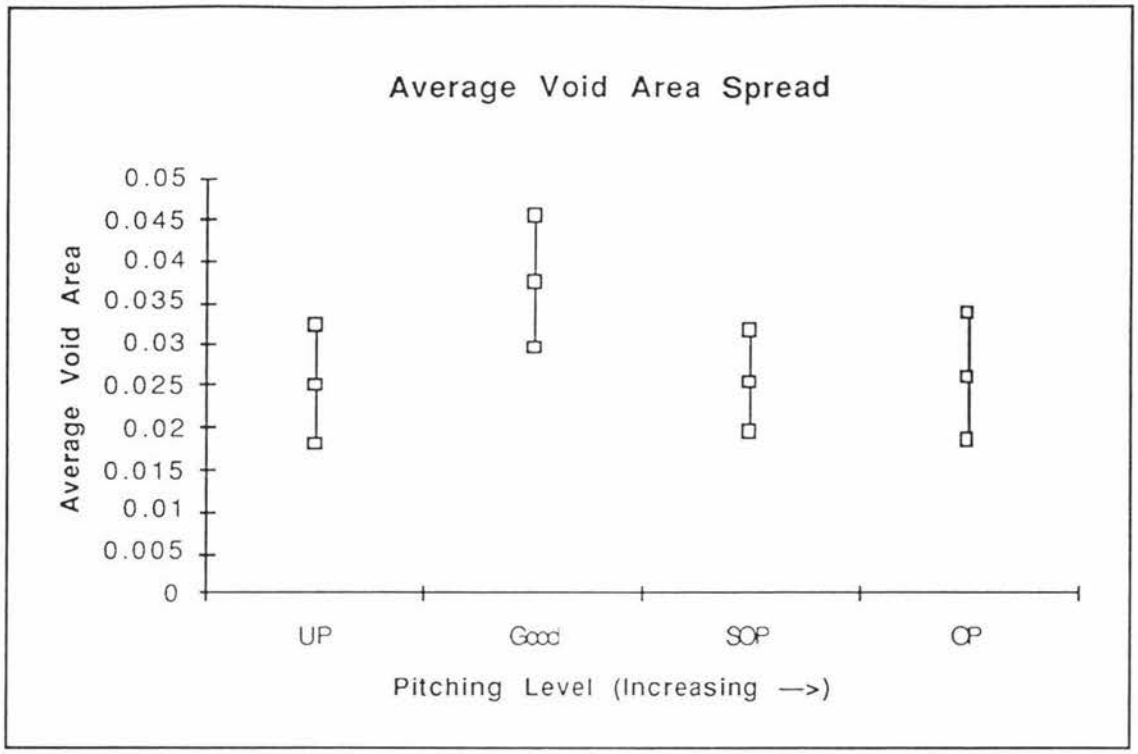


Figure 4.24 - Average Void Area Spread vs Pitching Level

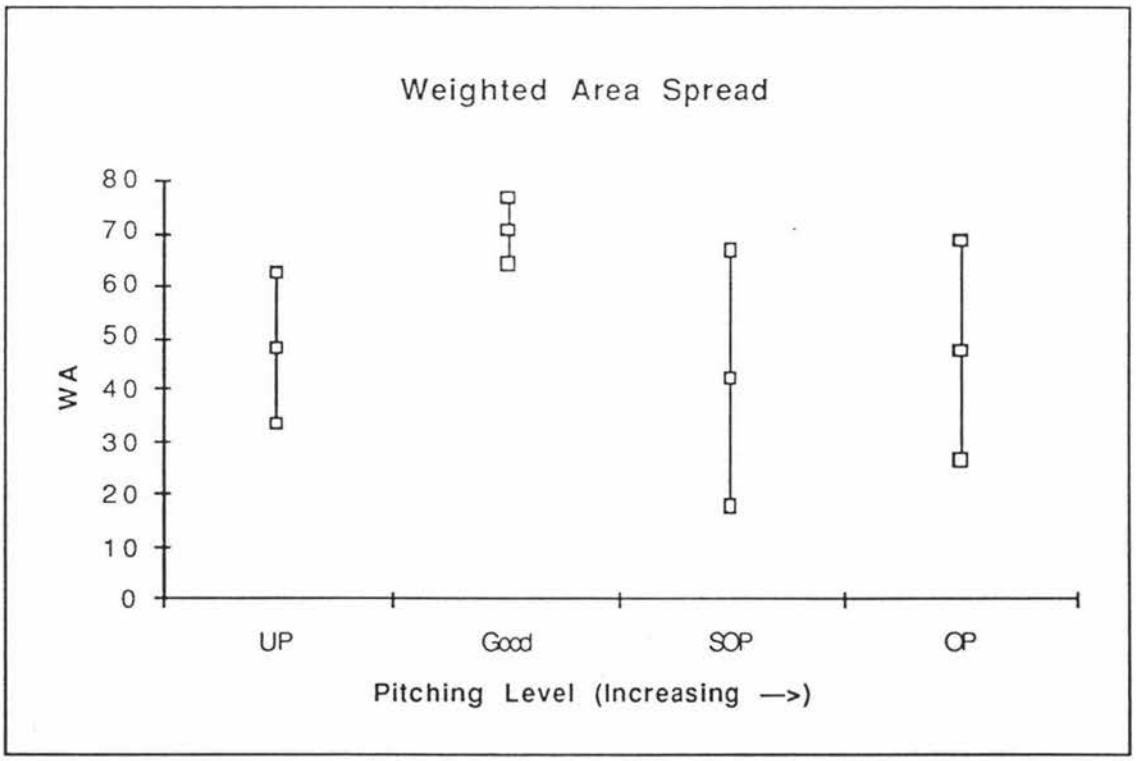


Figure 4.25 - Weighted Area Spread vs Pitching Level

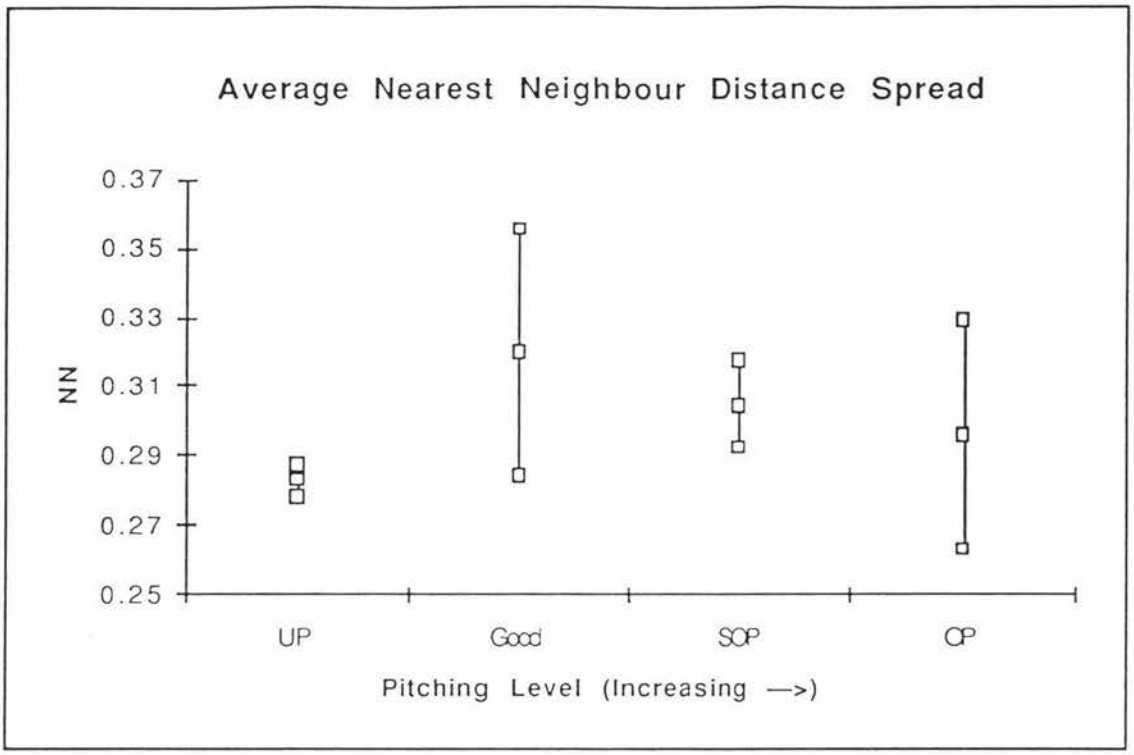


Figure 4.26 - Average Nearest Neighbour Distance Spread vs Pitching Level

From the above figures there exist no trends for the image features as a function of the pitching level although some statistical differences between differing pitching levels are evident.

For the inertia, distance factor, and the average nearest neighbour distance features the underpitched anode is statistically different from the slightly overpitched anode. The weighted area feature for the underpitched anode is statistically different from the good anode and the correlation feature for the good anode is statistically different from all the other pitching levels. These results could be useful in discriminating the anodes from one another.

4.2.2.2 *Area and Spatial Distributions*

For each of the images the area and nearest neighbour distance distributions were noted. The Kolgomorov-Smirnov (KS) tests were again performed on the area and nearest neighbour distance (spatial) distributions to see if the distributions for differing pitching levels were different.

As five images were captured for each pitching level KS tests were performed to see if the area and spatial distributions were the same for images of the same pitching level as well as seeing if the distributions were different for images of differing pitching level. The

results of the KS tests at a 99% level of confidence for the area distribution are shown in Figure 4.27 below.

	U1	U2	U3	U4	U5	G1	G2	G3	G4	G5	S1	S2	S3	S4	S5	O1	O2	O3	O4	O5
U1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
U2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
U3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
U4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1
U5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1
G1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0
G2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
G5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
S1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
S2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
S3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
S4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S5	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
O4	1	1	1	1	1	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0
O5	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Figure 4.27 - KS Results for the Area Distribution

	U1	U2	U3	U4	U5	G1	G2	G3	G4	G5	S1	S2	S3	S4	S5	O1	O2	O3	O4	O5
U1	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0
U2	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	0	0
U3	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0
U4	0	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
U5	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0
G1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G2	1	1	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1
G3	1	1	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1
G4	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
G5	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1
S1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
S2	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
S3	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
S4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
S5	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
O1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O2	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
O3	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0
O4	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
O5	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0

Figure 4.28 - KS Results for the Spatial Distribution

Where U = Underpitched

G = Good

S = Slightly Overpitched

O = Overpitched

and '0' signifies the distributions are the same

'1' signifies the distributions are different

Figure 4.27 shows that the area distribution for each image tends to be the same as any other image. Three of the images stand out as being different: G1, O4 and O5. This is because these images have more larger or smaller voids. Overall, the area distributions of the voids does not vary with pitching level in a consistent manner.

The results of the KS tests at a 99% level of confidence for the nearest neighbour distance distributions are shown in Figure 4.28.

For the spatial distribution, all of the images from the same pitching level are all the same but images from different pitching levels are not all different. Some of the anode types differ markedly. For example the images from the good anode are nearly all different from the images from the underpitched anode. These results can be summarised in figure 4.29 where the number in each cell corresponds to the number of differences (1's) between pitching levels divided by 25 (the total number of KS tests performed between each pitching level).

	UP	Good	SOP	OP
UP	0	0.88	0.48	0.24
Good	0.88	0	0.28	0.52
SOP	0.48	0.28	0	0.12
OP	0.24	0.52	0.12	0

Figure 4.29 - Summarised KS Results for the Spatial Distribution

It can be seen from figure 4.29 that the higher the number then the greater the difference between the corresponding spatial distributions. This does not tell what values are significant or not. Is the value of 0.12 for the SOP/OP test significant at a 99% level of confidence? To answer this question, the area and nearest neighbour distance distributions for the five images of each pitching level were pooled together. This will result in one area and nearest neighbour distance distribution for each pitching level.

The results of this analysis for a 99% level of confidence is shown in figures 4.30 and 4.31.

	UP	Good	SOP	OP
UP	Same	Different	Different	Different
Good	Different	Same	Same	Same
SOP	Different	Same	Same	Same
OP	Different	Same	Same	Same

Figure 4.30 - Pooled KS Results for the Area Distribution at a 99% Level of Confidence

	UP	Good	SOP	OP
UP	Same	Different	Different	Different
Good	Different	Same	Different	Different
SOP	Different	Different	Same	Same
OP	Different	Different	Same	Same

Figure 4.31 - Pooled KS Results for the Spatial Distribution at a 99% Level of Confidence

Figures 4.30 shows that the area distribution for the underpitched anode varies significantly from the other anodes. This shows that the KS test for the area distributions could be used to discriminate underpitched anodes.

Figure 4.31 shows that the spatial distributions for the pitching levels are all significantly different from each other at the 99% level of confidence, except for the slightly overpitched anode with the overpitched anode. The KS ratio for this test was 0.98 so the two anodes are almost significantly different from each other. This KS ratio of 0.98 could be expected as there is only a little difference between the slightly overpitched and overpitched anodes with respect to the amount of pitch present. This ratio of 0.98 also corresponds to the number 3 in figure 4.29, so the assumption that the higher the number then the more different the pitching levels are seems valid.

The KS ratios were adjusted to a 95% confidence level and the results are shown in figures 4.32 and 4.33.

	UP	Good	SOP	OP
UP	Same	Different	Different	Different
Good	Different	Same	Same	Different
SOP	Different	Same	Same	Different
OP	Different	Different	Different	Same

Figure 4.32 - Pooled KS Results for the Area Distribution at a 95% Level of Confidence

	UP	Good	SOP	OP
UP	Same	Different	Different	Different
Good	Different	Same	Different	Different
SOP	Different	Different	Same	Different
OP	Different	Different	Different	Same

Figure 4.33 - Pooled KS Results for the Spatial Distribution at a 95% Level of Confidence

Lowering the confidence level increases the discriminating power of the KS test. The overpitched anode now has a significantly different area distribution from all the other anodes. The KS test at this confidence level can only discriminate overpitched or underpitched anodes from good or slightly overpitched ones.

The lower confidence level has increased the KS ratio of the overpitched/slightly overpitched KS test from 0.98 to 1.18, making it significant. This now means any anode of a particular pitching level has a significantly different spatial distribution from any anode of a different pitching level, at a 95% confidence level.

Overall the KS test at 40 pixels/mm resolution indicates that the spatial distribution of the voids varies according to pitching level.

4.3 Changes in Forming Conditions

4.3.1 10 pixels/mm

4.3.1.1 *Image Features*

Images of a resolution of 10 pixels/mm were captured of each of the prepared samples formed under differing conditions. At this resolution only one image was able to be captured for each forming condition. These images were each 512 x 512 in size, covering an area of 2324.2 mm^2. This provided 6 images, one for each set of forming conditions, to work with.

The image features of each image were calculated using "Image" and "Excel", and the area and nearest neighbour distance distributions were also noted. Plots of the image features as a function of the forming conditions are shown in Figures 4.34 to 4.43 below.

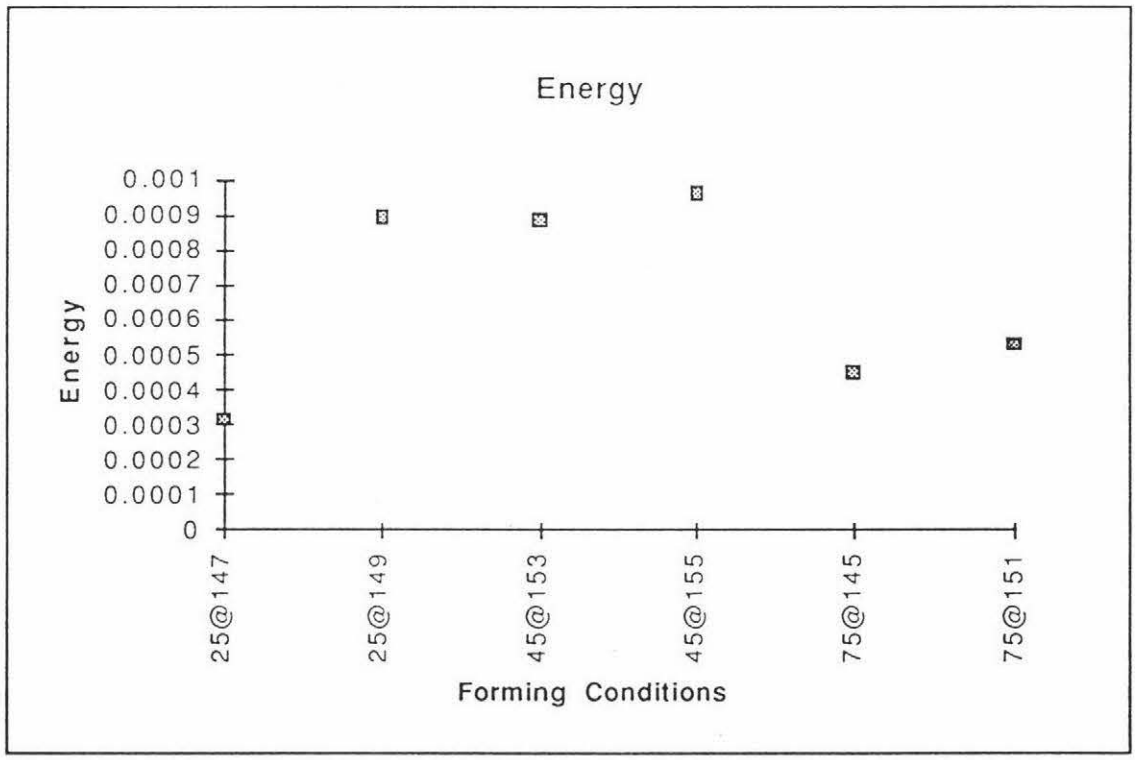


Figure 4.34 - Energy vs Forming Conditions

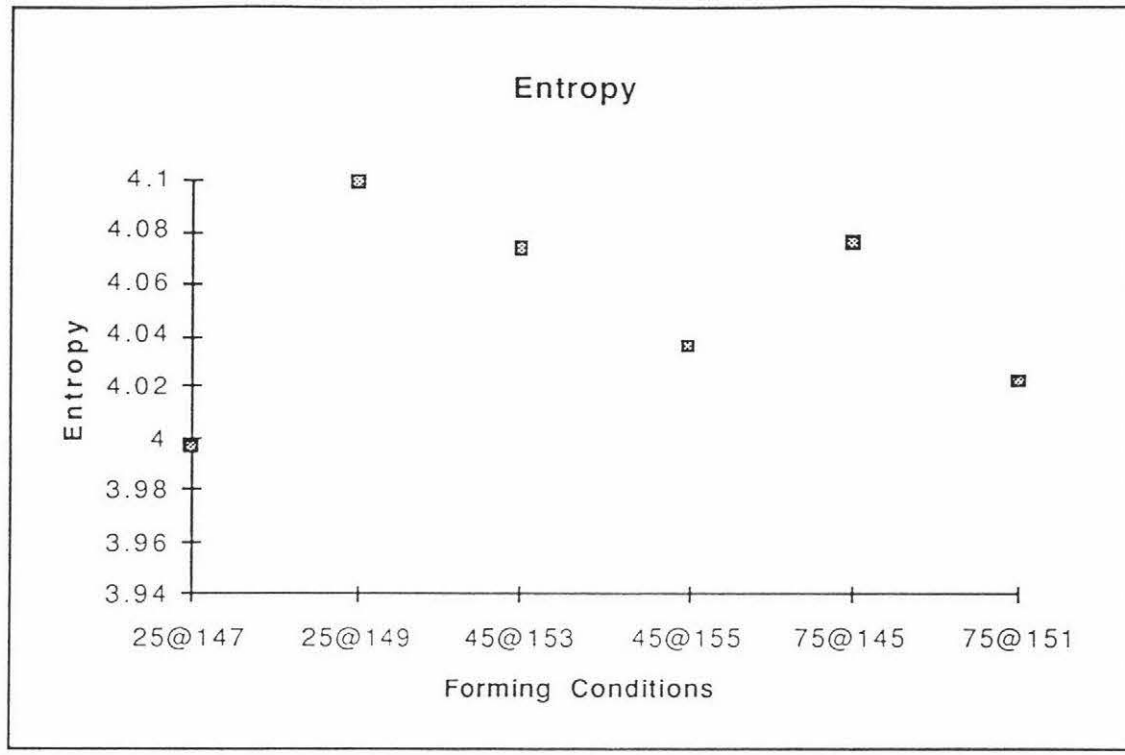


Figure 4.35 - Entropy vs Forming Conditions

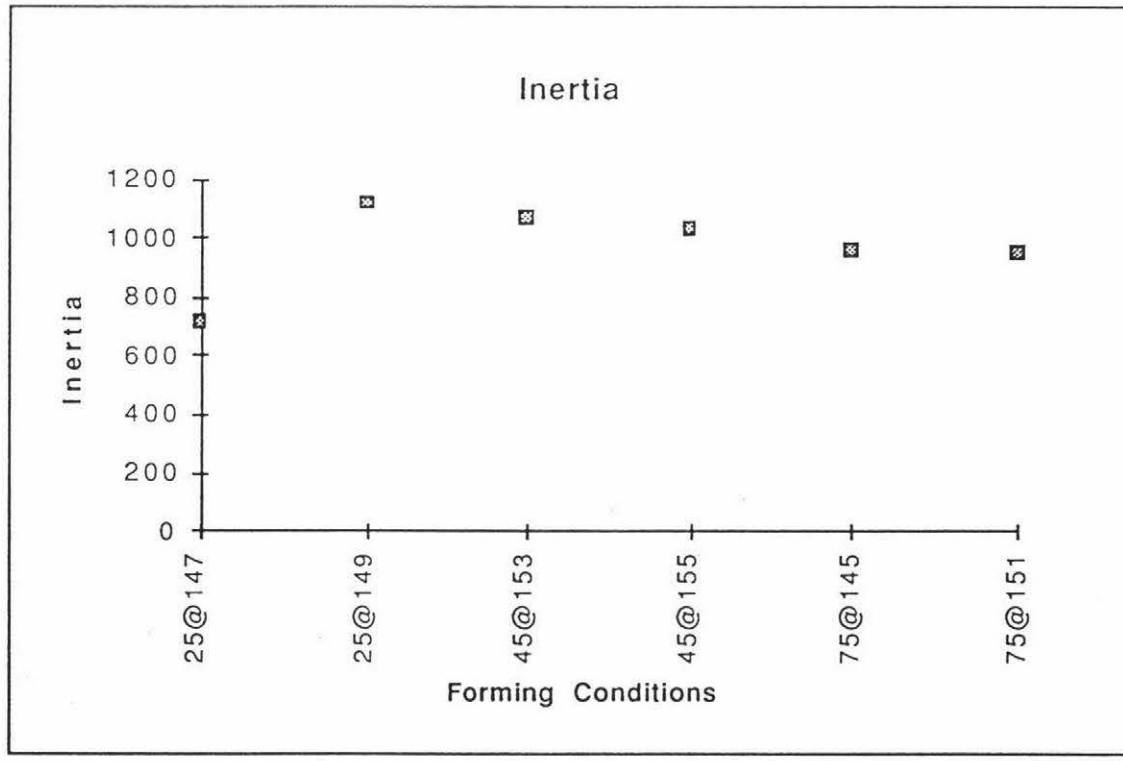


Figure 4.36 - Inertia vs Forming Conditions

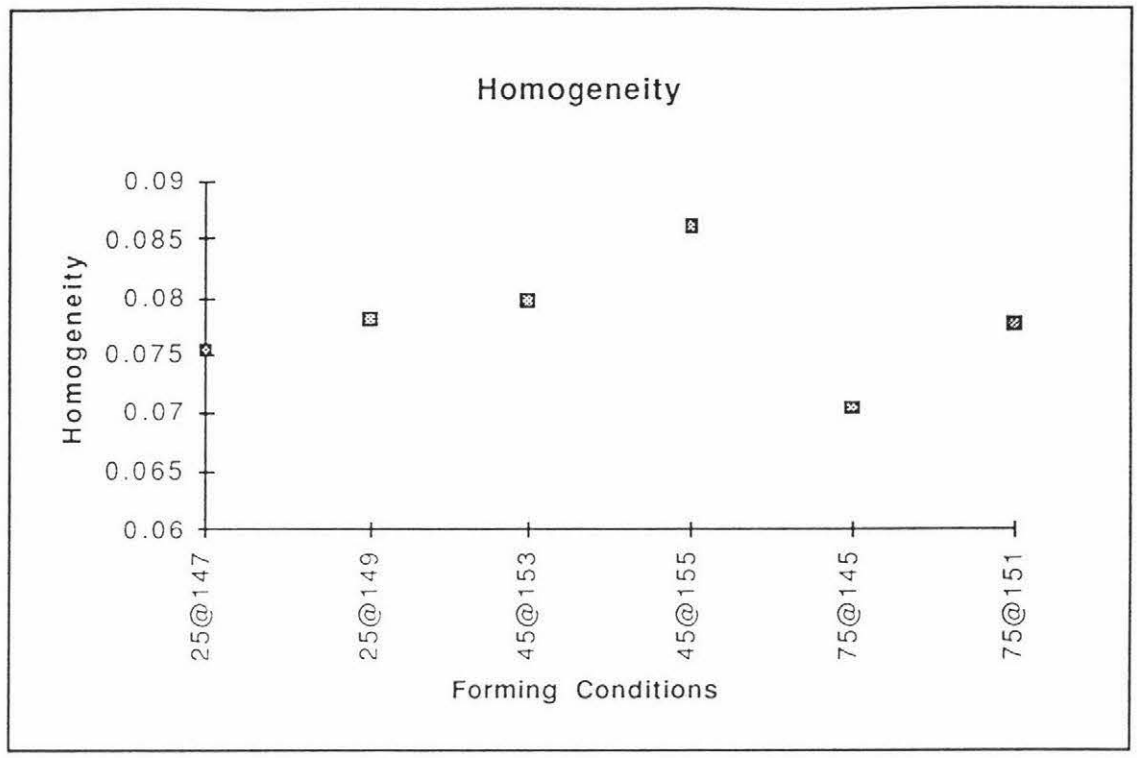


Figure 4.37 - Homogeneity vs Forming Conditions

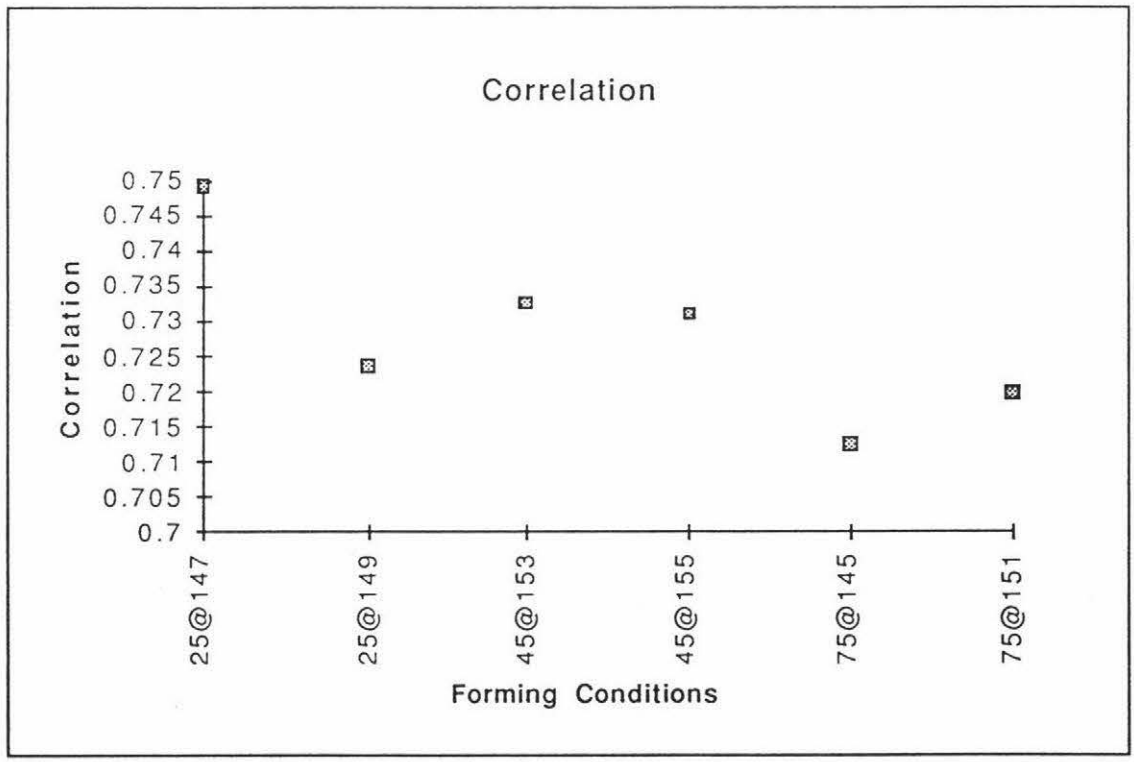


Figure 4.38 - Correlation vs Forming Conditions

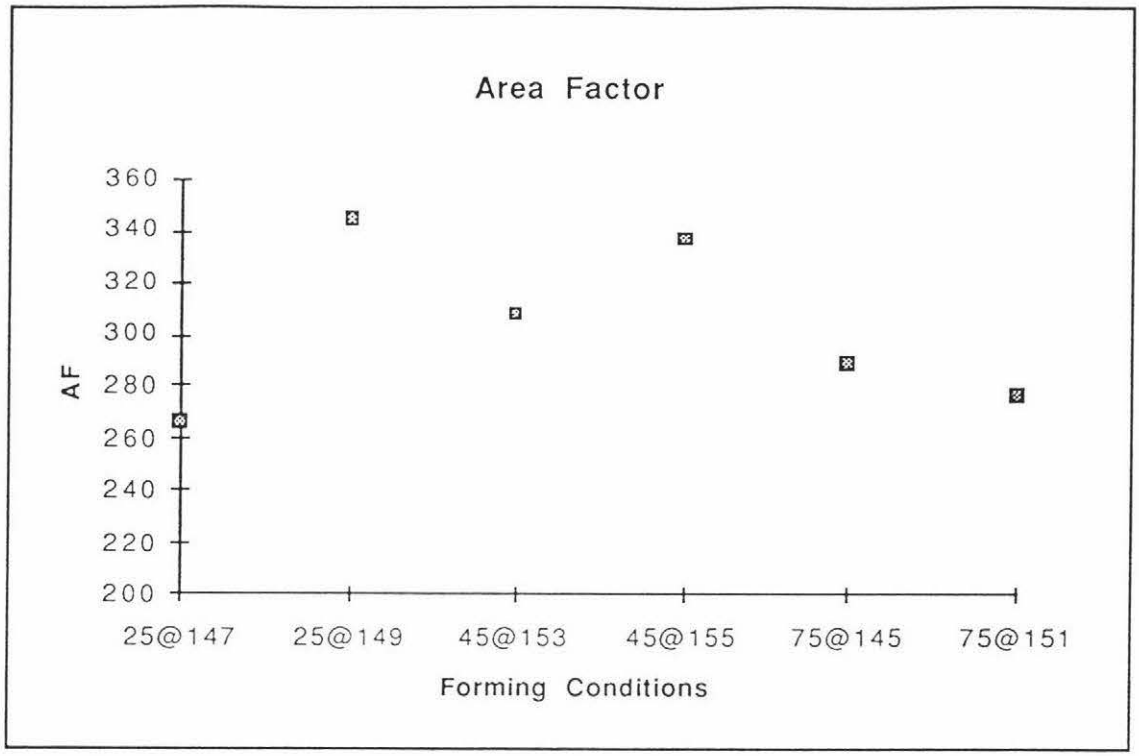


Figure 4.39 - Area Factor vs Forming Conditions

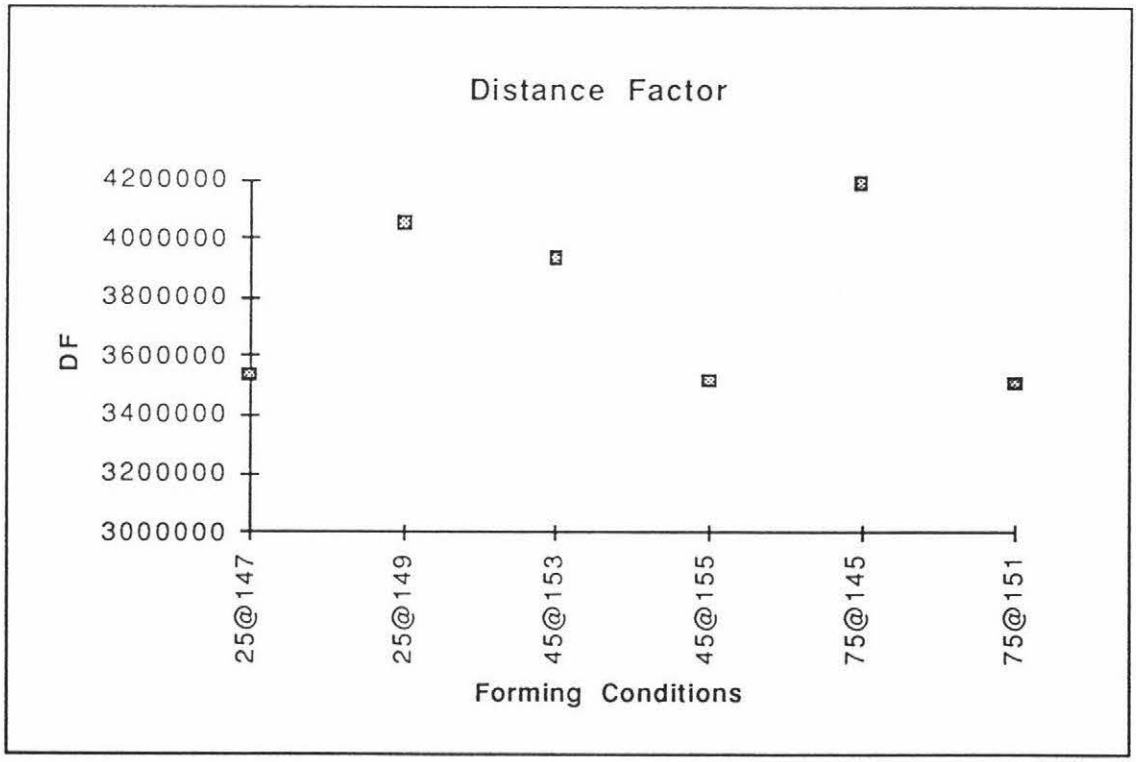


Figure 4.40 - Distance Factor vs Forming Conditions

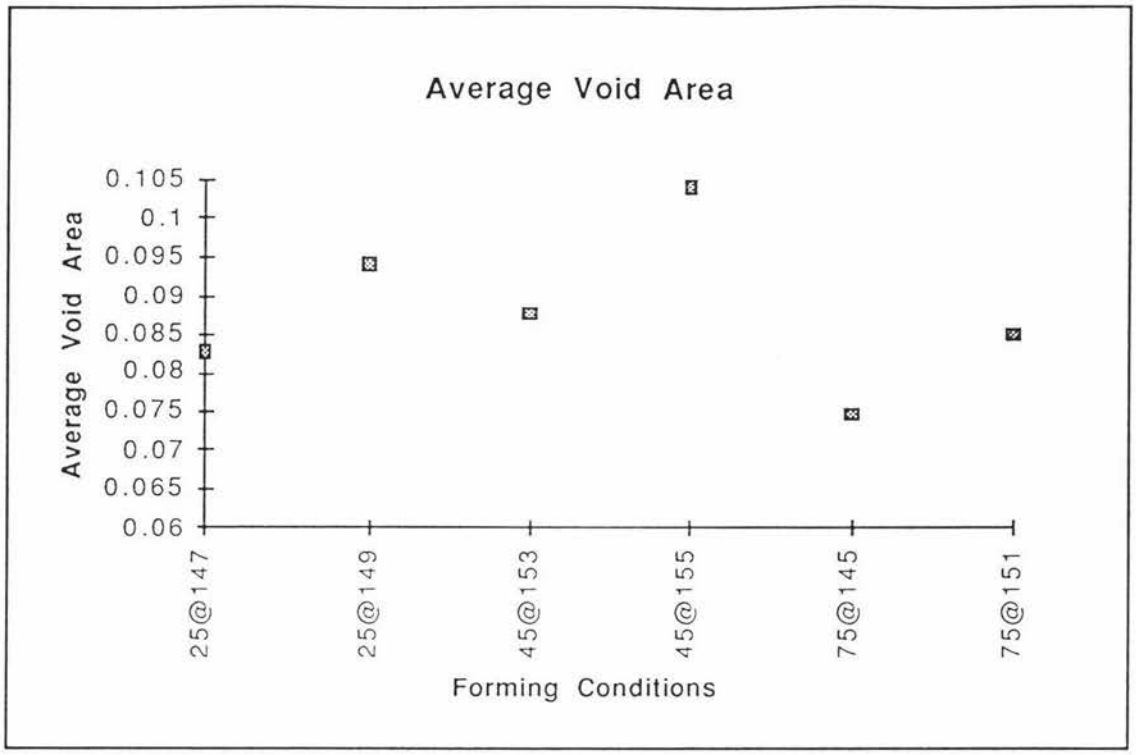


Figure 4.41 - Average Void Area vs Forming Conditions

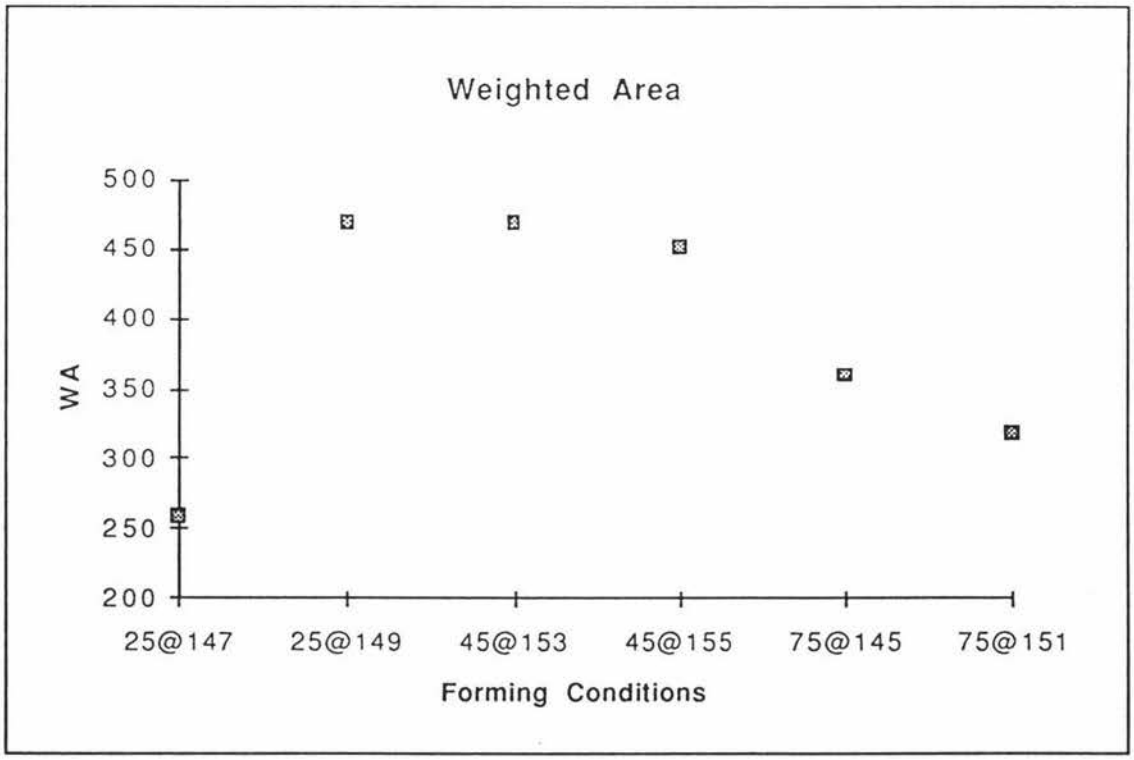


Figure 4.42 - Weighted Area vs Forming Conditions

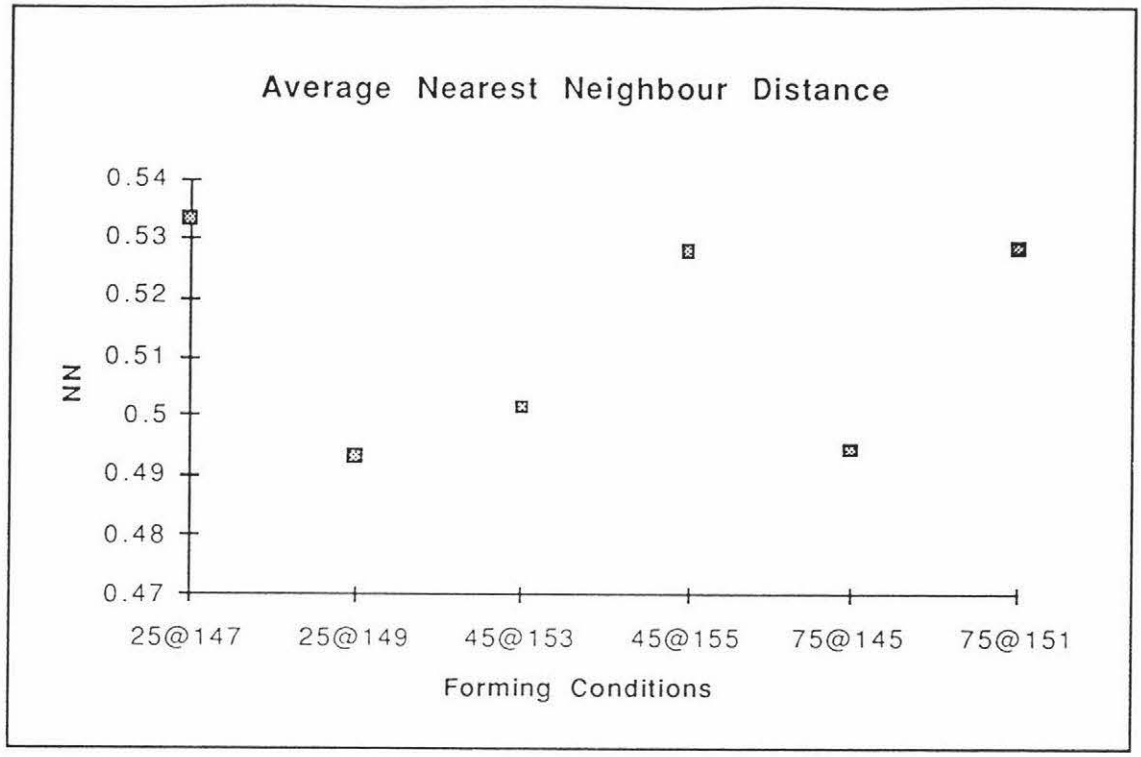


Figure 4.43 - Average Nearest Neighbour Distance vs Forming Conditions

For the energy, homogeneity, and average area plots the respective image features rises as the temperature rises for each of the three vibration times (25s, 45s, and 75s). This is a useful result as a high value in any of these features in a prepared sample could signify a higher forming temperature.

Only the correlation plot shows any trend with the vibration time. If the average correlation is taken for each vibration time then a downward trend is observed. This is a useful result as a high correlation value in a prepared sample could signify a small vibration time.

If the sample from the 25@147 forming condition was anomalous in some way, then the inertia and weighted area plots exhibit a systematic trend with the forming conditions.

4.3.1.2 *Area and Spatial Distributions*

For each of the forming conditions the area and nearest neighbour distance distributions were noted. This was done to perform the Kolgomorov-Smirnov (KS) tests as outlined earlier. These tests will determine if the area and spatial distributions of the voids vary with the forming conditions.

The KS tests were performed at a 99% confidence level between the six different forming conditions for the area distribution and the results are shown in figure 4.44 below.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Same	Same	Same	Same	Same
25@149	Same	Same	Same	Same	Same	Same
45@153	Same	Same	Same	Same	Same	Same
45@155	Same	Same	Same	Same	Same	Same
75@145	Same	Same	Same	Same	Same	Same
75@151	Same	Same	Same	Same	Same	Same

Figure 4.44 - KS Results for the Area Distribution at a 99% Level of Confidence

This table shows that the area distribution of the voids does not change at 10 pixels/mm resolution according to the forming conditions.

The KS tests were performed at a 99% confidence level between the six different forming conditions for the nearest neighbour distance distribution and the results are shown in figure 4.45 below.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Different	Different	Different	Different	Same
25@149	Different	Same	Same	Different	Same	Different
45@153	Different	Same	Same	Different	Same	Different
45@155	Different	Different	Different	Same	Different	Same
75@145	Different	Same	Same	Different	Same	Different
75@151	Same	Different	Different	Same	Different	Same

Figure 4.45 - KS Results for the Spatial Distribution at a 99% Level of Confidence

From the above data, the temperature change associated with each of the vibration times results in a significantly different spatial distributions. This could be useful in detecting forming temperature changes however there are no consistent differences between the spatial distributions of anodes with differing vibration times.

The KS ratios were adjusted for a 95% level of confidence and the results are shown in figures 4.46 and 4.47.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Same	Same	Same	Same	Different
25@149	Same	Same	Same	Same	Same	Same
45@153	Same	Same	Same	Same	Same	Same
45@155	Same	Same	Same	Same	Different	Same
75@145	Same	Same	Same	Different	Same	Same
75@151	Different	Same	Same	Same	Same	Same

Figure 4.46 - KS Results for the Area Distribution at a 95% Level of Confidence

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Different	Different	Different	Different	Same
25@149	Different	Same	Same	Different	Same	Different
45@153	Different	Same	Same	Different	Same	Different
45@155	Different	Different	Different	Same	Different	Same
75@145	Different	Same	Same	Different	Same	Different
75@151	Same	Different	Different	Same	Different	Same

Figure 4.47 - KS Results for the Spatial Distribution at a 95% Level of Confidence

The lower confidence level has made the area distributions of the 25@147 forming condition significantly different from the 75@151 forming condition, and the 45@155 forming condition is now significantly different from the 75@145 forming condition.

The lower confidence level has not changed the results for the spatial distributions.

4.3.2 40 pixels/mm

4.3.2.1 *Image Features*

Ten images were captured at 40 pixels/mm resolution for each of the different forming conditions. These images were all 512 x 768 in size and covered an area of 240.4 mm^2. This provided ten images per forming condition to analyse, a total of 60 images.

For each image, the image features were calculated using "Image" and "Excel" and the area and nearest neighbour distance distributions were also noted. Graphs of the image features as a function of the forming conditions are shown in Figures 4.48 to 4.57 below. The graphs show the mean and upper and lower bounds (ie ±3 standard deviations) after the two largest and two smallest features of the ten images have been discarded. This removes any outliers and possible anomalies from consideration.

This analysis will indicate the amount of variability each of the features has. It will also show if there are any statistical differences between the features for different forming conditions.

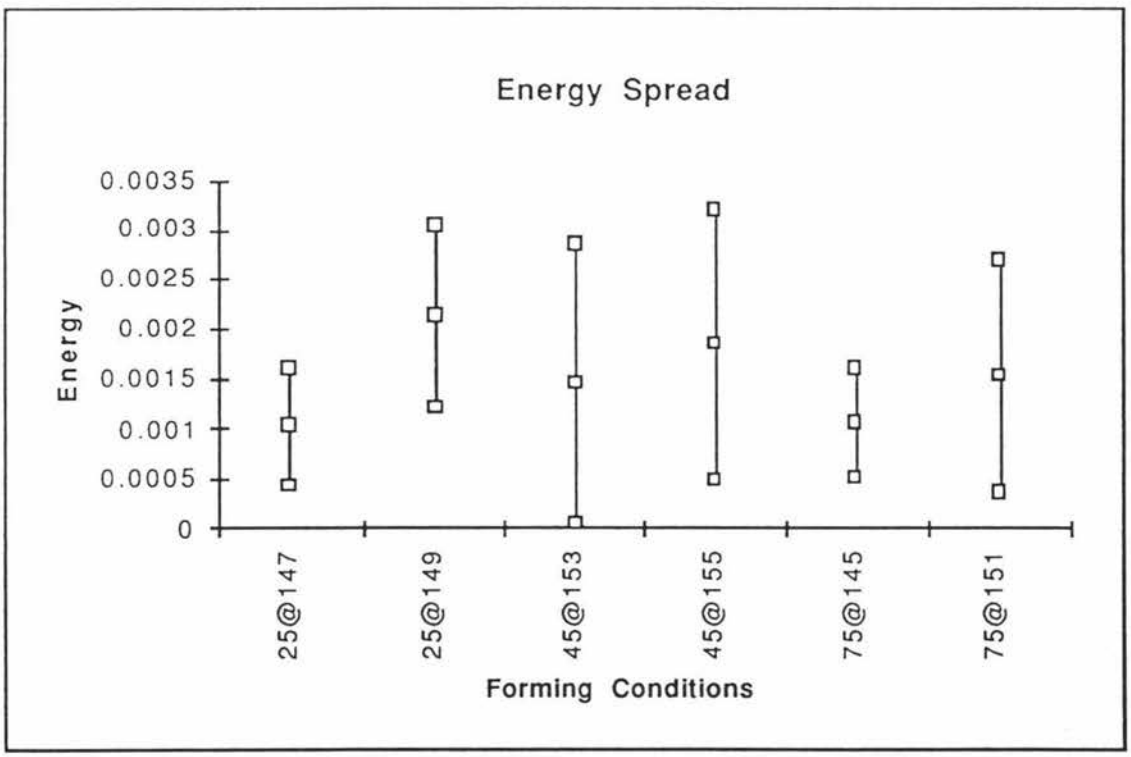


Figure 4.48 - Energy Spread vs Forming Conditions

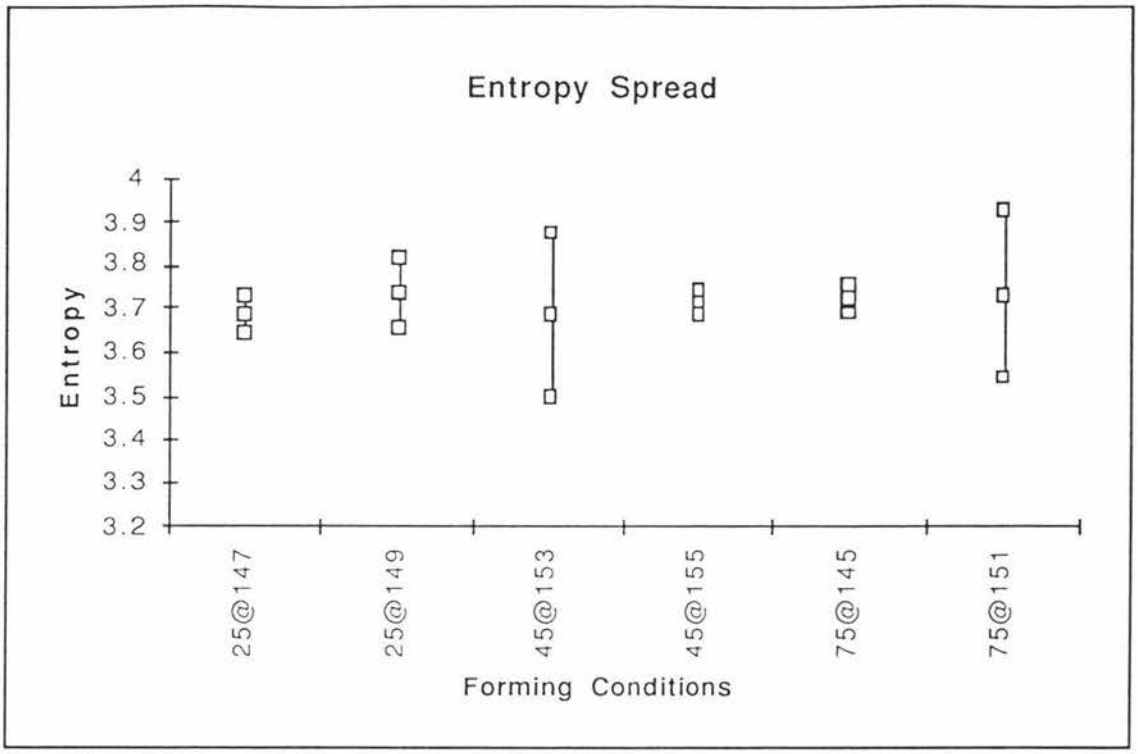


Figure 4.49 - Entropy Spread vs Forming Conditions

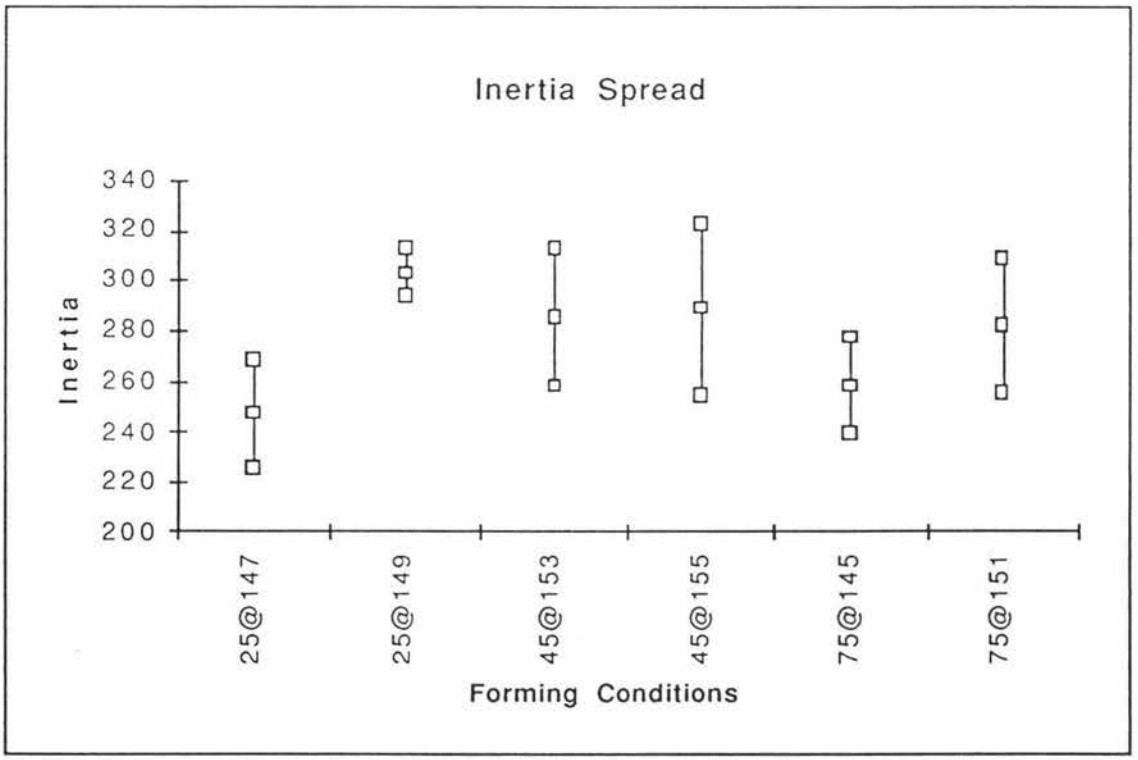


Figure 4.50 - Inertia Spread vs Forming Conditions

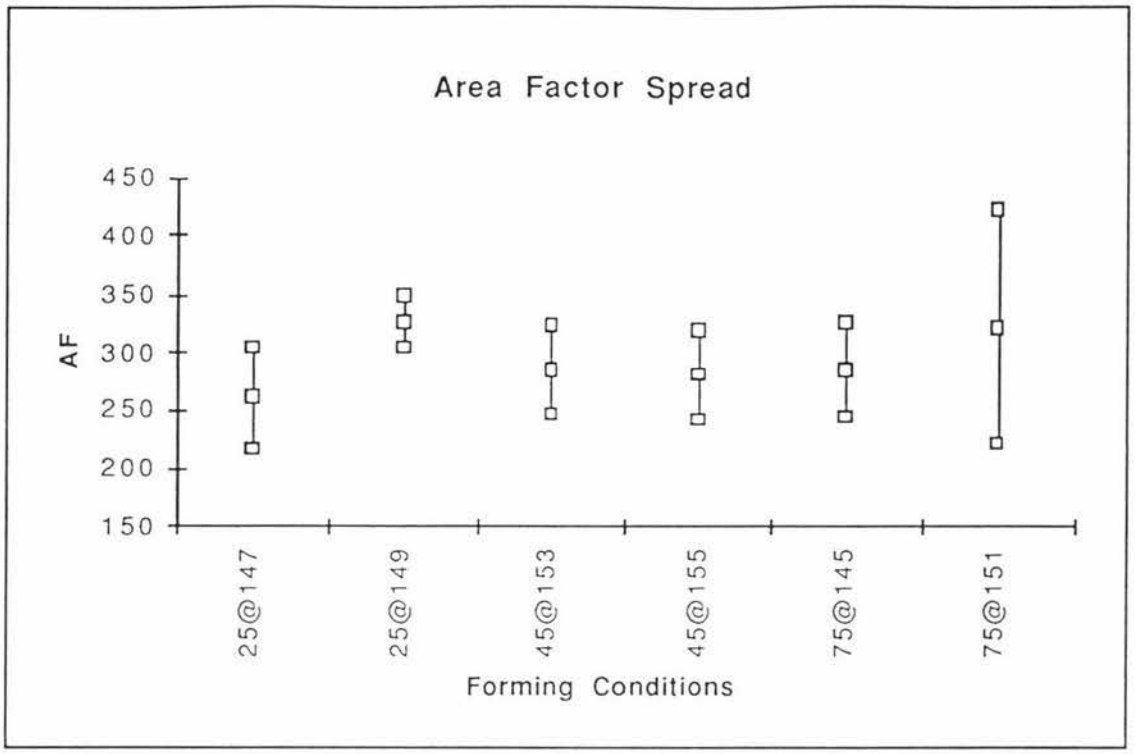


Figure 4.53 - Area Factor Spread vs Forming Conditions

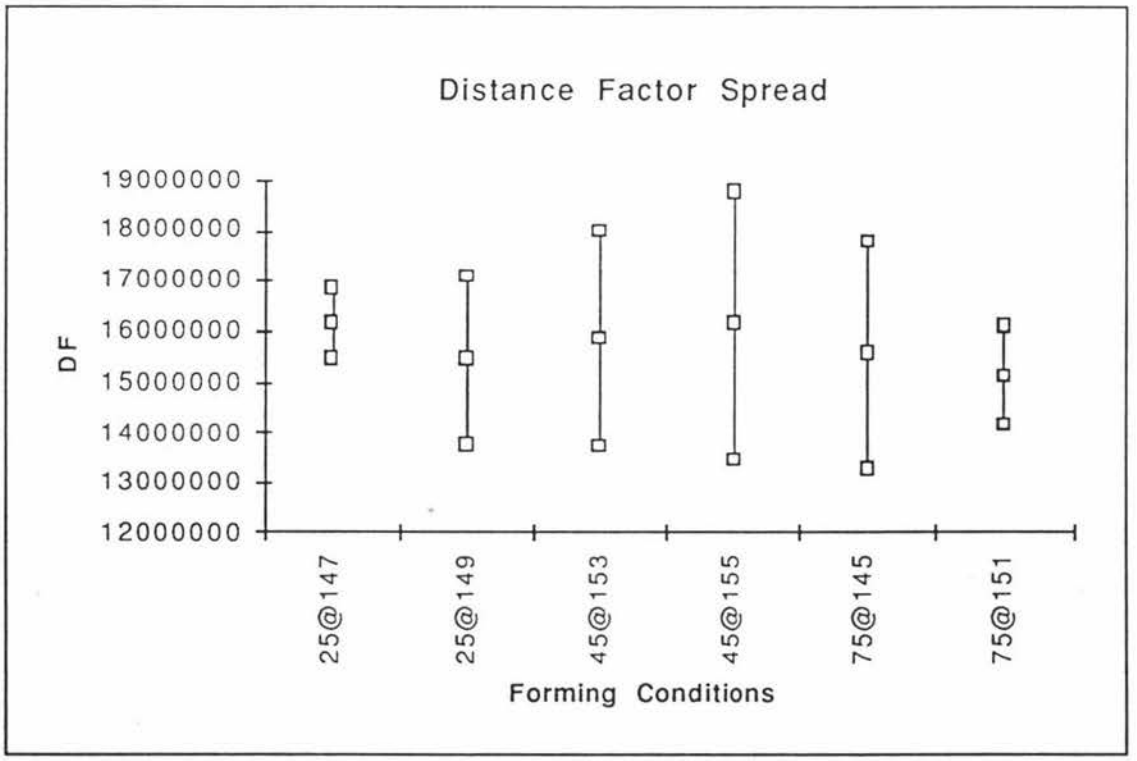


Figure 4.54 - Distance Factor Spread vs Forming Conditions

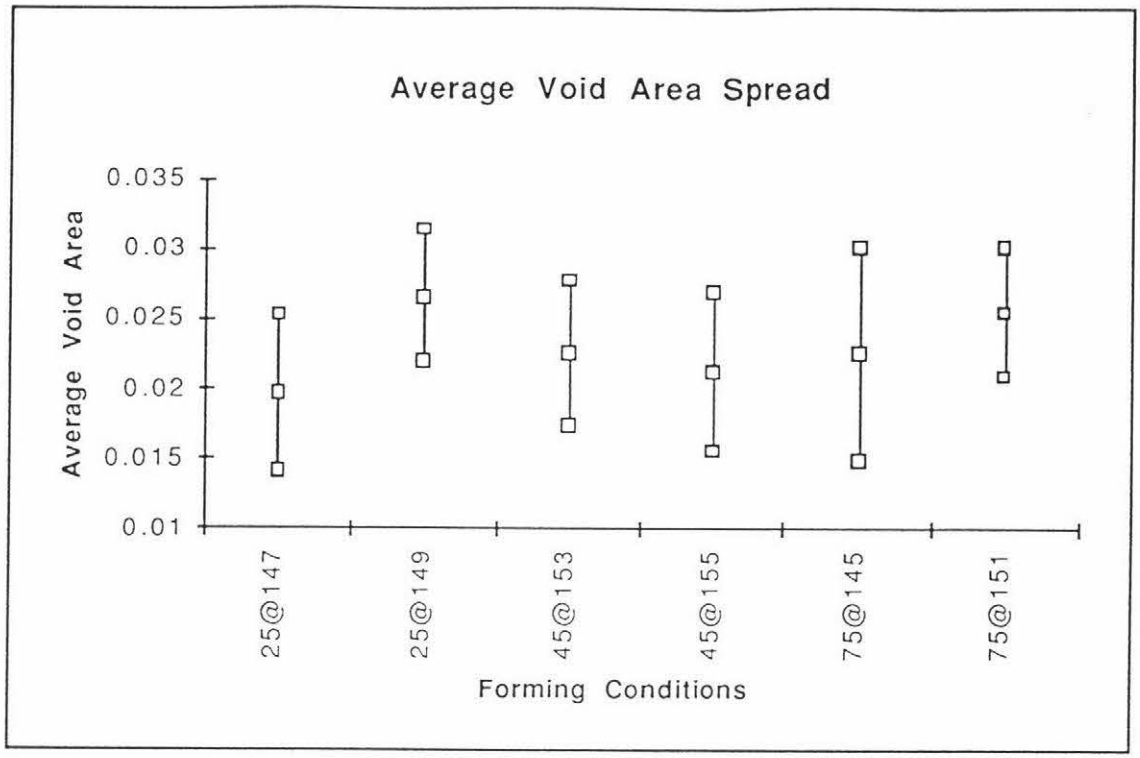


Figure 4.55 - Average Area Spread vs Forming Conditions

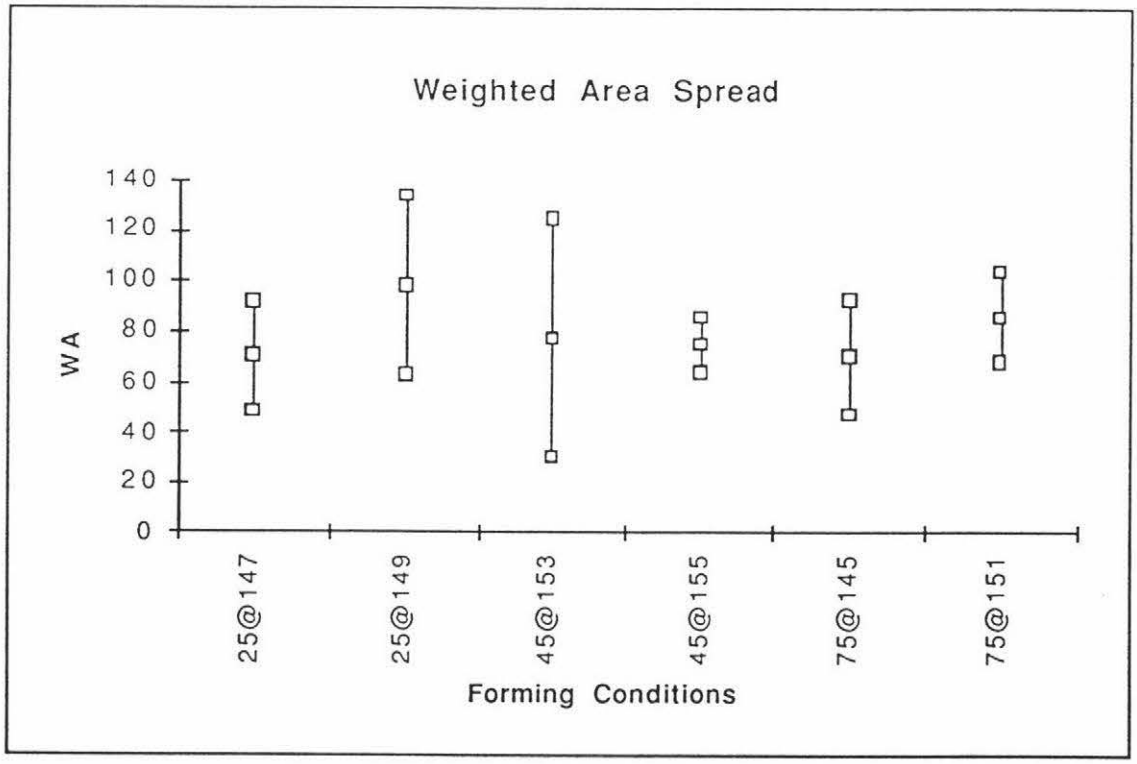


Figure 4.56 - Weighted Area Spread vs Forming Conditions

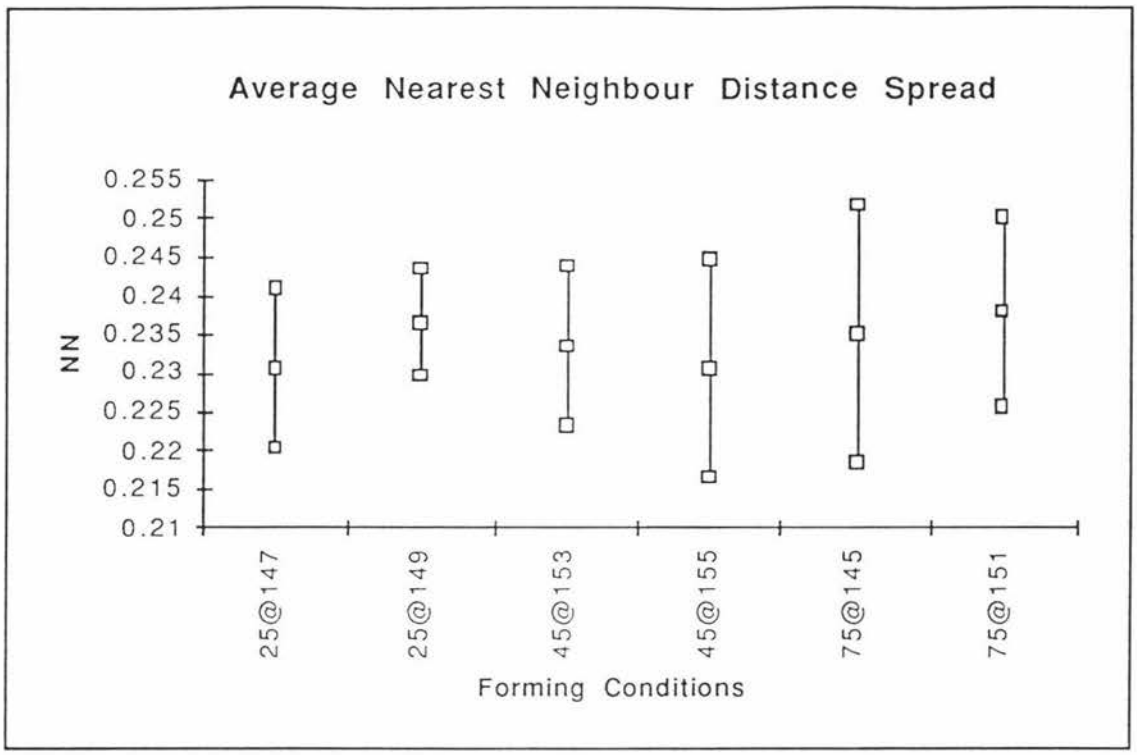


Figure 4.57 - Average Nearest Neighbour Distance Spread vs Forming Conditions

For the energy, and inertia plots the respective image feature mean marginally rises as the temperature rises for each of the three vibration times (25s, 45s, and 75s). This is a useful result as a rise in any of these features in a prepared sample could signify a temperature increase during the vibration pressing. Only the inertia plot shows any significant difference between forming conditions, in as much as the 25@149 value is significantly different from the 25@147 and 75@145 values.

None of the plots showed any trends between the image feature and the vibration time.

4.3.2.2 *Area and Spatial Distributions*

For each of the images the area and nearest neighbour distance distributions were noted. The Kolgomorov-Smirnov (KS) tests were again performed on the area and nearest neighbour distance (spatial) distributions to see if the distributions for differing forming conditions were different.

As ten images were captured for each forming condition, KS tests were performed to see if the area and spatial distributions were the same for images of the same forming condition as well as seeing if the distributions were different for images of differing forming conditions.

The KS results for the area distribution at a 99% level of confidence are summarised in Figure 4.58 where the number in each cell corresponds to the number of area distribution

differences (1's) that individual images had between the respective forming conditions divided by 100 (the total number of KS tests performed between each forming condition).

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	0.05	0.33	0.17	0.22	0.16	0.46
25@149	0.33	0.01	0.18	0.32	0.10	0.13
45@153	0.17	0.18	0.12	0.28	0.10	0.34
45@155	0.22	0.32	0.28	0.18	0.22	0.34
75@145	0.16	0.10	0.10	0.22	0.03	0.19
75@151	0.46	0.13	0.34	0.34	0.19	0.12

Figure 4.58 - Summarised KS Results for the Area Distribution at a 99% Level of Confidence

The KS results for the nearest neighbour distance (spatial) distribution at a 99% level of confidence are summarised in Figure 4.59.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	0.05	0.23	0.17	0.28	0.28	0.41
25@149	0.23	0.11	0.20	0.34	0.13	0.19
45@153	0.17	0.20	0.12	0.33	0.23	0.33
45@155	0.28	0.34	0.33	0.22	0.35	0.42
75@145	0.28	0.13	0.23	0.35	0.18	0.29
75@151	0.41	0.19	0.33	0.42	0.29	0.09

Figure 4.59 - Summarised KS Results for the Spatial Distribution at a 99% Level of Confidence

As seen earlier these results by themselves mean nothing as we do not know which values are significant or not. The large numbers on the leading diagonal (intersample) KS tests would indicate there is a large amount of variation between the samples. This indicates that there may need to be more samples taken to get less variation. The area and

nearest neighbour distance distributions of the ten images per forming condition were pooled together and this resulted in one area and one nearest neighbour distance distribution for each forming condition.

The results of the KS tests at a 99% level of confidence on these distributions is shown in Figures 4.60 and 4.61 below.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Different	Different	Same	Different	Different
25@149	Different	Same	Different	Different	Same	Same
45@153	Different	Different	Same	Same	Same	Different
45@155	Same	Different	Same	Same	Same	Different
75@145	Different	Same	Same	Same	Same	Different
75@151	Different	Same	Different	Different	Different	Same

Figure 4.60 - Pooled KS Results for the Area Distribution at a 99% Level of Confidence

	25-147	25-149	45-153	45-155	75-145	75-151
25@147	Same	Different	Same	Same	Different	Different
25@149	Different	Same	Same	Same	Same	Different
45@153	Same	Same	Same	Same	Same	Different
45@155	Same	Same	Same	Same	Different	Different
75@145	Different	Same	Same	Different	Same	Same
75@151	Different	Different	Different	Different	Same	Same

Figure 4.61 - Pooled KS Results for the Spatial Distribution at a 99% Level of Confidence

From the above data, it seems the KS test at a 99% level of confidence does not discriminate between the anodes in any useful manner. The temperature change associated with each of the vibration times does not result in significant differences in measured area

or spatial distributions. There are also no consistent differences between the area and spatial distributions of anodes with differing vibration times. It must be noted here that the leading diagonal for both tables is zero because there is only one set of pooled data for each forming condition. This means when a KS test is performed on two sets of the same data the result has to be zero. This accounts for the discrepancies in the leading diagonal from the summarised tables (figures 4.58 and 4.59).

The KS ratios were adjusted for a 95% level of confidence and the results are shown in figures 4.62 and 4.63.

	25@147	25@149	45@153	45@155	75@145	75@151
25@147	Same	Different	Different	Same	Different	Different
25@149	Different	Same	Different	Different	Different	Same
45@153	Different	Different	Same	Same	Same	Different
45@155	Same	Different	Same	Same	Same	Different
75@145	Different	Different	Same	Same	Same	Different
75@151	Different	Same	Different	Different	Different	Same

Figure 4.62 - Pooled KS Results for the Area Distribution at a 95% Level of Confidence

The lower confidence level does not lead to any new conclusions.

Overall it must be concluded that at a resolution of 40 pixels/mm, the measured area and spatial distribution of the voids does not change consistently with the forming conditions.

	25-147	25-149	45-153	45-155	75-145	75-151
25@147	Same	Different	Different	Same	Different	Different
25@149	Different	Same	Same	Same	Same	Different
45@153	Different	Same	Same	Same	Different	Different
45@155	Same	Same	Same	Same	Different	Different
75@145	Different	Same	Different	Different	Same	Different
75@151	Different	Different	Different	Different	Different	Same

Figure 4.63 - Pooled KS Results for the Spatial Distribution at a 95% Level of Confidence

4.4 Modelling

An attempt was made to model the forming conditions (temperature and time) from the ten image features. The image features of a series of images at 10 and 40 pixels/mm were noted and these used to create a model of the forming temperature and vibration time using the image features as predictors.

The models were created using "Minitab", a widely used statistical analysis package, but the results were limited. The best linear models had an R-squared (adj.) result of around 20%. This implies that a linear model was not appropriate. Lack of time prevented any attempts at extending the model to quadratic terms and allowing interaction of variables. Another potential problem was the lack of replicate data necessary to test the fit.

Any modelling of the forming conditions from the image features is future work.

5 CONCLUSIONS AND FUTURE WORK

For the pitching level changes at 10 pixels/mm the following may be concluded:

- The inertia, homogeneity, and distance factor image features vary systematically with the pitching level.
- The measured area histogram of the voids does not vary with changes in the pitching level at a 99% level of confidence. At the 95% level of confidence the measured area histogram of the underpitched anode differs significantly from that of the good anode and the slightly overpitched anode.
- The spatial distribution of the voids of the good anode was significantly different from the overpitched anode at a 99% level of confidence. At the 95% level of confidence the spatial distribution of the voids of the slightly overpitched anode was also significantly different from the overpitched anode.

For the pitching level changes at 40 pixels/mm the following may be concluded:

- None of the image features varied systematically with the pitching level.
- The measured area histogram of the underpitched anode varies significantly from the measured area histograms of the other anodes at a 99% level of confidence. This result is useful for discriminating the underpitched anode from the others. At the 95% level of confidence the overpitched anode also becomes significantly different from all the others. The KS test at this level of confidence now discriminates overpitched and underpitched anodes from better quality ones.
- The spatial distributions of the anodes are all significantly different from each other at a 99% level of confidence other except the slightly overpitched anode is the same as the overpitched anode with a KS ratio of 0.98. At the 95% level of confidence this KS ratio increased and hence all the anodes of any pitching level now have significantly different spatial distributions from any other pitching level.

For the forming condition changes at 10 pixels/mm the following may be concluded:

- The energy, homogeneity, and average void area image features rise when the forming temperature was raised and the vibration time kept constant.
- The correlation image feature varies linearly with the vibration time.

- The measured area histogram of the voids does not vary with changes in the forming conditions at a 99% level of confidence. At the 95% level of confidence the measured area histogram of the 25@147 forming condition differs from the 75@151 measured area histogram. The measured area histogram of the 45@155 forming condition also differs from the 75@145 measured area histogram.
- The spatial distribution of the voids changes significantly at a 99% level of confidence when the forming temperature rises at a constant vibration time. This result does not change for a 95% level of confidence.

For the forming condition changes at 40 pixels/mm the following may be concluded:

- The energy, and inertia image features rise slightly when the forming temperature was raised and the vibration time kept constant.
- The measured area histograms and spatial distributions of the voids vary inconsistently with the changes in the forming conditions at both 99% and 95% level of confidence.

Overall the results for the changing pitching levels for the 40 pixels/mm resolution are the most promising. It appears that the methods outlined in this work will successfully detect changes in pitching level but not changes in the forming conditions.

Better results may occur if more images' area and nearest neighbour distance distributions are pooled for the KS tests. This work used a maximum of ten images from a small individual sample. More images from more individual samples may give better discrimination results. It must be noted that this work has only shown that differences in the area and spatial distributions of the voids actually exist.

The following image processing routines provide a useful base for any future follow-on projects and related studies:

- Statistical Texture Analysis [3,4,5,6]
- Rank and Range Filters [14]
- Area/Distance Factors
- Weighted Area
- Nearest Neighbour Analysis (Spatial Distribution) [9]
- Circularity
- Connectivity

- Texture Primitives [15]

These routines have the potential to allow the necessary discrimination of anodes with different pitching levels or forming conditions to be made.

More work is necessary in the modelling of the forming conditions, along with physical measurements such as density, resistivity etc., from the image features. Experimental design using the forming conditions and measuring the images features could be useful in modelling the process.

REFERENCES

1. **B.A. Sadler.** The Optical Macroscopy Technique For Baked Anode Quality Assessment, *Technical Memorandum CRC/TM/86/89,* Comalco Research Centre, 1986.

2. **D.G. Bailey.** Report of Visit to NZAS. *Private Communication,* 1992.

3. **R.M. Haralick, K. Shanmugam, and I. Dinstein.** Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics,* 3: 610-621, 1973.

4. **J.S. Weszka, C.R. Dyer, and A. Rosenfeld.** A Comparative Study of Textural Methods for Terrain Classification, *IEEE Transactions on Systems, Man, and Cybernetics,* 6: 269-285, 1976.

5. **M.M Galloway.** Texture Analysis Using Gray Level Run Lengths, *Computer Vision, Graphics and Image Processing,* 4: 172-179, 1975.

6. **C. Sun, and W.G. Wee.** Neighbouring Gray Level Dependence Matrix for Texture Classification, *Computer Vision, Graphics and Image Processing,* 23: 341-352, 1982.

7. **L.S. Siew.** Texture Measures for Carpet Wear Assessment. Thesis, M.E., *University of Canterbury,* 1987.

8. **L.S. Siew, R.M. Hodgson, and E.J. Wood.** Texture Measurements for Carpet Wear Assessment. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 10: 92-105, 1988.

9. **A. Aplin.** *Order Neighbour Analysis.* Geo Books, Norwich, 1983.

10. **S. Siegel.** *Nonparametric Statistics for the Behavioural Sciences.* McGraw-Hill, New York, 1956.

11. **W. Rasband.** *NIH Image 1.44.* National Institute of Health, 1992.

12. **M. Vivino.** *Inside Image*. National Institute of Health, 1992.

13. **Image Analysis Unit, Massey University.** VIPS - Reference Manual and Users Guide - Version 4.1, 1991.

14. **R.M. Hodgson, D.G. Bailey, M.J. Naylor, A.L.M. Ng, and S.J. McNeill.** Properties, Implementations and Applications of Rank Filters. *Image and Vision Computing*, 3: 3-14, 1985.

15. **S. Wang, F.R.D. Velasco, A.Y. Wu, and A. Rosenfeld.** Relative Effectiveness of Selected Texture Primitive Statistics for Texture Discrimination. *IEEE Transactions on Systems, Man, and Cybernetics*, 11: 360-370, 1981.

APPENDIX A - User Manual

Introduction

Before one starts with this package one should be familiar with standard "Image1.44" [11,12]. The **User** menu, shown in figure 1, adds new analysis packages such as statistical texture analysis and neighbour analysis to "Image1.44". Other features such as a rank and range filter, and various measures such as circularity, area and distance factors, and weighted area have also been added. First of all the four statistical texture methods (SGLDM..., GLDM..., GLRLM..., and NGLDM...) [3,4,5,6] will be explained then the two sub-menus (**Filters** and **Measures**) will be described. Finally the macro commands for the **User** menu will be described.

N.B. It is imperative that a region of interest is chosen on any image before any of the statistical texture analysis commands are chosen. These statistical texture analysis will NOT work properly unless a region of interest is chosen first. If the user wants to analyse the whole image then the whole image must be selected using Select All in the Edit menu or another appropriate method.

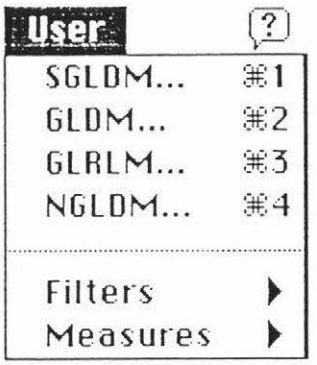
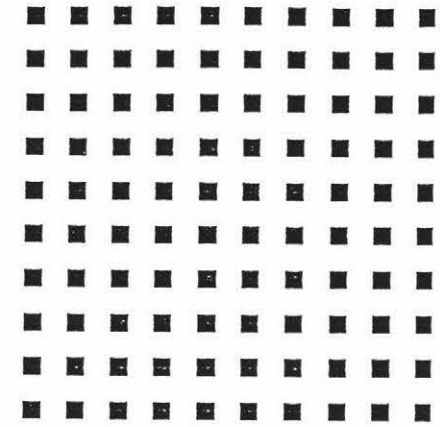


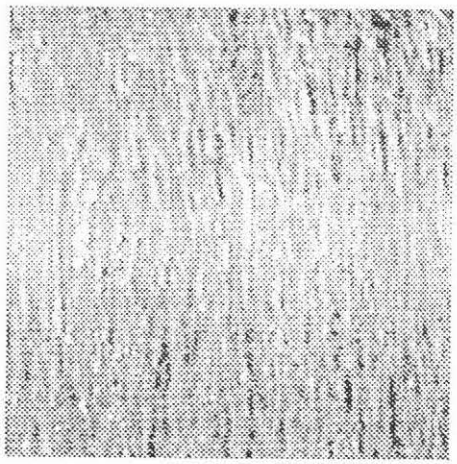
Figure 1 User Menu

Statistical Texture Analysis

Statistical texture analysis [3,4,5,6] is a powerful technique that is used to characterise texture features in an image in a quantitative, consistent and objective way. The texture of an image is concerned with the spatial distribution of the gray levels in the image. This distribution can be deterministic or stochastic in the extreme as shown in figure 2.



Deterministic



Stochastic

Figure 2 Texture Types

Deterministic textures are best analysed using structural methods such as placement rules and tree diagrams while stochastic textures such as carbon anodes are best analysed statistically. These stochastic textures can be analysed statistically using four different intermediate matrix methods:

1. Spatial gray level dependency method, SGLDM [3]
2. Gray level difference method, GLDM [4]
3. Gray level run length method, GLRLM [5]
4. Neighbouring gray level dependency method, NGLDM [6]

The intermediate matrices calculated from each of the above methods describe in coded form the spatial relationships between the gray levels in the image. These intermediate matrices allow the calculation of texture features to be made which in turn attempt to describe the texture in a meaningful way.

Spatial Gray Level Dependency Method

The spatial gray level dependency method [3] is the most widely used method and is based on the estimation of the second order joint conditional probability density functions *f(i,j,d,a)* where ***d*** is the intersample spacing and *a* is the direction *(ie 0°, 45°, 90°, 135°)*. This is illustrated in figure 3.

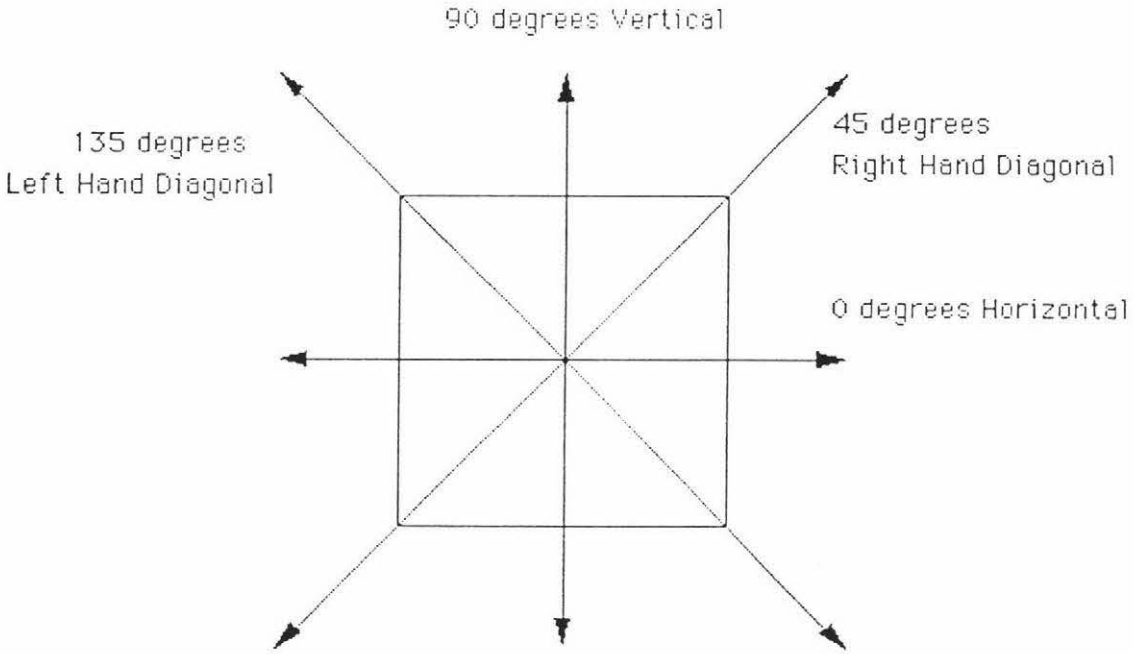


Figure 3 - Directions

f(i,j,d,a) is the probability of going from gray level *i* to gray level *j*, in distance ***d*** between the two, and in direction *a (0°, 45°, 90°, and 135°)*. If there are *Ng* gray levels in the image then the intermediate matrix *P(i,j,d,a)* is *Ng* × *Ng* in size.

The above method calculates four P(i,j,d,a) intermediate matrices, one for each direction **a**. For example, the simple image in figure 4 which has 5 gray levels (0-4), will result in four 5 × 5 matrices, one for each direction **a**. The resulting intermediate matrix for the horizontal direction is shown in figure 5.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	4

Figure 4 - Simple Image

4	2	1	0	0
2	4	0	0	0
1	0	6	1	0
0	0	1	0	1
0	0	0	1	0

Figure 5 - *P(i,j,1,0): 0°*

The underscored 4 in the *P(i,j,1,0°)* matrix above is the number of times a gray level 0 is next to the same level in the image in the horizontal direction.

The underscored 1 in the *P(i,j,1,0°)* matrix above is the number of times gray level 0 is next to gray level 2 in the image in the horizontal direction.

The four *P(i,j,d,a)* matrices are normalised by a normalising function dependant on the size of the region of interest (ROI) in the chosen image to give the four intermediate matrices: *f(i,j,d,0°)*, *f(i,j,d,45°)*, *f(i,j,d,90°)*, and *f(i,j,d,135°)*.

Haralick et al. [3] proposed various features that can be calculated from the these *f(i,j,d,a)* matrices. These features are energy, entropy, inertia, homogeneity and correlation.

- Energy is a measure of the homogeneity of the image. In a homogeneous image the energy will be high and for a nonhomogeneous image the energy will be low.
- Entropy is a measure of the complexity of an image. The more complex an image is the higher the entropy is.
- Inertia is a measure of the amount of local variations present in the image. The more local variations (contrast) present the higher the inertia.
- Homogeneity is a measure of the degree with which similar gray levels tend to be neighbours.
- Correlation is a measure of gray level linear dependencies.

When **SGLDM...** is chosen from the **User** menu, "Image" displays a dialogue box which asks for an intersample distance *d*. "Image" uses this distance to calculate the above features from the chosen region of interest in the image. This dialogue box is shown in figure 6. In order to put the SGLDM results into the results window, one must choose the **Texture** option in the options dialogue box first as shown in figure 7. This dialogue box appears when **Options...** is chosen from the **Analyze** Menu as shown in figure 8. This will display the results for each direction in tabular form in the results

window when the **Show Results** command is chosen from the **Analyze** menu. This results window is shown in figure 9.

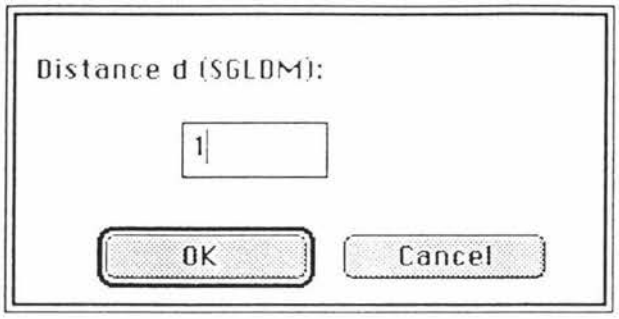


Figure 6 - SGLDM Distance

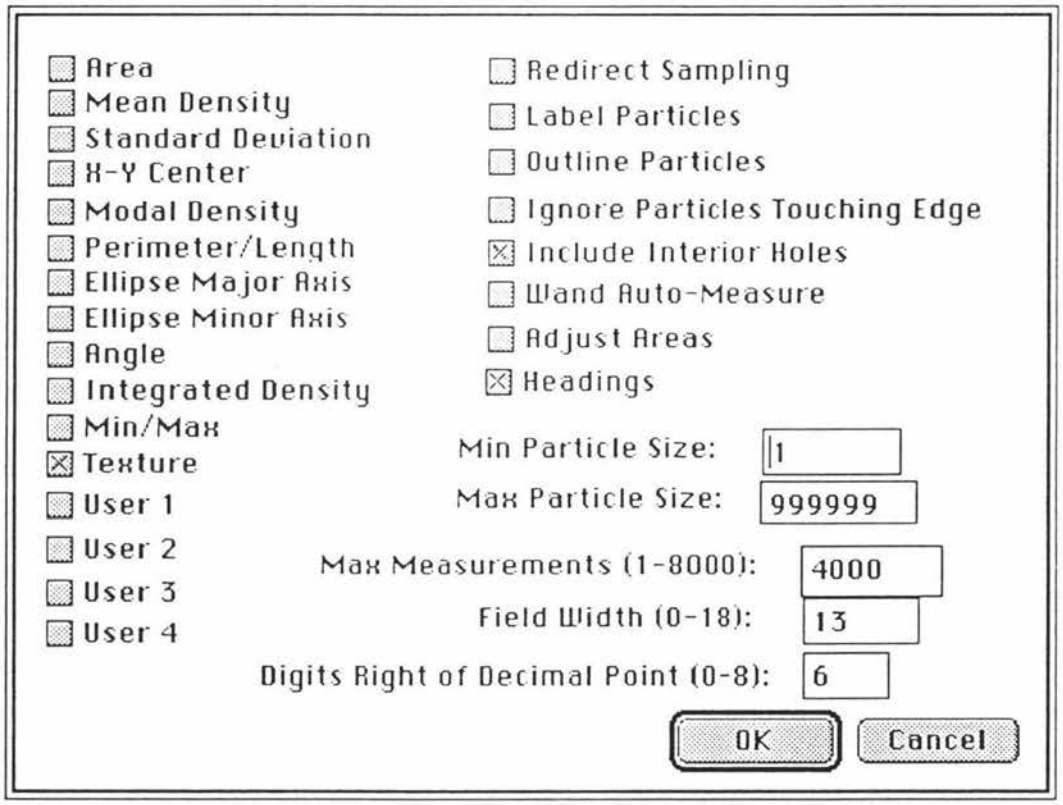


Figure 7 - Options Dialogue Box

Figure 8 - Options...

Results

1. Texture Features (SGLDM)

Direction	Energy	Entropy	Inertia	Homog.	Corr.
0	0.013661	1.993221	6.137362	0.388163	0.920243
45	0.009779	2.127071	18.875000	0.250940	0.771299
90	0.009707	2.155408	12.527778	0.309827	0.856494
135	0.010558	2.104840	13.214286	0.312718	0.839595
Avg	0.010926	2.095135	12.688607	0.315412	0.846908
Max	0.013661	2.155408	18.875000	0.388163	0.920243
Min	0.009707	1.993221	6.137362	0.250940	0.771299
Ran	0.003954	0.162187	12.737638	0.137223	0.148944
SD	0.001614	0.061509	4.513658	0.048703	0.053009

Spacing = 1

Figure 9 - SGLDM Results

This table has been put into the results window which can then be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

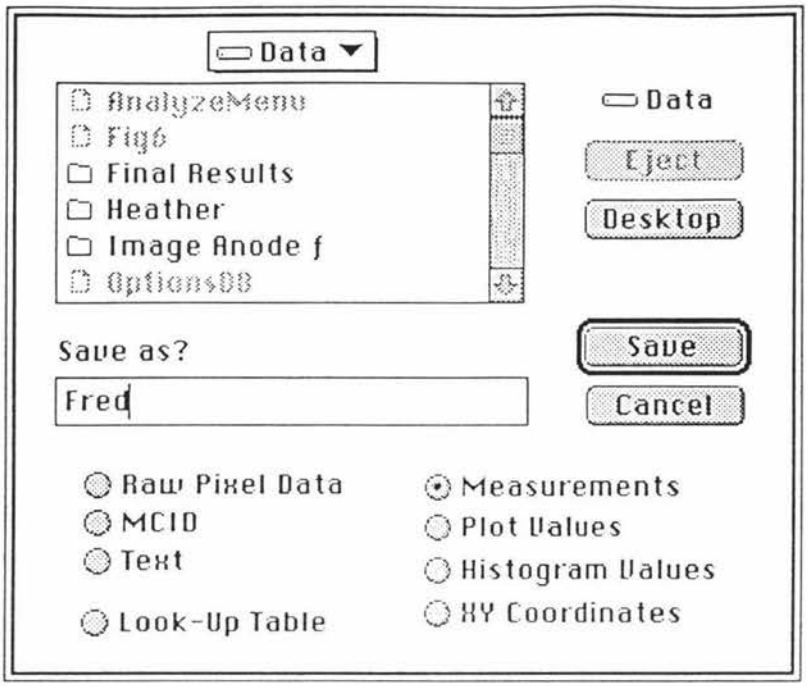


Figure 10 Save As... Dialogue Box

Doing more texture analysis will add tables to the results window so when the results window is next saved it will include all work done up to then. The window and hence the results to date can be cleared at will by choosing Reset from the **Analyze** menu, as shown in figure 11.

Figure 11 - Reset

Gray Level Difference Method

The gray level difference method [4] works on the gray level differences between two adjacent pixels separated by a distance *d*. Let $f(x,y)$ be the digital image and $f'(x,y) = |f(x,y)-f(x+\Delta x, y+\Delta y)|$ where Δx and Δy are integers giving the displacement, *d*.

Let P' be the probability density function of f'. If there are Ng gray levels in the image then P' has the form of a Ng dimensional vector whose ith component is the probability that $f'(x,y)$ will have value i. It is simple to compute $P'(i)$ from f by counting the number of times each value of $f'(x,y)$ occurs.

The above method is calculated for each of the four basic directions *a* as shown in figure 3.

Weszka et al. [4] proposed various features that can be calculated from the four $P'(i)$ matrices. These features are contrast (CON), angular second moment (ASM), entropy (ENT), mean (MEAN) and inverse different moment (IDM). These features attempt to describe the texture numerically.

- The contrast is the second moment about $P'(i)$. This is greatest when the visual contrast in the image is large.
- The angular second moment is smallest when $P'(i)$ are all as equal as possible and large when some values are high and some low.
- The entropy is largest for equal $P'(i)$ and smallest when they are very unequal.
- The mean is smallest when $P'(i)$ are concentrated near the origin and largest when they are far from the origin.

When **GLDM...** is chosen from the **User** menu, "Image" displays a dialogue box which asks for an intersample distance *d*. "Image" uses this distance to calculate the above features from the chosen region of interest in the image. This dialogue box is shown in figure 12. In order to put the GLDM results into the results window, one must choose the **Texture** option in the options dialogue box first as shown in figure 7. This will display the results for each direction in tabular form in the results window when the **Show Results** command is chosen from the **Analyze** menu. This results window is shown in figure 13.

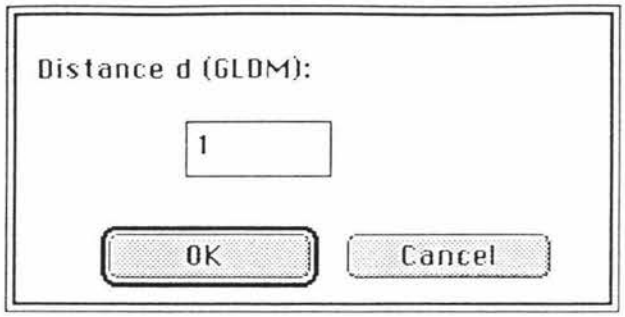


Figure 12 - GLDM Distance

Results

1. Texture Features (GLDM)

Direction	CON	ASM	ENT	MEAN	IDM
0	6.20556	0.20512	0.75928	1.91667	0.39248
45	18.87500	0.12238	0.97303	3.44643	0.25094
90	12.39011	0.15016	0.88688	2.71978	0.30642
135	13.21429	0.15710	0.89211	2.72619	0.31272
Avg	12.67124	0.15869	0.87782	2.70227	0.31564

Spacing = 1

Figure 13 - GLDM Results Window

This table has been put into the results window which can then be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

Doing more texture analysis will add tables to the results window so when the results window is next saved it will include all work done up to then. The window and hence the results to date can be cleared at will by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Gray Level Run Length Method

The gray level run length method [5] is based on calculating the number of gray level runs of various lengths in the four basic directions a as shown in figure 3. A gray level run is a set of consecutive co-linear pixels of the same gray level. The length of the run is the number of pixels in the run.

The intermediate matrices $P(i,j)$ specify the number of times the image contains a run of length j, in the given direction, consisting of pixels of gray level i. Let Ng be the number of gray levels and Nr be the number of different possible run lengths.

Using the image in figure 14, the $P(i,j)$ matrix for $0°$ is shown in figure 15.

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

Figure 14 - The Image

	Run Length N			
	1	2	3	4
Gray 0	4	0	0	0
Level 1	1	0	1	0
2	3	0	0	0
3	3	1	0	0

Figure 15 - $P(i,j)$ for $0°$

The underscored 1 means the gray level 1 has one run of length 3 in the image.

Galloway [5] proposed various features that can be calculated from the four $P(i,j)$ matrices. These features are long run emphasis (LRE), short run emphasis (SRE), gray level nonuniformity (GLNU), run length nonuniformity (RLNU) and run percentage (RPC). These features attempt to describe the texture numerically.

- The long run emphasis gives greater weight to long runs of any gray level.
- The short run emphasis gives greater weight to short runs of any gray level.
- When runs are equally distributed throughout the gray levels the gray level nonuniformity is smallest.
- When runs are equally distributed throughout the run lengths the run length nonuniformity is smallest.
- The run percentage is the lower for images with the greatest linear structure.

When **GLRLM...** is chosen from the **User** menu, "Image" calculates the above features from the chosen region of interest in the image. Provided the **Texture** option has been chosen from the options dialogue box, a table similar to that shown in figure 16 will be displayed when the **Show Results** command is chosen from the **Analyze** menu.

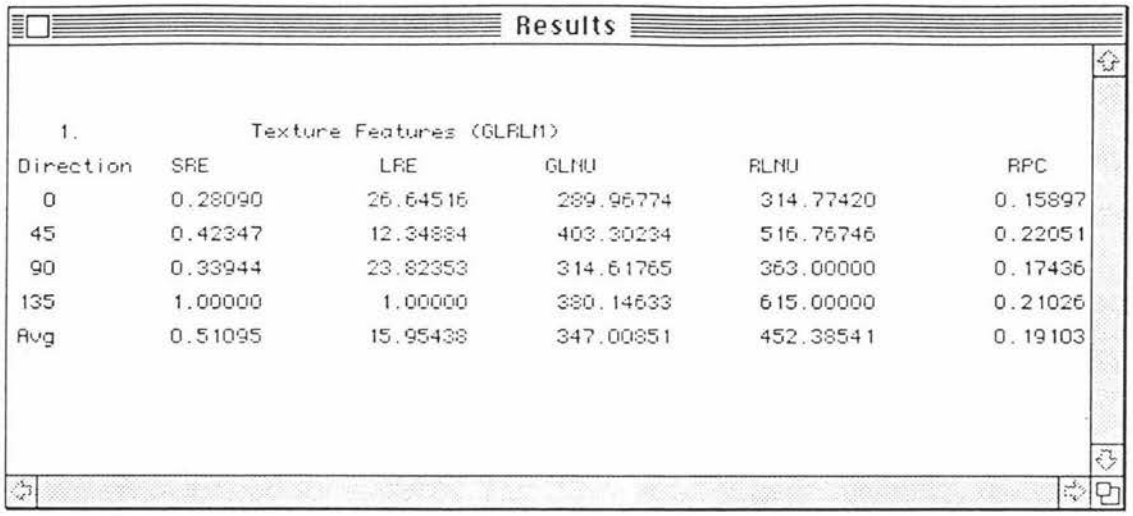


Figure 16 - GLRLM Results Window

This table has been put into the results window which can then be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

Doing more texture analysis will add tables to the results window so when the results window is next saved it will include all work done up to then. The window and hence the results to date can be cleared at will by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Neighbouring Gray Level Dependency Method

The neighbouring gray level dependency method [6] is directionally independent. The intermediate matrix $Q(k,s)$ is calculated by considering the relationship between a pixel and all its neighbouring pixels, at a distance less than or equal to d, at one time instead of in one direction at a time.

For example consider the image in figure 17 which has 8 gray levels 0-7.

4	4	6	5	4	3
4	4	5	3	0	1
3	3	5	0	0	1
2	0	7	3	3	2
0	0	7	7	3	3
0	1	6	6	2	2

Figure 17 - The Image

An intermediate intermediate matrix *P(i,j,d,a)* is calculated. For the above example this is calculated on pixel *3,3*, which has a gray level of 5, in a neighbourhood of ***d=1*** around it, with the difference factor, ***a=zero***. There is only one pixel in the neighbourhood of distance 1 ***(d=1)*** with a gray level of 5 equal ***(a=0)*** to that of pixel *3,3*. Therefore *P(3,3,1,0)=(5,1)* where 5 is the gray level and 1 is the NGLDM number for the pixel *3,3*.

The complete *P(i,j,1,0)* for the image is shown in figure 18.

(4,3)	(5,2)	(3,0)	(0,2)
(3,1)	(5,1)	(0,2)	(0,2)
(0,2)	(7,2)	(3,2)	(3,3)
(0,3)	(7,2)	(7,2)	(3,3)

Figure 18 - *P(i,j,1,0)*

Q(k,s) is the intermediate matrix for NGLDM and is the total number of entries in *P* that have gray level ***k*** and NGLDM number ***s***.

eg. *Q(7,2)=3* because there are 3 entries of *(7,2)* in the *P* matrix in figure 18. Therefore the *Q* matrix in figure 19 can be considered as frequency counts of the greyness variation of an image. It is similar to the histogram of the image.

		NGLDM Numbers s							
		0	1	2	3	4	5	6	7
	0	0	0	4	1	0	0	0	0
	1	0	0	0	0	0	0	0	0
Gray	2	0	0	0	0	0	0	0	0
Level	3	1	1	1	2	0	0	0	0
k	4	0	0	0	1	0	0	0	0
	5	0	1	1	0	0	0	0	0
	6	0	0	0	0	0	0	0	0
	7	0	0	3	0	0	0	0	0

Figure 19 - $Q(k,s)$

As shown above $Q(7,2) = 3$.

Sun and Wee [6] proposed various features that can be calculated from the $Q(k,s)$ matrix. These features are small number emphasis (SNE), large number emphasis (LNE), number nonuniformity (NNU), second moment (SM) and entropy (ENT). These features attempt to describe the texture numerically.

- The small number emphasis is a measure of the fineness of the image. The finer the image is the larger the small number emphasis is.
- The large number emphasis is a measure of the coarseness of the image. The coarser the image is the larger the large number emphasis is.
- The number nonuniformity is related to the coarseness of the image.
- The second moment is a measure of the homogeneity of the image. The larger the second moment is the more homogeneous the image is.
- The entropy is related to the coarseness of the image.

When **NGLDM...** is chosen from the **User** menu, "Image" displays two dialogue boxes which asks for an intersample distance d, and a difference factor a. "Image" uses these numbers to calculate the above features from the chosen region of interest in the image. These dialogue boxes are shown in figures 20 and 21. In order to put the NGLDM results into the results window, one must choose the **Texture** option in the options dialogue box first as shown in figure 7. This will display the results in tabular form in the results window when the **Show Results** command is chosen from the **Analyze** menu. This results window is shown in figure 22.

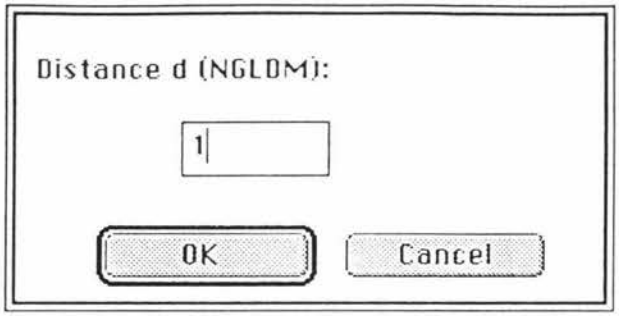


Figure 20 - NGLDM Distance

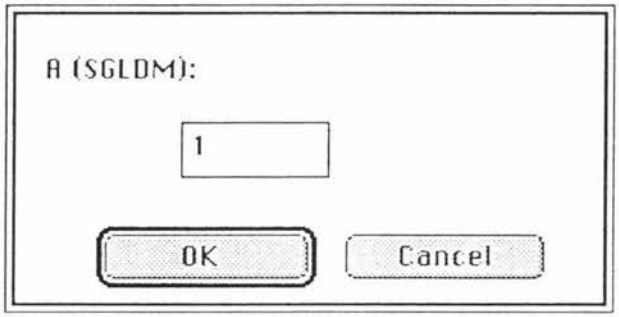


Figure 21 NGLDM Difference

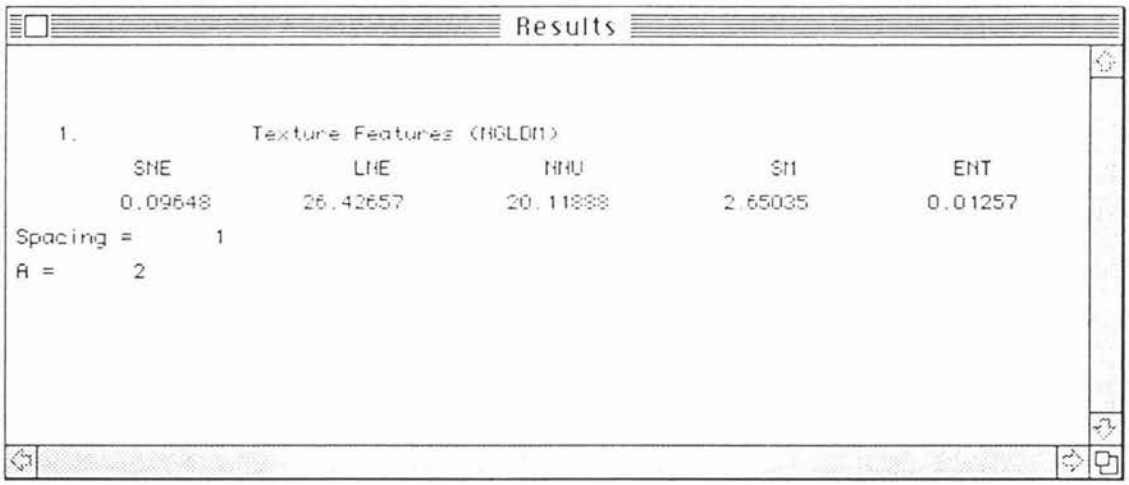


Figure 22 NGLDM Results Window

This table has been put into the results window which can then be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

Doing more texture analysis will add tables to the results window so when the results window is next saved it will include all work done up to then. The window and hence the results to date can be cleared at will by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

The Filters Submenu

The **Filters** Sub-menu contains the following items as shown in figure 23:

1. Rank Filter...
2. Range Filter...

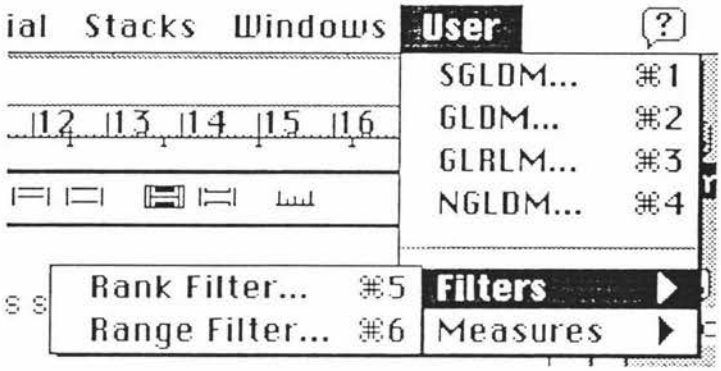


Figure 23 - Filters Sub-Menu

Rank Filter [14]

This menu item performs a rank filter operation on an image. The user is first asked what rank they want done. A rank of 1 is a Minimum filter, a rank of 9 is a Maximum filter, and a rank of 5 is a median filter.

Range Filter

This menu item performs a range filter (edge detection) operation on an image. The user is first asked what ranks they want performed.

The Measures Submenu

The **Measures** Sub-menu contains the following items as shown in figure 24:

- Area Factor
- Distance Factor
- Weighted Area
- Neighbour Analysis [9]
- Circularity
- Extend Image [13]
- Blobs [13]
- Connectivity

- Texture Primitives [15]

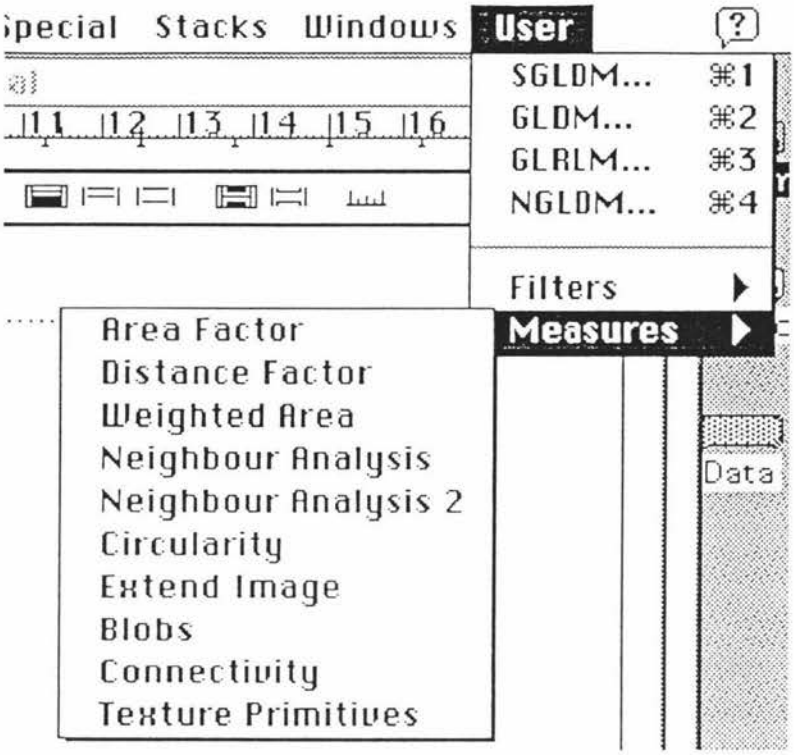


Figure 25 - Measures Sub-Menu

Area Factor (NB - Intended for binary images only)

The area factor (AF) measures the spatial and size distribution from the areas and distances between all the voids in an image. For a particular image, the AF is calculated as follows:

$$AF = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{N}\frac{A_j}{\left(d_{ij}\right)^2}\right] \qquad i \neq j \tag{1}$$

where N is the total number of voids in the image

A_j is the area of the *jth* void

d_{ij} is the distance between the *ith* and *jth* void

When **Area Factor** is chosen from the **Measures** submenu, "Image" calculates and displays the area factor for each and every void in the image in the results window, as shown in figure 26. In order to get the area factor results into the correct place, the **Area** and **User1** options must have been chosen first from the options dialogue box, as shown in figure 27.

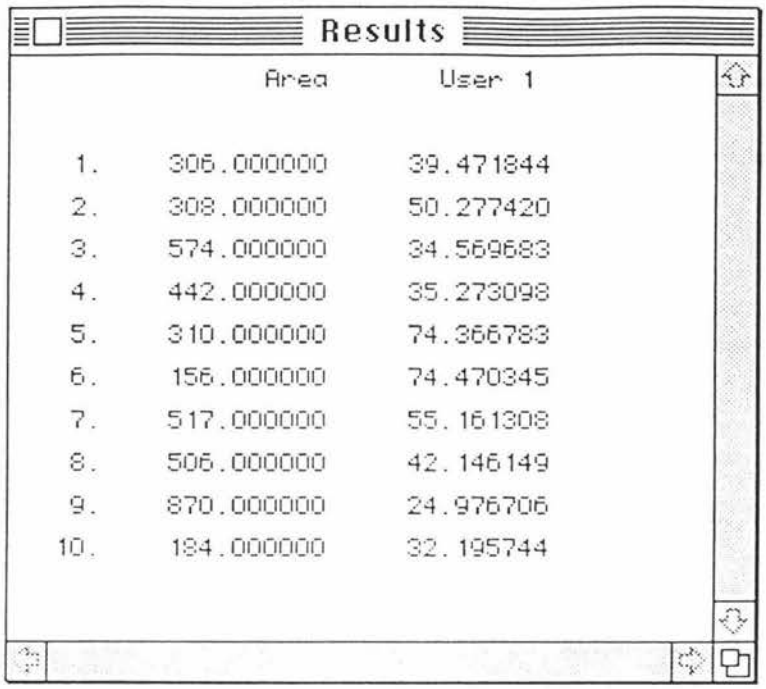


Figure 26 - Area Factor Results

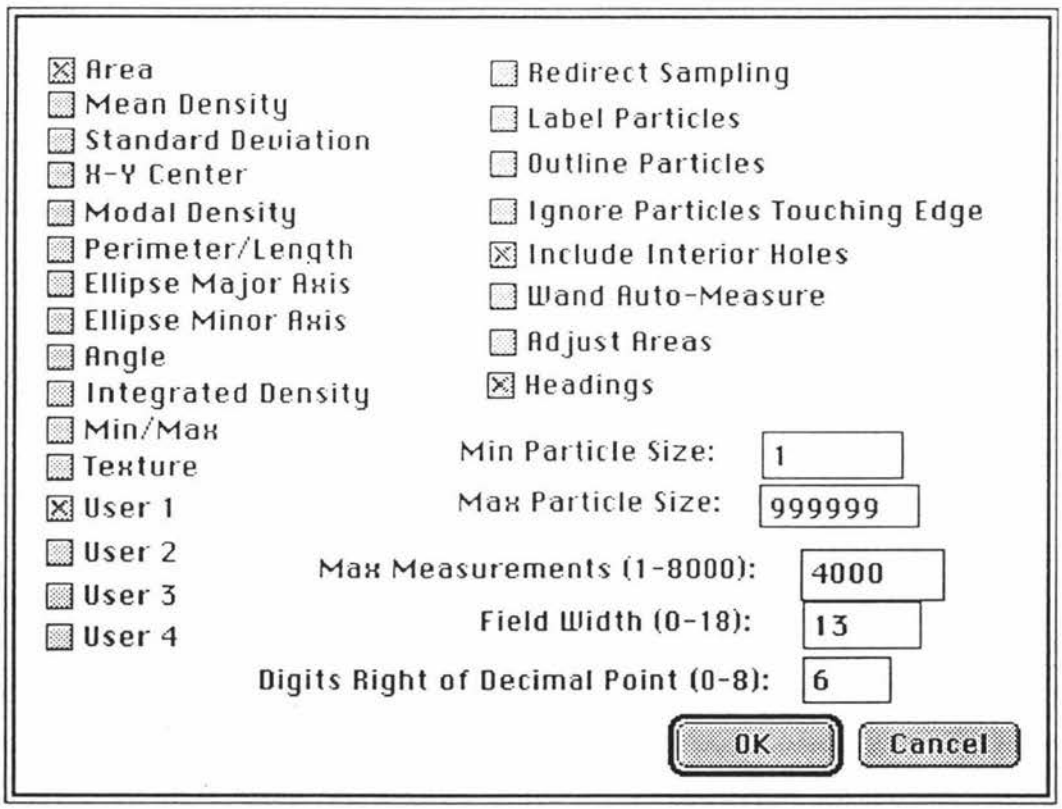


Figure 27 - Options Dialogue Box

The results in figure 26 show that there were 10 voids in the image with the individual area factors in the User1 column. To get the area factor for the image as a whole, the User1 column in the results window is averaged. To do this, the results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as

measurements. This is shown in figure 10. The average of the User1 column can now be found using "Excel".

To do more area factor analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Distance Factor (NB - Intended for binary images only)

The distance factor (DF) quantifies only the spatial distribution of the voids. This measure is the same as the area factor except that it is not weighted by the areas of the voids.

$$DF = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{N}\frac{1}{\left(d_{ij}\right)^2}\right] \qquad i \neq j \tag{2}$$

where N is the total number of voids in the image

d_{ij} is the distance between the *ith* and *jth* void

When **Distance Factor** is chosen from the **Measures** submenu, "Image" calculates and displays the distance factor for each and every void in the image in the results window, as shown in figure 28. In order to get the distance factor results into the correct place, the **Area** and **User1** options must have been chosen first from the options dialogue box, as shown in figure 27.

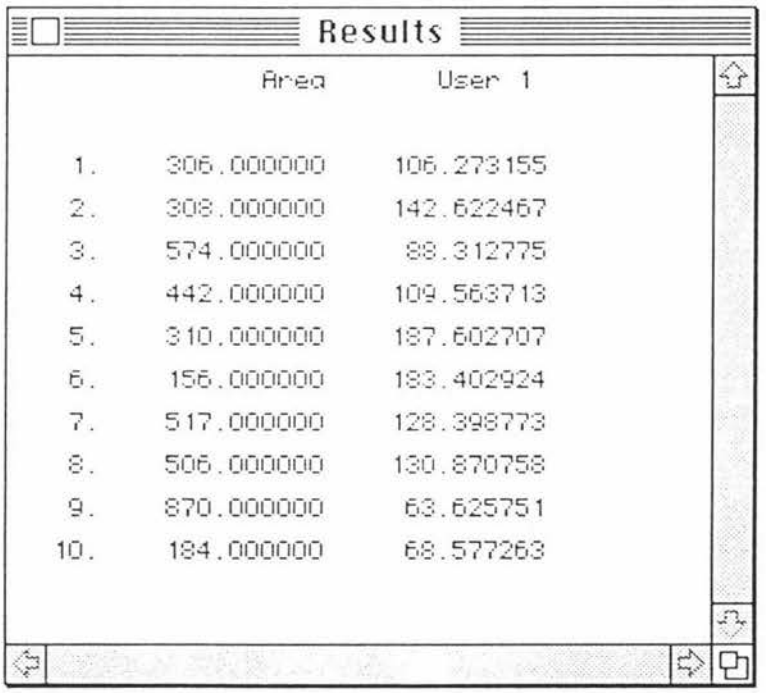


Figure 28 - Distance Factor Results

The results in figure 28 show that there were 10 voids in the image with the individual distance factors in the User1 column. To get the distance factor for the image as a whole, the User1 column in the results window is averaged. To do this, the results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10. The average of the User1 column can now be found using "Excel".

To do more distance factor analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Weighted Area (NB - Intended for binary images only)

This is a measure of the volume of the voids in a sample. It is assumed that the larger the cross-sectional area of a void in our image of a sample, the larger the volume of the void in the original material will be. Each pixel within a void is weighted in proportion to its distance from the edge. The weighted area (WA) is calculated by using a MIN filter to remove the outermost layer of pixels from all of the voids in the image. The change in total area is given a weighting of 1 for the first layer removed, a weighting of 2 for the second layer etc, as shown in figure 29. The process is repeated until all the voids in the image have disappeared, that is eroded to nothing. The area changes for each layer removed, along with the respective weightings, are summed to give a weighted area for the image as a whole.

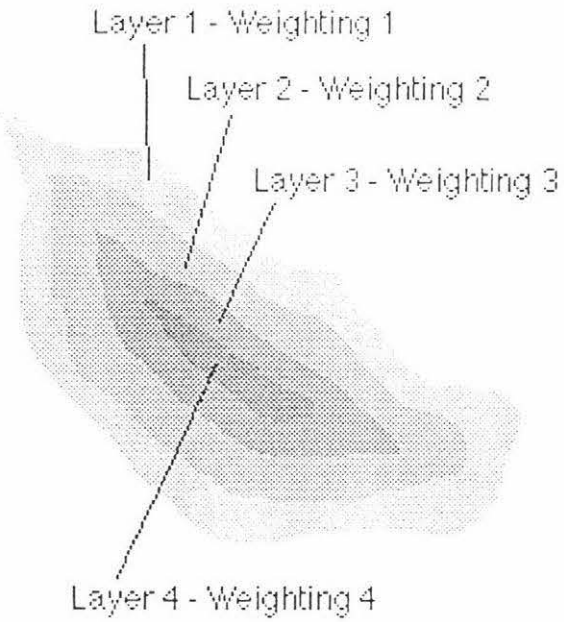


Figure 29 - WA Four Layers

The weighted area can be summarised in the following equation:

$$WA = \sum_{i=1}^{N} iA_i \qquad (3)$$

Where N is the number of layers in the void

A_i is the area of the layer

When **Weighted Area** is chosen from the **Measures** submenu, "Image" calculates and displays the weighted area for each iteration of the MIN filter in the results window, as shown in figure 30. In order to get the weighted area results into the correct place, the

Area and **User1** options must have been chosen first from the options dialogue box, as shown in figure 27.

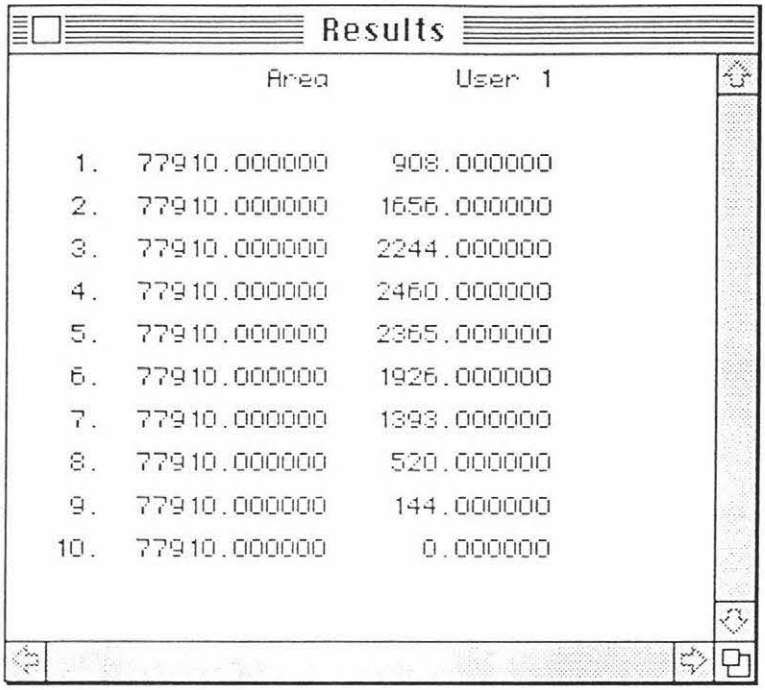


Figure 30 - Weighted Area Results

The results in figure 30 show that 10 iterations of the MIN filter were necessary to find the weighted areas in the User1 column. To get the weighted area for the image as a whole, the User1 column in the results window is summed. To do this, the results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10. The average of the User1 column can now be found using "Excel".

To do more weighted area analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Neighbour Analysis (NB - Intended for binary images only)

This routine is where the nearest neighbour distances are calculated for characterising the spatial distribution. The data from this routine is used in "Excel" for the KS tests and to find the average nearest neighbour distance.

"Image" calculates and displays the distances to the nearest four neighbours for each void in the image if **Neighbour Analysis** is chosen from the **Measures** submenu. If **Neighbour Analysis 2** is chosen, "Image" calculates and displays the 5th, 6th, 7th and 8th nearest neighbour distances for each void in the image.

If the distance to the nearest boundary is closer to the void in question than the distance to its first nearest visible neighbour, then the first nearest neighbour distance for that void is

zero (ie it doesn't exist). This is because what lies outside the boundary is unknown. If a void has no first nearest neighbour then it cannot possibly have a second or third and so on. This process can be repeated for higher neighbouring levels. Therefore if the distance to the nearest boundary is closer to the void in question than the distance to its second nearest visible neighbour, then the second nearest neighbour distance for that void is zero (ie it doesn't exist). This again is because what lies outside the boundary is unknown. If a void has no second nearest neighbour then it cannot possibly have a third or fourth and so on.

Each time the neighbour level increased, the effective area over which the neighbour analysis was taking place, grew smaller. This is because the outer voids were being ignored as they were closer to the boundaries of the image than their nearest visible neighbour. These area changes are noted by "Image" and displayed in the results window in the area column. These are used in calculating the R statistics. These R statistics can be assumed to have no boundary as only voids which have visible nearest neighbours closer than the nearest boundary are accounted for.

When **Neighbour Analysis** is chosen from the **Measures** submenu, "Image" calculates and displays the distances to the nearest four neighbours for each void in the image in the results window, as shown in figure 31. In order to get the nearest neighbour distances into the correct place, the **Area**, **X-Y Centre**, **User1**, **User2**, **User3**, and **User4** options must have been chosen first from the options dialogue box, as shown in figure 32.

Results

	Area	X	Y	User 1	User 2	User 3	User 4
1.	26058.000000	47.000000	43.000000	0.000000	0.000000	0.000000	0.000000
2.	11550.000000	100.000000	65.000000	57.140179	0.000000	0.000000	0.000000
3.	11550.000000	187.000000	76.000000	0.000000	0.000000	0.000000	0.000000
4.	11550.000000	26.000000	93.000000	0.000000	0.000000	0.000000	0.000000
5.	310.000000	131.000000	113.000000	46.690472	48.414875	57.140179	67.119293
6.	156.000000	85.000000	121.000000	46.690472	52.172791	57.974133	65.306969
7.	517.000000	169.000000	143.000000	48.414875	0.000000	0.000000	0.000000
8.	506.000000	74.000000	172.000000	52.172791	0.000000	0.000000	0.000000
9.	870.000000	161.000000	220.000000	77.414467	0.000000	0.000000	0.000000
10.	184.000000	48.000000	225.000000	0.000000	0.000000	0.000000	0.000000

Figure 31 - Neighbour Analysis Results

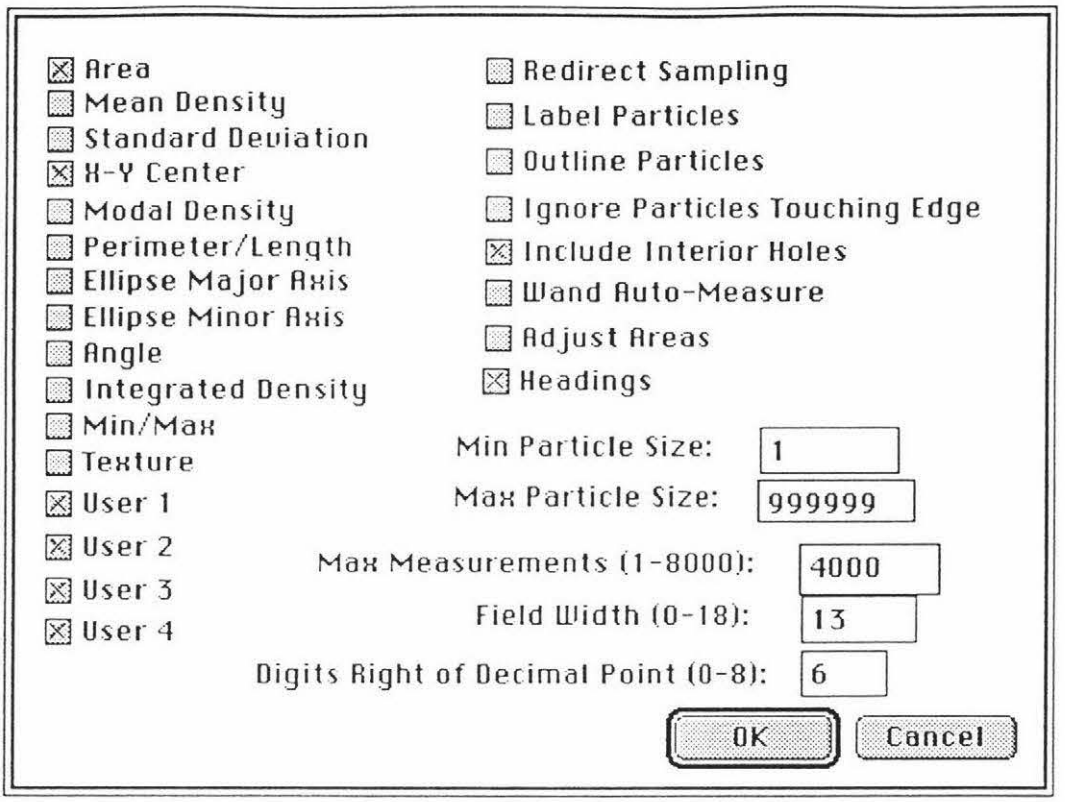


Figure 32 - Options Dialogue Box

The results in figure 31 show that there were 10 voids in the image and the four nearest neighbour distances are shown in the User1, User2, User3, and User4 columns. To get the average nearest neighbour distance, the User1 column in the results window is averaged. To do this, the results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10. The average of the User1 column can now be found using "Excel". Note the large number of zeros present in the results. These zeros show many of the voids were closer to the edges of the image than the other voids.

To do more neighbour analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Circularity (NB - Intended for binary images only)

As "Image" measures lengths of the major and minor axes of the best fitting ellipse of a void in an image then a measure of the circularity of the voids could be created. In this case the circularity is the minor axis divided by the major axis. Therefore a circularity of 1.0 would indicate a perfectly circular void. The lower the circularity, the longer and thinner the void in question. It must be noted that the circularity measure would work best on voids with more pixels as the circularity measure could be distorted by extremely small voids (<20 pixels).

When **Circularity** is chosen from the **Measures** submenu, "Image" calculates and displays the circularity for each and every void in the image in the results window, as shown in figure 33. In order to get the circularity results into the correct place, the **Area**, **Ellipse Major Axis**, **Ellipse Minor Axis**, and **User1** options must have been chosen first from the options dialogue box, as shown in figure 34.

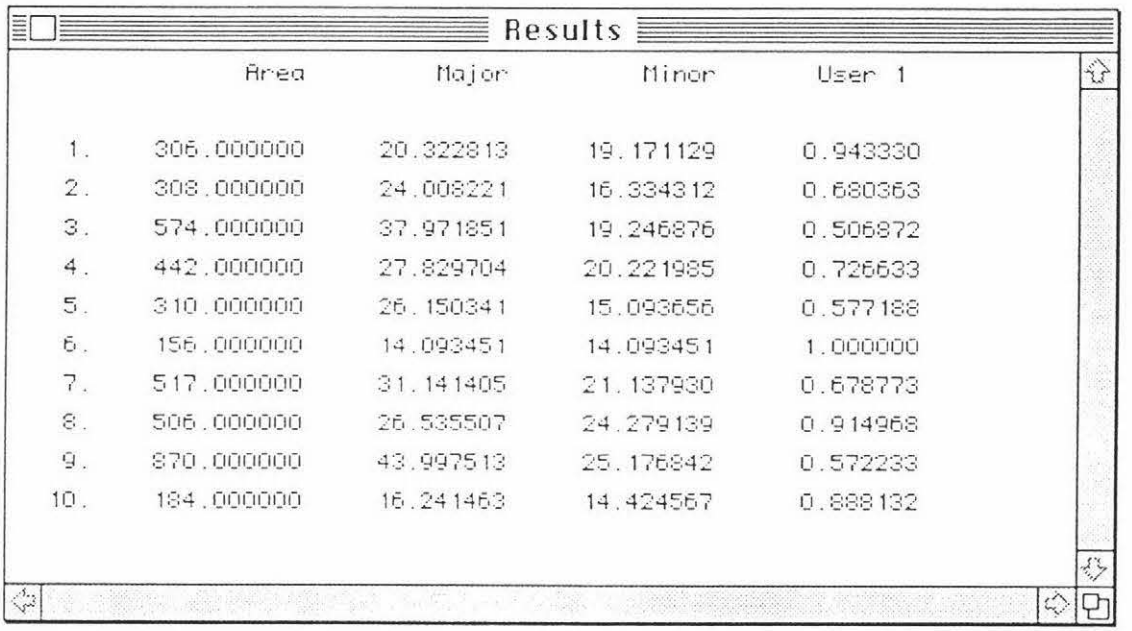


Figure 33 - Circularity Results

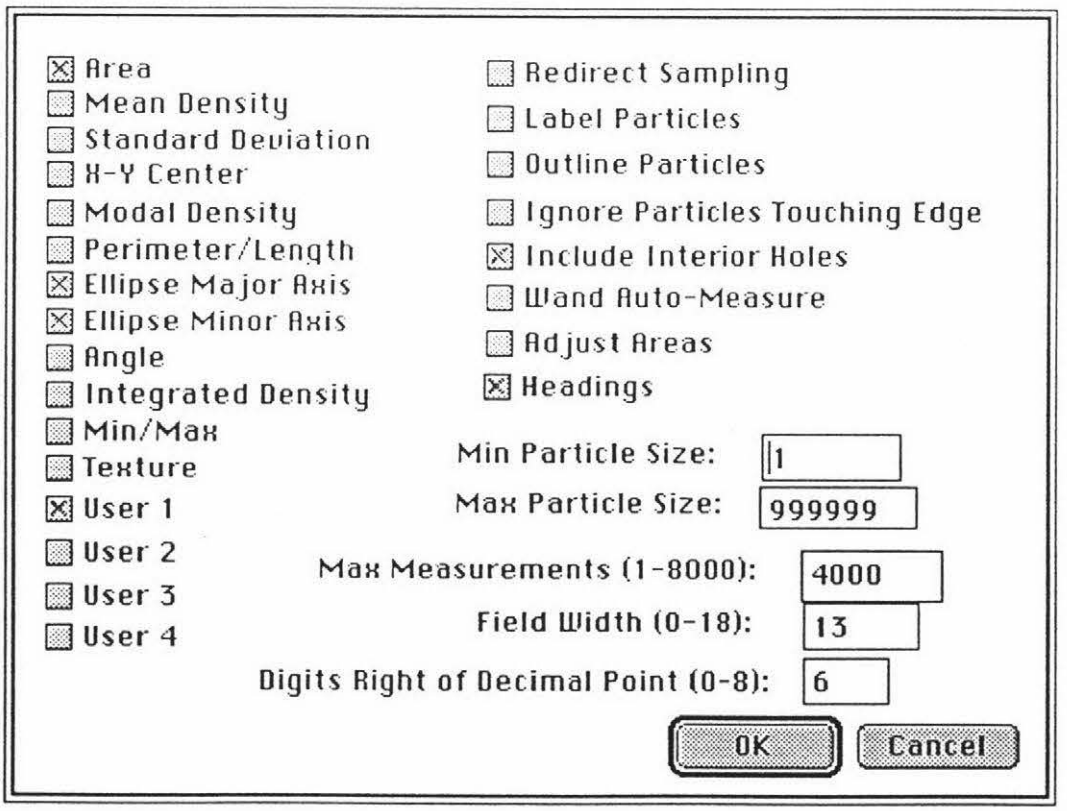


Figure 34 - Options Dialogue Box

The results in figure 33 show that there were 10 voids in the image with the circularity for each void in the User1 column. The results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

To do more circularity analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Extend Image

When the **Extend** command is chosen from the **Measures** submenu, "Image" extends the second outermost layer of pixels in an image to the outermost layer.

Blobs (NB - Intended for binary images only)

When **Blobs** is chosen from the **Measures** submenu, "Image" counts and displays the number of voids in the image in the results window, as shown in figure 35. In order to get the blob count into the correct place, the **Area** and **User1** options must have been chosen first from the options dialogue box, as shown in figure 27.

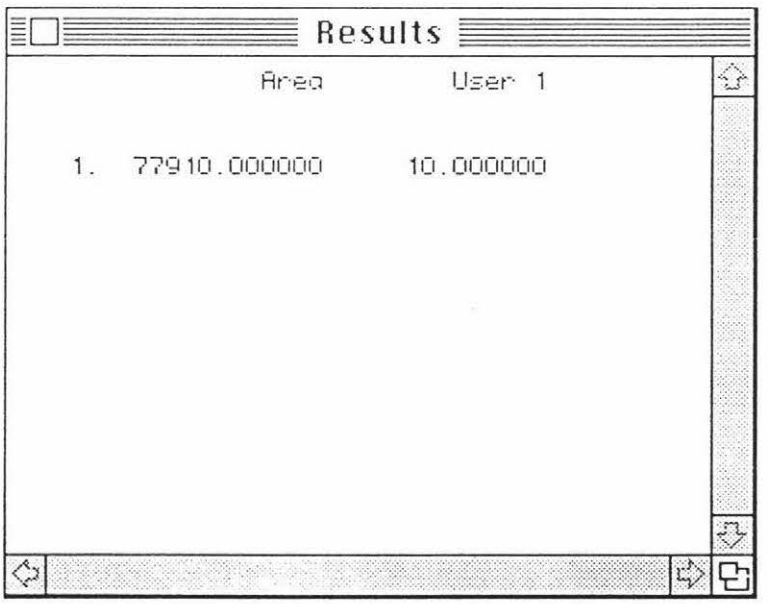


Figure 35 - Blob Count Results

The results in figure 35 show that there were 10 voids in the image with the count shown in the User1 column. The results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

Connectivity (NB - Intended for binary images only)

Connectivity looks at the changes in the number of voids in the image as successive layers of pixels are added to the voids using a MAX filter. The final result of the connectivity analysis is one single blob where all the original voids in the image have been connected together.

When **Connectivity** is chosen from the **Measures** submenu, "Image" calculates and displays the number of voids in the image for each iteration of the MAX filter in the results window, as shown in figure 36. In order to get the connectivity results into the correct place, the **Area** and **User1** options must have been chosen first from the options dialogue box, as shown in figure 27.

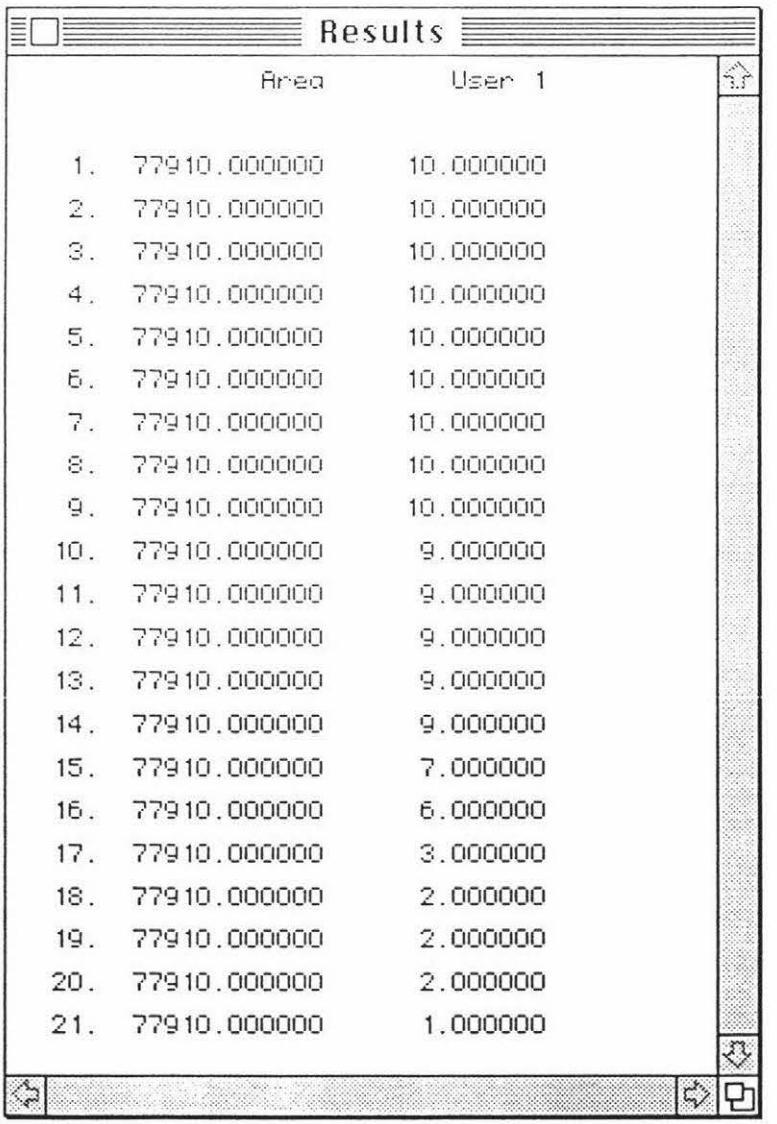


Figure 36 - Connectivity Results

The results in figure 36 show that 21 iterations of the MAX filter were needed to join the voids in the image together. The void count per iteration are shown in the User1 column. The results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10. A plot of the void count as a function of the number of iterations of the MAX filter can be plotted in "Excel".

To do more connectivity analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Texture Primitives (NB - Intended for binary images only)

This method uses various attributes of the texture primitives (in this case the voids) such as the area, perimeter, compactness, eccentricity, and direction to analyse the images. The area and perimeter of a void are self-explanatory, whereas the compactness is the ratio of the square of the perimeter to the area, the eccentricity is the inverse of the circularity, and the direction is the angle the major axis makes with the horizontal.

The steps "Image" takes to calculate the texture primitives are as follows:

- The attributes (area, perimeter, compactness, eccentricity, and direction) are calculated for each void in the image
- For each void the four nearest neighbours are found
- Primitive attribute co-occurence matrices are constructed for each attribute
- Second-order statistics are calculated from these co-occurence matrices

The first two steps have been discussed earlier. The final two steps resemble statistical texture analysis.

For each of the attributes, the values are divided into N=16 intervals. These intervals form an N x N co-occurence matrix for each attribute. To construct the co-occurence matrix P for the area attribute say, each void and it four neighbours are looked at. If the area for a particular void is in interval i and the nearest neighbour's area is in interval j, then we add 1 to $P(i,j)$, the (i,j)th entry of the area co-occurence matrix. Finally, the matrix is normalised.

From the co-occurence matrix for each attribute, the following second-order textural statistics are calculated:

- Angular Second Moment (ASM)
- Entropy (ENT)

- Inverse Different Moment (IDM)
- Contrast (CON)

When **Texture Primitives** is chosen from the **Measures** submenu, "Image" calculates a table of 20 texture primitives, as shown in figure 37. In order to get the texture primitives results into the correct place, the **Perimeter, Ellipse Major Axis, Ellipse Minor Axis**, and **Texture** options must have been chosen first from the options dialogue box, as shown in figure 38.

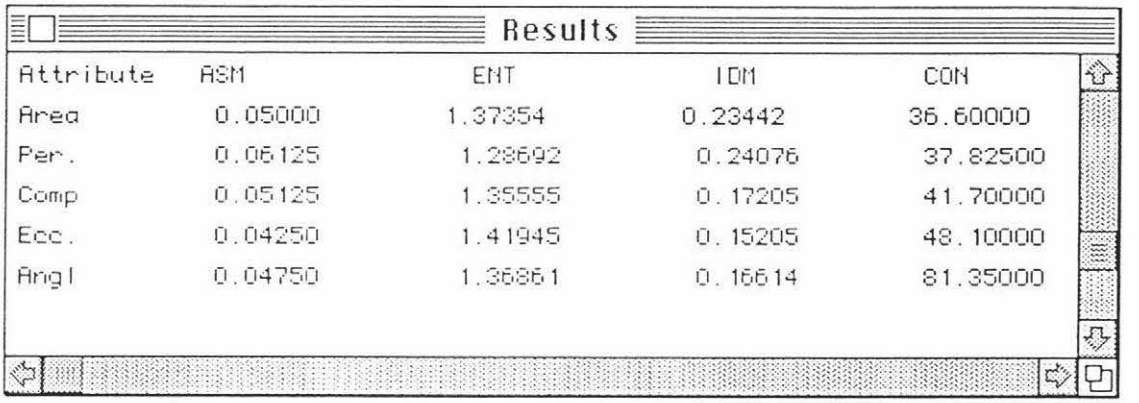


Figure 37 - Texture Primitives Results

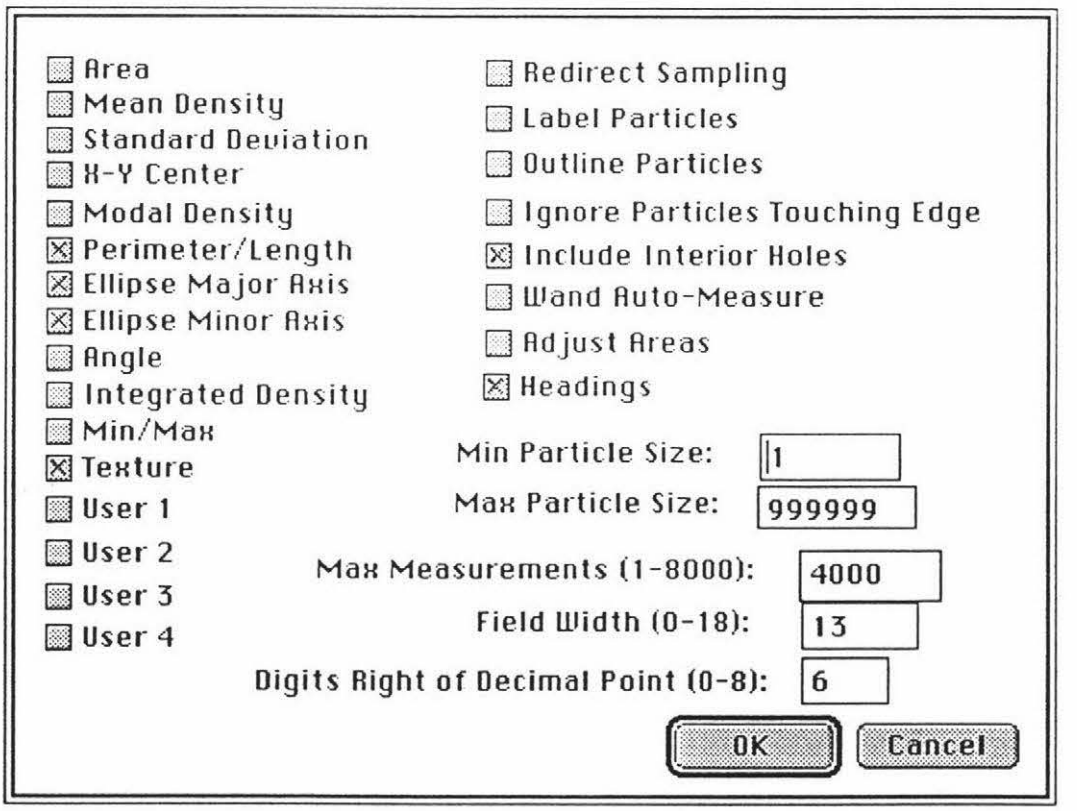


Figure 38 - Options Dialogue Box

The results in figure 38 show the texture primitives table. The results window can be saved as an "Excel" text file by choosing **Save As..** from the **File** menu and saving as **measurements**. This is shown in figure 10.

To do more texture primitive analysis on other images, the results will need to be cleared by choosing **Reset** from the **Analyze** menu, as shown in figure 11.

Macros [11]

The following list is all the macro commands I have added to Image Anode to allow one to write macros using the commands from the **User** menu.

REMEMBER for the Texture Measures to select a Region of Interest (ROI) first using the SelectAll macro command

REMEMBER for all measures to select the appropriate options first using the SelectOptions macro command

DoSGLDM(d)	This macro command does the **SGLDM...** command with an intersample spacing of d
DoGLDM(d)	This macro command does the **GLDM...** command with an intersample spacing of d
DoGLRLM	This macro command does the **GLRLM...** command
DoNGLDM(d,a)	This macro command does the **GLDM...** command with an intersample spacing of d and a difference of a
DoRank(r)	This macro command does the **Rank Filter** command with a rank of r
DoRange(r1,r2)	This macro command does the **Range Filter** command with ranks of r1 and r2
AreaFactor	This macro command does the **Area Factor** command
DistFactor	This macro command does the **Distance Factor** command
DoWA	This macro command does the **Weighted Area** command
Neighbour	This macro command does the **Neighbour Analysis** command

Neighbour2	This macro command does the **Neighbour Analysis 2** command
DoCircularity	This macro command does the **Circularity** command
DoExtend	This macro command does the **Extend Image** command
DoBlobs	This macro command does the **Blobs** command
DoConnectivity	This macro command does the **Connectivity** command
DoPrimitives	This macro command does the **Texture Primitives** command
Shutdown	This macro command turns the computer off

APPENDIX B - Source Code

```
const
 MaxMaxRegions = 8000;
type
 RealArray = array[0..MaxMaxRegions] of real;
 User2Type = RealArray;
 User3Type = RealArray;
 User4Type = RealArray;
 User5Type = RealArray;
 TextureResults = record
   TF: integer;
   TextureData: array[1..MaxNumData, 1..MaxTextHead] of real;
   Spacing: integer;
   qw: integer;
 end;
 TextureRecordArray = array[0..4000] of TextureResults;
 User1Type = TextureRecordArray;
 User1Ptr = ^User1Type;
 User2Ptr = ^User2Type;
 User3Ptr = ^User3Type;
 User4Ptr = ^User4Type;
 User5Ptr = ^User5Type;
 matphlinetype = array[0..255] of real;
 matph = array[0..255] of matphlinetype;
 matptr = ^matph;
 denfn_0type = array[0..255] of real;
 denfn_0ptr = ^denfn_0type;
 denfn_45type = array[0..255] of real;
 denfn_45ptr = ^denfn_45type;
 denfn_90type = array[0..255] of real;
 denfn_90ptr = ^denfn_90type;
 denfn_135type = array[0..255] of real;
 denfn_135ptr = ^denfn_135type;

var
 User1: User1Ptr;
 User2: User2Ptr;
 User3: User3Ptr;
 User4: User4Ptr;
 User5: User5Ptr;
 a1, d, e, f1, g1, rank, rank_h, rank_l, Conn: longint;
 Texture: TextureResults;
 matphp: matptr;
 denfn_0: denfn_0ptr;
 denfn_45: denfn_45ptr;
 denfn_90: denfn_90ptr;
 denfn_135: denfn_135ptr;
 width, height, ucol, lcol, urow, lrow, gmin, gmax, gwidth: longint;
 wasaccepted: boolean;

procedure GetD; {This procedure gets the Intersample spacing for the SGLDM procedure}

begin
 D := GetInt('Distance d (SGLDM):', 0, wasaccepted);
 if wasaccepted then
   exit(GetD);
```

```
end;


procedure GetE;  {This procedure gets the Intersample spacing for the GLDM procedure}

begin
 e := GetInt('Distance d (GLDM):', 0, wasaccepted);
 if wasaccepted then
   exit(GetE);
end;

procedure GetG1;  {This procedure gets the Intersample spacing for the NGLDM procedure}

begin
 g1 := GetInt('Distance d (NGLDM):', 0, wasaccepted);
 if wasaccepted then
   exit(GetG1);
end;

procedure GetA1;  {This procedure gets the Difference for the NGLDM procedure}

begin
 a1 := GetInt(' A (SGLDM):', 0, wasaccepted);
 if wasaccepted then
   exit(GetA1);
end;

procedure  GetRank; {This procedure gets the rank for the rank filter procedure}

begin
 Rank := GetInt('Filter Rank?:', 0, wasaccepted);
 if wasaccepted then
   exit(GetRank);
end;

procedure GetRank_h;  {This procedure gets the first rank for the range filter procedure}

begin
 Rank_h := GetInt('First Rank?:', 0, wasaccepted);
 if wasaccepted then
   exit(GetRank_h);
end;

procedure GetRank_l;  {This procedure gets the second rank for the range filter procedure}

begin
 Rank_l := GetInt('Second Rank?:', 0, wasaccepted);
 if wasaccepted then
   exit(GetRank_l);
end;

procedure DoSGLDM;   {SGLDM}
 label
   100;
 var
   m, lrow: integer;
```

{THIS PROCEDURE READS AN IMAGE LINE BY LINE AND CALCULATES THE MATPH MATRIX FOR ZERO DEGREE ANGLE}

```
procedure sgldm_zero (var matphp: matptr; D: integer);
 var
  hloc, vloc, i, j: integer;
  R, L: integer;
  line: linetype;
begin
 with info^.roirect do
  begin
   for vloc := top to bottom - 1 do
    begin
      getline(left, vloc, width, line);
    for hloc := 0 to width - 1 do
     begin
     R := hloc + D;
     L := hloc - D;
    if (R <= width - 1) and (L >= 0) then
    begin
     MATPHP^[LINE[HLOC], LINE[R]] := MATPHP^[LINE[HLOC], LINE[R]] + 1;
     MATPHP^[LINE[HLOC], LINE[L]] := MATPHP^[LINE[HLOC], LINE[L]] + 1;
    end
    else if (R > width - 1) then
     MATPHP^[LINE[HLOC], LINE[L]] := MATPHP^[LINE[HLOC], LINE[L]] + 1
    else
     MATPHP^[LINE[HLOC], LINE[R]] := MATPHP^[LINE[HLOC], LINE[R]] + 1;
    end;
   end;
 end;
end;
```

{THIS PROCEDURE READS AN IMAGE LINE BY LINE AND CALCULATES THE MATPH MATRIX FOR NINETY DEGREE ANGLE}

```
procedure sgldm_ninety (var matphp: matptr; D: integer);
 var
  hloc, vloc, up, down, i, j: integer;
  line: linetype;
begin
 with info^.roirect do
  begin
   for hloc := left to right - 1 do
    begin
      getcolumn(hloc, top, height, line);
    for vloc := 0 to height - 1 do
     begin
     up := vloc - D;
     down := vloc + D;
    if (up >= 0) and (down <= height - 1) then
    begin
     MATPHP^[LINE[VLOC], LINE[down]] := MATPHP^[LINE[VLOC], LINE[down]] + 1;
     MATPHP^[LINE[VLOC], LINE[up]] := MATPHP^[LINE[VLOC], LINE[up]] + 1;
    end
    else if (down > height - 1) then
     MATPHP^[LINE[VLOC], LINE[up]] := MATPHP^[LINE[VLOC], LINE[up]] + 1
    else
     MATPHP^[LINE[VLOC], LINE[down]] := MATPHP^[LINE[VLOC], LINE[down]] + 1;
```

```
    end;
   end;
  end;
end;
```

{THIS PROCEDURE READS AN IMAGE LINE BY LINE AND CALCULATES THE MATPH MATRIX FOR FORTYFIVE DEGREE ANGLE}

```
procedure sgldm_fortyfive (var matphp: matptr; D: integer);
 var
  i, j, up, down, l, r: integer;
begin
 with info^.roirect do
  begin
   for i := top to bottom - 1 do
    for j := left to right - 1 do
     begin
      r := j + D;
      l := j - D;
      down := i + D;
      up := i - D;
     if (r <= ucol) and (up >= lrow) and (l >= lcol) and (down <= urow) then
     begin
        MATPHP^[mygetpixel(j, i), mygetpixel(r, up)] := MATPHP^[mygetpixel(j, i),
mygetpixel(r, up)] + 1;
        MATPHP^[mygetpixel(j, i), mygetpixel(l, down)] := MATPHP^[mygetpixel(j, i),
mygetpixel(l, down)] + 1;
     end
     else
     begin
     if (r <= ucol) and (up >= lrow) then
        MATPHP^[mygetpixel(j, i), mygetpixel(r, up)] := MATPHP^[mygetpixel(j, i),
mygetpixel(r, up)] + 1;
     if (l >= lcol) and (down <= urow) then
        MATPHP^[mygetpixel(j, i), mygetpixel(l, down)] := MATPHP^[mygetpixel(j, i),
mygetpixel(l, down)] + 1;
     end;
    end;
  end;
end;
```

{THIS PROCEDURE READS AN IMAGE LINE BY LINE AND CALCULATES THE MATPH MATRIX FOR 135 DEGREE ANGLE}

```
procedure sgldm_onethreefive (var matphp: matptr; D: integer);
 var
  i, j, up, down, l, r: integer;
begin
 with info^.roirect do
  begin
   for i := top to bottom - 1 do
    for j := left to right - 1 do
     begin
      r := j + D;
      l := j - D;
      down := i + D;
      up := i - D;
     if (r <= ucol) and (l >= lcol) and (up >= lrow) and (down <= urow) then
```

```
    begin
      MATPHP^[mygetpixel(j, i), mygetpixel(r, down)] := MATPHP^[mygetpixel(j, i),
mygetpixel(r, down)] + 1;
      MATPHP^[mygetpixel(j, i), mygetpixel(l, up)] := MATPHP^[mygetpixel(j, i),
mygetpixel(l, up)] + 1;
    end
    else
    begin
    if (r <= ucol) and (down <= urow) then
      MATPHP^[mygetpixel(j, i), mygetpixel(r, down)] := MATPHP^[mygetpixel(j, i),
mygetpixel(r, down)] + 1;
    if (l >= lcol) and (up >= lrow) then
      MATPHP^[mygetpixel(j, i), mygetpixel(l, up)] := MATPHP^[mygetpixel(j, i),
mygetpixel(l, up)] + 1;
    end;
   end;
  end;
end;

{THIS PROCEDURE CALCULATES THE TEXTURE ENERGY OF A GIVEN SGLD MATRIX}

procedure texture_energy (var matphp: matptr; D: integer; nx: longint; ny: longint;
angle: integer; var energy: real);
 var
  i, j: integer;
  ri, toten, suben: real;
begin
 with info^.roirect do
  begin
   toten := 0.0;
  case angle of
   0:
     ri := 2 * ny * (nx - d);
    45, 135:
     ri := 2 * (ny - d) * (nx - d);
   90:
     ri := 2 * nx * (ny - d);
  end;
  for i := gmin to gmax do
   begin
    suben := 0.0;
    for j := gmin to gmax do
     begin
      suben := suben + (matphp^[i, j] * matphp^[i, j]);
     end;
     toten := toten + suben;
   end;
    toten := toten / (ri * ri);
   energy := toten;
  end;
end;


{THIS PROCEDURE CALCULATES THE TEXTURE ENTROPY OF A GIVEN SGLD MATRIX}

procedure texture_entropy (var matphp: matptr; d: integer; nx: longint; ny: longint;
angle: integer; var entropy: real);
 var
```

```
   i, j: integer;
   ri, totent, subent, l10: real;
begin
 with info^.roirect do
  begin
   totent := 0.0;
   subent := 0.0;
   l10 := ln(10);
  case angle of
   0:
     ri := 2 * ny * (nx - d);
    45, 135:
     ri := 2 * (ny - d) * (nx - d);
   90:
     ri := 2 * nx * (ny - d);
  end;
  for i := gmin to gmax do
   for j := gmin to gmax do
    begin
     if (matphp^[i, j] <> 0) then
        subent := subent + (matphp^[i, j] * ((ln(matphp^[i, j] / ri)) / l10));
    end;
    totent := (subent / ri);
    entropy := -totent;
 end;
end;
```

{THIS PROCEDURE CALCULATES THE TEXTURE INERTIA OF A GIVEN SGLD MATRIX}

```
procedure texture_inertia (var matphp: matptr; d: integer; nx: longint; ny: longint; angle:
integer; var inertia: real);
 var
   i, j: integer;
   ri, totinert: real;
begin
 with info^.roirect do
  begin
   totinert := 0.0;
  case angle of
   0:
     ri := 2 * ny * (nx - d);
    45, 135:
     ri := 2 * (ny - d) * (nx - d);
   90:
     ri := 2 * nx * (ny - d);
  end;
  for i := gmin to gmax do
   for j := gmin to gmax do
    begin
       totinert := totinert + (((i - j) / 1.0) * (i - j) * matphp^[i, j]);
    end;
    totinert := totinert / ri;
    inertia := totinert;
 end;
end;
```

{THIS PROCEDURE CALCULATES THE LOCAL HOMOGENEITY OF A GIVEN SGLD MATRIX}

```
procedure texture_homog (var matphp: matptr; d: integer; nx: longint; ny: longint; angle:
integer; var homog: real);
 var
  i, j: integer;
  ri, result: real;
begin
 with info^.roirect do
  begin
   result := 0.0;
  case angle of
   0:
     ri := 2 * ny * (nx - d);
   45, 135:
     ri := 2 * (ny - d) * (nx - d);
   90:
     ri := 2 * nx * (ny - d);
  end;
  for i := gmin to gmax do
   for j := gmin to gmax do
    begin
       result := result + (matphp^[i, j] / (1 + ((i - j) / 1.0) * (i - j)));
    end;
    result := result / ri;
   homog := result;
 end;
end;
```

{THIS PROCEDURE CALCULATES THE TEXTURE CORRELATION OF A GIVEN SGLD MATRIX}

```
procedure texture_correlation (var matphp: matptr; d: integer; nx: longint; ny: longint;
angle: integer; var correlation: real);
 var
  i, j: integer;
  ri, sdx, sdy, result: real;
  x, y, subtotal, total: real;
```

{THIS FUNCTION CALCULATES THE MEAN X OF A GIVEN SGLD MATRIX}

```
function meanx (p: matptr; ri: real): real;
 var
  i, j: integer;
  tempmean, sumx: real;
begin
 with info^.roirect do
  begin
   tempmean := 0.0;
  for i := gmin to gmax do
   begin
    sumx := 0.0;
    for j := gmin to gmax do
      sumx := sumx + (matphp^[i, j] / ri);
      tempmean := tempmean + ((i - lrow) * sumx);
   end;
   meanx := tempmean
  end;
end;
```

{THIS FUNCTION CALCULATES THE MEAN Y OF A GIVEN SGLD MATRIX}

```
function meany (p: matptr; ri: real): real;
 var
  i, j: integer;
  tempmean, sumy: real;
begin
 with info^.roirect do
  begin
   tempmean := 0.0;
  for j := gmin to gmax do
   begin
    sumy := 0.0;
   for i := gmin to gmax do
     sumy := sumy + (matphp^[i, j] / ri);
     tempmean := tempmean + (sumy * (j - lcol));
   end;
   meany := tempmean;
 end;
end;
```

{THIS FUNCTION CALCULATES THE STD DEV X OF A GIVEN SGLD MATRIX}

```
function stdevx (p: matptr; ri: real; meanx: real): real;
 var
  i, j: integer;
  sum, variance: real;
begin
 with info^.roirect do
  begin
   variance := 0.0;
  for i := gmin to gmax do
   begin
    sum := 0.0;
   for j := gmin to gmax do
     sum := sum + (matphp^[i, j] / ri);
     variance := variance + ((sqr(i - lrow - meanx)) * sum);
   end;
    stdevx := sqrt(variance);
 end;
end;
```

{THIS FUNCTION CALCULATES THE STD DEV Y OF A GIVEN SGLD MATRIX}

```
function stdevy (p: matptr; ri: real; meany: real): real;
 var
  i, j, width: integer;
  sum, variance: real;
begin
 with info^.roirect do
  begin
   variance := 0.0;
  for j := gmin to gmax do
   begin
    sum := 0.0;
   for i := gmin to gmax do
     sum := sum + (matphp^[i, j] / ri);
     variance := variance + ((sqr(j - lcol - meany)) * sum);
   end;
```

```
      stdevy := sqrt(variance);
    end;
  end;

begin
 with info^.roirect do
  begin
   case angle of
    0:
      ri := 2 * ny * (nx - d);
    45, 135:
      ri := 2 * (ny - d) * (nx - d);
    90:
      ri := 2 * nx * (ny - d);
   end;
    x := meanx(matphp, ri);
    y := meany(matphp, ri);
    sdx := stdevx(matphp, ri, x);
    sdy := stdevy(matphp, ri, y);
   total := 0;
  if (sdx = 0.0) or (sdy = 0.0) then
    result := 1.0
  else
   begin
    for i := gmin to gmax do
     begin
      subtotal := 0.0;
     for j := gmin to gmax do
     begin
        subtotal := subtotal + ((i - lrow - x) * (j - lcol - y) * (matphp^[i, j] / ri));
     end;
      total := total + subtotal;
     end;
      result := total / (sdx * sdy);
   end;
    correlation := result;
  end;
end;

{THIS IS THE MAIN PROCEDURE OF THE PROGRAM}

procedure process_image (var matphp: matptr; D: integer);
 var
  zeroline: matphlinetype;
  line: linetype;
  nx, ny: longint;
  i, j, m, n, vloc, hloc, k, angle: integer;
   energy, entropy, homog, inertia, correlation: real;
   tot1, tot2, tot3, tot4, tot5: real;
  sumf1sq, sumf2sq, sumf3sq, sumf4sq, sumf5sq: real;
  maxen, maxent, maxinert, maxhomog, maxcor: real;
  minen, minent, mininert, minhomog, mincor: real;
  rangef1, rangef2, rangef3, rangef4, rangef5: real;
  sd1, sd2, sd3, sd4, sd5: real;

begin
 with info^.roirect do
  begin
```

```
gmax := 0;
gmin := 255;
for vloc := top to bottom - 1 do
 begin
   getline(left, vloc, width, line);
 for hloc := 0 to width - 1 do
  begin
  if line[hloc] >= gmax then
    gmax := line[hloc];                {Finds the maximum gray level in the image}
  if line[hloc] <= gmin then
    gmin := line[hloc];                {Finds the minimum gray level in the image}
  end;
end;
gwidth := gmax - gmin + 1;

with Texture do
 begin
  TF := 1;            {Tells program which table to use - TF1 is the SGLDM table}
 end;

angle := 0;
k := 1;

 tot1 := 0.0;
 tot2 := 0.0;
 tot3 := 0.0;
 tot4 := 0.0;
 tot5 := 0.0;

minen := 1.0;
 minent := 1.0e+10;
 mininert := 1.0e+10;
minhomog := 1.0;
 mincor := 1.0;

maxen := 0.0;
 maxent := 0.0;
 maxinert := 0.0;
maxhomog := 0.0;
 maxcor := 0.0;

 sumf1sq := 0.0;
 sumf2sq := 0.0;
 sumf3sq := 0.0;
 sumf4sq := 0.0;
 sumf5sq := 0.0;

 nx := urow - lrow + 1;
 ny := ucol - lcol + 1;

for j := 0 to 255 do
  zeroline[j] := 0;

repeat
 for i := 0 to 255 do
   matphp^[i] := zeroline;             {Clears the MATPHP matrix before use}
case angle of
 0:
```

```
    sgldm_zero(matphp, D);
  45:
     sgldm_fortyfive(matphp, D);
  90:
    sgldm_ninety(matphp, D);
  135:
    sgldm_onethreefive(matphp, D);
end;

  texture_energy(matphp, D, nx, ny, angle, energy);
  texture_entropy(matphp, D, nx, ny, angle, entropy);
  texture_inertia(matphp, D, nx, ny, angle, inertia);
 texture_homog(matphp, D, nx, ny, angle, homog);
  texture_correlation(matphp, D, nx, ny, angle, correlation);

with Texture do
 begin     {Puts the results into the texture record for display to the results window}
    TextureData[k, 1] := energy;
    TextureData[k, 2] := entropy;
    TextureData[k, 3] := inertia;
   TextureData[k, 4] := homog;
    TextureData[k, 5] := correlation;
 end;

 tot1 := tot1 + energy;       {Sums the results for use in determining the averages}
 tot2 := tot2 + entropy;
 tot3 := tot3 + inertia;
tot4 := tot4 + homog;
 tot5 := tot5 + correlation;

 sumf1sq := sumf1sq + energy * energy;  {Likewise sum of squares for the standard}
                                                          {deviation}
 sumf2sq := sumf2sq + entropy * entropy;
 sumf3sq := sumf3sq + inertia * inertia;
sumf4sq := sumf4sq + homog * homog;
 sumf5sq := sumf5sq + correlation * correlation;

if (energy > maxen) then          {Finds the maximum and minimum feature values}
  maxen := energy;
if (energy < minen) then
  minen := energy;
if (entropy > maxent) then
  maxent := entropy;
if (entropy < minent) then
  minent := entropy;
if (inertia > maxinert) then
   maxinert := inertia;
if (inertia < mininert) then
   mininert := inertia;
if (homog > maxhomog) then
 maxhomog := homog;
if (homog < minhomog) then
 minhomog := homog;
if (correlation > maxcor) then
   maxcor := correlation;
if (correlation < mincor) then
   mincor := correlation;
```

```
    k := k + 1;
    angle := angle + 45;
  until (k > 4) or (angle > 135);

  rangef1 := maxen - minen;          {Finds the ranges}
  rangef2 := maxent - minent;
  rangef3 := maxinert - mininert;
  rangef4 := maxhomog - minhomog;
  rangef5 := maxcor - mincor;

  sd1 := sqrt(sumf1sq / 4 - (tot1 * tot1) / 16);          {Finds the standard deviations}
  sd2 := sqrt(sumf2sq / 4 - (tot2 * tot2) / 16);
  sd3 := sqrt(sumf3sq / 4 - (tot3 * tot3) / 16);
  sd4 := sqrt(sumf4sq / 4 - (tot4 * tot4) / 16);
  sd5 := sqrt(sumf5sq / 4 - (tot5 * tot5) / 16);

  tot1 := tot1 / 4;                  {Finds the means}
  tot2 := tot2 / 4;
  tot3 := tot3 / 4;
  tot4 := tot4 / 4;
  tot5 := tot5 / 4;

with Texture do
  begin     {Puts results into the texture record for display to the results window}
    TextureData[5, 1] := tot1;
    TextureData[5, 2] := tot2;
    TextureData[5, 3] := tot3;
    TextureData[5, 4] := tot4;
    TextureData[5, 5] := tot5;

    TextureData[6, 1] := maxen;
    TextureData[6, 2] := maxent;
    TextureData[6, 3] := maxinert;
    TextureData[6, 4] := maxhomog;
    TextureData[6, 5] := maxcor;

    TextureData[7, 1] := minen;
    TextureData[7, 2] := minent;
    TextureData[7, 3] := mininert;
    TextureData[7, 4] := minhomog;
    TextureData[7, 5] := mincor;

    TextureData[8, 1] := rangef1;
    TextureData[8, 2] := rangef2;
    TextureData[8, 3] := rangef3;
    TextureData[8, 4] := rangef4;
    TextureData[8, 5] := rangef5;

    TextureData[9, 1] := sd1;
    TextureData[9, 2] := sd2;
    TextureData[9, 3] := sd3;
    TextureData[9, 4] := sd4;
    TextureData[9, 5] := sd5;

  Spacing := d;

  Measure;
  ShowResults;
```

```
    end;
   end;
  end;

{THIS IS THE MAIN PART OF THE PROGRAM}

begin
 with info^.roirect do
  begin
    width := right - left;
   height := bottom - top;
    lrow := top;
    urow := bottom - 1;
    lcol := left;
    ucol := right - 1;
  if wasaccepted then
    exit(DoSGLDM);
  if D > width then
    begin
      putmessage(concat('Intersample Spacing D ', long2str(D), ' is greater than width ',
long2str(width), '. Please try again'));
     exit(DoSGLDM);
    end;
     matphp := matptr(NewPtr(sizeof(matph)));
  if matphp = nil then
    goto 100;
   process_image(matphp, D);
    DisposPtr(Ptr(matphp));
   matphp := nil;
 end;
100:
end;

procedure DoGLDM;    {GLDM}
 label
  100;

{THIS PROCEDURE COMPUTES THE GRAY LEVEL DENSITY FUNCTIONS FROM THE IMAGE}

 procedure compute_den_fn (var denfn_0: denfn_0ptr; var denfn_45: denfn_45ptr; var
denfn_90: denfn_90ptr; var denfn_135: denfn_135ptr; e: integer);
 type
   mlinetype = array[0..500] of longint;
   mtype = array[0..500] of  mlinetype;
   mptr = ^mtype;
 var
   zeroline: mlinetype;
   m: mptr;
   line: linetype;
   i, j, x: integer;
   nf1, nf2, nf3, nf4: longint;
 begin
 with info^.roirect do
  begin
     m := mptr(NewPtr(sizeof(mtype)));
  if m = nil then
    goto 100;
```

```
  for x := 0 to 255 do
    zeroline[x] := 0;

  for i := top to bottom - 1 do
   for j := left to right - 1 - e do
      m^[i, j] := abs(mygetpixel(j, i) - mygetpixel(j + e, i));
   nf1 := (urow - lrow + 1) * (ucol - lcol - e + 1);

  for i := top to bottom - 1 do
   for j := left to right - 1 - e do
      denfn_0^[m^[i, j]] := denfn_0^[m^[i, j]] + 1;

  for i := top + e to bottom - 1 do
   for j := left to right - 1 - e do
      m^[i, j] := abs(mygetpixel(j, i) - mygetpixel(j + e, i - e));
   nf2 := (urow - lrow - e + 1) * (ucol - lcol - e + 1);

  for i := top + e to bottom - 1 do
   for j := left to right - 1 - e do
      denfn_45^[m^[i, j]] := denfn_45^[m^[i, j]] + 1;

  for i := top to bottom - 1 - e do
   for j := left to right - 1 do
      m^[i, j] := abs(mygetpixel(j, i) - mygetpixel(j, i + e));
   nf3 := (urow - lrow - e + 1) * (ucol - lcol + 1);

  for i := top to bottom - 1 - e do
   for j := left to right - 1 do
      denfn_90^[m^[i, j]] := denfn_90^[m^[i, j]] + 1;

  for i := top to bottom - 1 - e do
   for j := left to right - 1 - e do
      m^[i, j] := abs(mygetpixel(j, i) - mygetpixel(j + e, i + e));
   nf4 := (urow - lrow - e + 1) * (ucol - lcol - e + 1);

  for i := top to bottom - 1 - e do
   for j := left to right - 1 - e do
      denfn_135^[m^[i, j]] := denfn_135^[m^[i, j]] + 1;

  for i := 0 to 255 do
   begin
     denfn_0^[i] := denfn_0^[i] / nf1;
     denfn_45^[i] := denfn_45^[i] / nf2;
     denfn_90^[i] := denfn_90^[i] / nf3;
     denfn_135^[i] := denfn_135^[i] / nf4;
   end;

    DisposPtr(Ptr(m));
   m := nil;
 end;
end;
```

{THIS PROCEDURE CALCULATES THE FEATURE SET FROM THE DENSITY FUNCTIONS}

```
procedure GLDM_features (e: integer);
 var
  asm, ent, mean, idm, con: array[1..4] of real;
  i, j: longint;
```

```
  l10, sumcon, sumasm, sument, sumean, sumidm: real;
  avecon, aveasm, aveent, avemean, aveidm: real;
begin
 with info^.roirect do
 begin
   l10 := ln(10);
 if (e < width) or (e < height) then
  begin
   for i := 0 to 255 do
   begin
     denfn_0^[i] := 0.0;
     denfn_45^[i] := 0.0;
     denfn_90^[i] := 0.0;
     denfn_135^[i] := 0.0;
   end;
   for i := 1 to 4 do
   begin
     asm[i] := 0.0;
     mean[i] := 0.0;
     ent[i] := 0.0;
     idm[i] := 0.0;
     con[i] := 0.0;
   end;

   with Texture do
    begin
    TF := 2;            {Tells program which table to use - TF2 is the GLDM table}
   end;

   sumcon := 0.0;
   sumasm := 0.0;
   sumean := 0.0;
   sument := 0.0;
   sumidm := 0.0;

    compute_den_fn(denfn_0, denfn_45, denfn_90, denfn_135, e);

   for i := 0 to 255 do
   begin
     con[1] := con[1] + ((i * i) * denfn_0^[i]);
     con[2] := con[2] + ((i * i) * denfn_45^[i]);
     con[3] := con[3] + ((i * i) * denfn_90^[i]);
     con[4] := con[4] + ((i * i) * denfn_135^[i]);

     asm[1] := asm[1] + (denfn_0^[i] * denfn_0^[i]);
     asm[2] := asm[2] + (denfn_45^[i] * denfn_45^[i]);
     asm[3] := asm[3] + (denfn_90^[i] * denfn_90^[i]);
     asm[4] := asm[4] + (denfn_135^[i] * denfn_135^[i]);

   if (denfn_0^[i] <> 0) then
      ent[1] := ent[1] - (denfn_0^[i] * (ln(denfn_0^[i]) / l10));
   if (denfn_45^[i] <> 0) then
      ent[2] := ent[2] - (denfn_45^[i] * (ln(denfn_45^[i]) / l10));
   if (denfn_90^[i] <> 0) then
      ent[3] := ent[3] - (denfn_90^[i] * (ln(denfn_90^[i]) / l10));
   if (denfn_135^[i] <> 0) then
      ent[4] := ent[4] - (denfn_135^[i] * (ln(denfn_135^[i]) / l10));
```

```
      mean[1] := mean[1] + (i * denfn_0^[i]);
      mean[2] := mean[2] + (i * denfn_45^[i]);
      mean[3] := mean[3] + (i * denfn_90^[i]);
      mean[4] := mean[4] + (i * denfn_135^[i]);

      idm[1] := idm[1] + (denfn_0^[i] / (i * i + 1));
      idm[2] := idm[2] + (denfn_45^[i] / (i * i + 1));
      idm[3] := idm[3] + (denfn_90^[i] / (i * i + 1));
      idm[4] := idm[4] + (denfn_135^[i] / (i * i + 1));
    end;

    for i := 1 to 4 do
    begin
     sumcon := sumcon + con[i];                {Sums the results}
     sument := sument + ent[i];
     sumidm := sumidm + idm[i];
     sumasm := sumasm + asm[i];
     sumean := sumean + mean[i];
    end;

    avecon := sumcon / 4.0;              {Finds the mean results}
    aveent := sument / 4.0;
    aveidm := sumidm / 4.0;
    aveasm := sumasm / 4.0;
    avemean := sumean / 4.0;

   with Texture do
    begin            {Puts the results into the texture record for display}
    for i := 1 to 4 do
    begin
      TextureData[i, 1] := con[i];
      TextureData[i, 2] := asm[i];
      TextureData[i, 3] := ent[i];
      TextureData[i, 4] := mean[i];
      TextureData[i, 5] := idm[i];
    end;
      TextureData[5, 1] := avecon;
      TextureData[5, 2] := aveasm;
      TextureData[5, 3] := aveent;
      TextureData[5, 4] := avemean;
      TextureData[5, 5] := aveidm;

    Spacing := e;

    end;
    Measure;
    ShowResults;
   end;
  end;
end;


{THIS IS THE MAIN PART OF THE PROGRAM}

begin
 with info^.roirect do
  begin
```

```
    width := right - left;
    height := bottom - top;
    lrow := top;
    urow := bottom - 1;
    lcol := left;
    ucol := right - 1;
   if wasaccepted then
    exit(DoGLDM);
   if E > width then
    begin
      putmessage(concat('Intersample Spacing D ', long2str(E), ' is greater than width ',
long2str(width), '. Please try again'));
    exit(DoGLDM);
   end;
    denfn_0 := denfn_0ptr(NewPtr(sizeof(denfn_0type)));
    denfn_45 := denfn_45ptr(NewPtr(sizeof(denfn_45type)));
    denfn_90 := denfn_90ptr(NewPtr(sizeof(denfn_90type)));
    denfn_135 := denfn_135ptr(NewPtr(sizeof(denfn_135type)));
   if denfn_0 = nil then
    goto 100;
   if denfn_45 = nil then
    goto 100;
   if denfn_90 = nil then
    goto 100;
   if denfn_135 = nil then
    goto 100;

    gldm_features(e);

    DisposPtr(Ptr(denfn_0));
    DisposPtr(Ptr(denfn_45));
    DisposPtr(Ptr(denfn_90));
    DisposPtr(Ptr(denfn_135));
   denfn_0 := nil;
   denfn_45 := nil;
   denfn_90 := nil;
   denfn_135 := nil;

 end;
100:
end;

procedure DoGLRLM;   {GLRLM}
 label
  100;
 const
  maxlen = 400;
 type
  mtype = array[1..20, 1..maxlen] of longint;
  mptr = ^mtype;
  imagelinetype = array[0..maxlen] of longint;
  imagetype = array[0..maxlen] of imagelinetype;
  imageptr = ^imagetype;
 var
  m: mptr;
  image: imageptr;
  i, j, k, p, q, maxsize, f1: integer;
  sre, lre, rpc, glnu, rlnu, rlnu1: real;
```

```
  sre2, lre2, rpc2, glnu2, rlnu2, rlnu3: real;
  sre90, lre90, rpc90, glnu90, rlnu90, rlnu900: real;
  sre135, lre135, rpc135, glnu135, rlnu135, rlnu1350: real;
  avesre, avelre, averpc, aveglnu, averlnu: real;
  wasaccepted: boolean;

begin
 with info^.roirect do
  begin
   f1 := 0;
   width := right - left;
   height := bottom - top;
   lrow := top;
   urow := bottom - 1;
   lcol := left;
   ucol := right - 1;
    m := mptr(NewPtr(sizeof(mtype)));
  if m = nil then
   goto 100;
   image := imageptr(NewPtr(sizeof(imagetype)));
  if image = nil then
   goto 100;

  if (height >= width) then
   maxsize := height
  else
   maxsize := width;

  if maxsize > 256 then
   putmessage('Image too big for GLRLM - Choose smaller region of interest');

  with Texture do
   begin
    TF := 3;                    {Tells program which table to use - TF3 is the GLRLM table}
   end;

  for i := lrow to urow do
   begin
    for j := lcol to ucol do
      image^[i, j] := trunc(mygetpixel(j, i) / 12.8) + 1;
   end;

  for i := lrow to urow do
   begin
     image^[i, lcol] := 0;
     image^[i, ucol] := 0;
   end;
  for j := lcol to ucol do
   begin
     image^[lrow, j] := 0;
     image^[urow, j] := 0;
   end;

  for p := 1 to 20 do
   begin
    for q := 1 to maxsize do
     m^[p, q] := 0;
   end;
```

```
for i := lrow + 1 to urow - 2 do
 begin
  for j := lcol + 1 to ucol - 2 do
   begin
    k := 0;
   if image^[i, j - 1] <> image^[i, j] then
   begin
   repeat
   k := k + 1;
   until image^[i, j] <> image^[i, j + k];
     m^[image^[i, j], k] := m^[image^[i, j], k] + 1;
   end;
  end;
 end;

 sre := 0;
 lre := 0;
 rpc := 0;
 rlnu := 0;
 rlnu1 := 0;
 glnu := 0;
for p := 1 to 20 do
 begin
  for q := 1 to maxsize do
   begin
     sre := sre + m^[p, q] / (longint(q) * q);
     lre := lre + m^[p, q] * longint(q) * q;
    rpc := rpc + m^[p, q];
   end;
   glnu := glnu + rpc * rpc;
 end;
for q := 1 to maxsize do
 begin
  for p := 1 to 20 do
   begin
     rlnu := rlnu + m^[p, q];
   end;
    rlnu1 := rlnu1 + rlnu * rlnu;
 end;
 sre := sre / rpc;
 lre := lre / rpc;
 glnu := glnu / rpc;
 rlnu := rlnu1 / rpc;
 rpc := rpc / ((urow - lrow + 1) * (ucol - lcol + 1));

with Texture do
 begin                    {Puts the results into the texture record for display}
   TextureData[1, 1] := sre;
   TextureData[1, 2] := lre;
   TextureData[1, 3] := glnu;
   TextureData[1, 4] := rlnu;
   TextureData[1, 5] := rpc;
 end;

 f1 := 1;

for i := lrow to urow do
```

```
 begin
  for j := lcol to ucol do
     image^[i, j] := trunc(mygetpixel(j, i) / 12.8) + 1;{Converts image to 20 grayscales}
 end;

for i := lrow to urow do
 begin
   image^[i, lcol] := 0;
   image^[i, ucol] := 0;
 end;
for j := lcol to ucol do
 begin
   image^[lrow, j] := 0;
   image^[urow, j] := 0;
 end;

for p := 1 to 20 do
 begin
  for q := 1 to maxsize do
    m^[p, q] := 0;
 end;

for i := lrow + 1 to urow - 2 do
 begin
  for j := lcol + 1 to ucol - 2 do
   begin
    k := 0;
   if image^[i - 1, j + 1] <> image^[i, j] then
   begin
   repeat
   k := k + 1;
   until image^[i, j] <> image^[i + k, j + k];
     m^[image^[i, j], k] := m^[image^[i, j], k] + 1;
   end;
  end;
 end;

 sre2 := 0;
 lre2 := 0;
 rpc2 := 0;
 rlnu2 := 0;
 rlnu3 := 0;
 glnu2 := 0;
for p := 1 to 20 do
 begin
  for q := 1 to maxsize do
   begin
     sre2 := sre2 + m^[p, q] / (longint(q) * q);
     lre2 := lre2 + m^[p, q] * longint(q) * q;
    rpc2 := rpc2 + m^[p, q];
  end;
   glnu2 := glnu2 + rpc2 * rpc2;
 end;
for q := 1 to maxsize do
 begin
  for p := 1 to 20 do
   begin
     rlnu2 := rlnu2 + m^[p, q];
```

```
    end;
     rlnu3 := rlnu3 + rlnu2 * rlnu2;
   end;
   sre2 := sre2 / rpc2;
   lre2 := lre2 / rpc2;
   glnu2 := glnu2 / rpc2;
   rlnu2 := rlnu3 / rpc2;
   rpc2 := rpc2 / ((urow - lrow + 1) * (ucol - lcol + 1));

  with Texture do
   begin                    {Puts the results into the texture record for display}
     TextureData[2, 1] := sre2;
     TextureData[2, 2] := lre2;
     TextureData[2, 3] := glnu2;
     TextureData[2, 4] := rlnu2;
     TextureData[2, 5] := rpc2;
   end;

   f1 := 2;

  for i := lrow to urow do
   begin
    for j := lcol to ucol do
       image^[i, j] := trunc(mygetpixel(j, i) / 12.8) + 1;
   end;

  for i := lrow to urow do
   begin
     image^[i, lcol] := 0;
     image^[i, ucol] := 0;
   end;
  for j := lcol to ucol do
   begin
     image^[lrow, j] := 0;
     image^[urow, j] := 0;
   end;

  for p := 1 to 20 do
   begin
    for q := 1 to maxsize do
     m^[p, q] := 0;
   end;

  for i := lrow + 1 to urow - 2 do
   begin
    for j := lcol + 1 to ucol - 2 do
     begin
      k := 0;
     if image^[i - 1, j] <> image^[i, j] then
     begin
     repeat
     k := k + 1;
     until image^[i, j] <> image^[i + k, j];
       m^[image^[i, j], k] := m^[image^[i, j], k] + 1;
     end;
    end;
  end;
```

```
 sre90 := 0;
 lre90 := 0;
 rpc90 := 0;
 rlnu90 := 0;
 rlnu900 := 0;
 glnu90 := 0;
for p := 1 to 20 do
 begin
 for q := 1 to maxsize do
  begin
     sre90 := sre90 + m^[p, q] / (longint(q) * q);
     lre90 := lre90 + m^[p, q] * longint(q) * q;
    rpc90 := rpc90 + m^[p, q];
  end;
   glnu90 := glnu90 + rpc90 * rpc90;
 end;
for q := 1 to maxsize do
 begin
 for p := 1 to 20 do
  begin
    rlnu90 := rlnu90 + m^[p, q];
  end;
   rlnu900 := rlnu900 + rlnu90 * rlnu90;
 end;
 sre90 := sre90 / rpc90;
 lre90 := lre90 / rpc90;
 glnu90 := glnu90 / rpc90;
 rlnu90 := rlnu900 / rpc90;
 rpc90 := rpc90 / ((urow - lrow + 1) * (ucol - lcol + 1));

with Texture do
 begin                          {Puts the results into the texture record for display}
   TextureData[3, 1] := sre90;
   TextureData[3, 2] := lre90;
   TextureData[3, 3] := glnu90;
   TextureData[3, 4] := rlnu90;
   TextureData[3, 5] := rpc90;
 end;

 f1 := 3;

for i := lrow to urow do
 begin
 for j := lcol to ucol do
    image^[i, j] := trunc(mygetpixel(j, i) / 12.8) + 1;
 end;

for i := lrow to urow do
 begin
  image^[i, lcol] := 0;
  image^[i, ucol] := 0;
 end;
for j := lcol to ucol do
 begin
   image^[lrow, j] := 0;
   image^[urow, j] := 0;
 end;
```

```
for p := 1 to 20 do
 begin
  for q := 1 to maxsize do
   m^[p, q] := 0;
 end;

for i := lrow + 1 to urow - 2 do
 begin
  for j := lcol + 1 to ucol - 2 do
   begin
    k := 0;
   if image^[i - 1, j - 1] <> image^[i, j] then
   begin
   repeat
   k := k + 1;
   until image^[i, j] <> image^[i - k, j - k];
     m^[image^[i, j], k] := m^[image^[i, j], k] + 1;
   end;
  end;
 end;

 sre135 := 0;
 lre135 := 0;
 rpc135 := 0;
 rlnu135 := 0;
 rlnu1350 := 0;
 glnu135 := 0;
for p := 1 to 20 do
 begin
  for q := 1 to maxsize do
   begin
      sre135 := sre135 + m^[p, q] / (longint(q) * q);
      lre135 := lre135 + m^[p, q] * longint(q) * q;
     rpc135 := rpc135 + m^[p, q];
   end;
    glnu135 := glnu135 + rpc135 * rpc135;
 end;
for q := 1 to maxsize do
 begin
  for p := 1 to 20 do
   begin
     rlnu135 := rlnu135 + m^[p, q];
   end;
    rlnu1350 := rlnu1350 + rlnu135 * rlnu135;
 end;
 sre135 := sre135 / rpc135;
 lre135 := lre135 / rpc135;
 glnu135 := glnu135 / rpc135;
 rlnu135 := rlnu1350 / rpc135;
 rpc135 := rpc135 / ((urow - lrow + 1) * (ucol - lcol + 1));

with Texture do
 begin                    {Puts the results into the texture record for display}
    TextureData[4, 1] := sre135;
    TextureData[4, 2] := lre135;
    TextureData[4, 3] := glnu135;
    TextureData[4, 4] := rlnu135;
    TextureData[4, 5] := rpc135;
```

```
    end;

    avesre := (sre + sre2 + sre90 + sre135) / 4;      {Finds the mean of the results}
    avelre := (lre + lre2 + lre90 + lre135) / 4;
    aveglnu := (glnu + glnu2 + glnu90 + glnu135) / 4;
    averlnu := (rlnu + rlnu2 + rlnu90 + rlnu135) / 4;
    averpc := (rpc + rpc2 + rpc90 + rpc135) / 4;

   with Texture do
    begin                          {Puts the results into the texture record for display}
      TextureData[5, 1] := avesre;
      TextureData[5, 2] := avelre;
      TextureData[5, 3] := aveglnu;
      TextureData[5, 4] := averlnu;
      TextureData[5, 5] := averpc;
    end;

    Measure;
    ShowResults;

     DisposPtr(Ptr(m));
    m := nil;
     DisposPtr(Ptr(image));
    image := nil;
  end;
100:
end;

procedure DoNGLDM;    {NGLDM}
 label
  100;
 const
  lr = 0;
  ur = 255;
  lc = 0;
  uc = 255;
 type
  qtype = array[lr..ur, lc..uc] of longint;
  qptr = ^qtype;
 var
  q: qptr;
  i, j, sn, s, k, dd: longint;
   tot1, tot2, tot3, tot4, tot5, subtotal: real;
  n1, n2, n3, n4, n5, nf, l10: real;
  wasaccepted1, wasaccepted2: boolean;

begin
 with info^.roirect do
  begin
    l10 := ln(10);
    width := right - left;
   height := bottom - top;
    lrow := top;
    urow := bottom - 1;
    lcol := left;
    ucol := right - 1;
  if wasaccepted then
    exit(DoNGLDM);
```

```
 if G1 > width then
  begin
    putmessage(concat('Intersample Spacing D ', long2str(G1), ' is greater than width ',
long2str(width), '. Please try again'));
   exit(DoNGLDM);
  end;
   q := qptr(NewPtr(sizeof(qtype)));
 if q = nil then
  goto 100;

 with Texture do
  begin
   TF := 4;                    {Tells program which table to use - TF4 is the NGLDM table}
  end;

 if ((g1 < (height div 2)) and (g1 < (width div 2))) then
  if ((g1 > 0) and (g1 < 10)) then
   begin
    for i := lr to ur do
     for j := lc to uc do
      q^[i, j] := 0;
    for i := lrow + g1 to urow - g1 do
     for j := lcol + g1 to ucol - g1 do
     begin
      s := 0;
      dd := g1;
      repeat
      if abs(mygetpixel(j, i) - mygetpixel(j + dd, i)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j + dd, i - dd)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j, i - dd)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j - dd, i - dd)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j - dd, i)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j - dd, i + dd)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j, i + dd)) <= a1 then
      s := s + 1;
      if abs(mygetpixel(j, i) - mygetpixel(j + dd, i + dd)) <= a1 then
      s := s + 1;
      dd := dd - 1;
      until (dd < 1);
        q^[mygetpixel(j, i), s] := q^[mygetpixel(j, i), s] + 1;
     end;

     tot1 := 0.0;
     tot2 := 0.0;
     tot3 := 0.0;
     tot4 := 0.0;
     tot5 := 0.0;
     subtotal := 0.0;
     nf := 0.0;

    for k := lr to ur do
    for sn := lc to uc do
```

```
      begin
      if sn <> 0 then
        tot1 := tot1 + q^[k, sn] / (sn * sn);
        tot2 := tot2 + (sn * sn) * q^[k, sn];
        tot4 := tot4 + (q^[k, sn] * q^[k, sn]);
       nf := nf + q^[k, sn];
      end;

     for sn := lc to uc do
      begin
      for k := lr to ur do
      begin
      if (q^[k, sn] <> 0) then
         tot5 := tot5 - ((q^[k, sn] / nf) * ln(q^[k, sn] / nf) / l10);
       subtotal := subtotal + q^[k, sn];
      end;
        tot3 := tot3 + (subtotal * subtotal);
       subtotal := 0.0;
      end;

      n1 := tot1 / nf;
      n2 := tot2 / nf;
      n3 := tot3 / nf;
      n4 := tot4 / nf;
      n5 := tot5 / nf;

     with Texture do
      begin                    {Puts the results into the texture record for display}
        TextureData[1, 1] := n1;
        TextureData[1, 2] := n2;
        TextureData[1, 3] := n3;
        TextureData[1, 4] := n4;
        TextureData[1, 5] := n5;

      Spacing := g1;
      qw := a1;

      end;
     Measure;
     ShowResults;

      DisposPtr(Ptr(q));
     q := nil;

    end;
  end;
100:
end;

procedure RankFilter;          { Filter rank 1..9, rank5=median filter }

 const
  PixelsPerUpdate = 5000;
 var
  i, j, x, y, pix, Mark, NewMark, LinesPerUpdate, LineCount: integer;
  StartTicks, PixelsRemoved: longint;
  k: char;
  pt: point;
```

```
  MaskRect, frame, trect: rect;
  line: linetype;
  position, leftof: integer;
  hist: array[0..255] of integer;
  buffer: array[0..1] of LineType; { buffer filtered rows so they can be }
                    { put back into original image to save memory }
  bp: byte;    { buffer pointer }
  first, AutoSelectAll, UseMask: boolean; { flag to stop row being modified until it isn't}
            { needed for processing subsequent rows }

begin
 if NotinBounds then
   exit(RankFilter);
 AutoSelectAll := not Info^.RoiShowing;
 if AutoSelectAll then
  with info^ do
   begin
     SelectAll(false);
     SetPort(wptr);
    PenNormal;
     PenPat(pat[PatIndex]);
     FrameRect(wrect);
  end;
 if TooWide then
   exit(RankFilter);
 ShowWatch;
 if info^.RoiType <> RectRoi then
  UseMask := SetupMask
 else
  UseMask := false;
 frame := info^.RoiRect;
 StartTicks := TickCount;
 with frame, info^ do
  begin
   changes := true;
   RoiShowing := false;
   if left > 0 then
     left := left - 1;
   if right < PicRect.right then
     right := right + 1;
    width := right - left;
   height := bottom - top;
   LinesPerUpdate := PixelsPerUpdate div width;
   LinesPerUpdate := LinesPerUpdate div 3;
   Mark := RoiRect.top;
   LineCount := 0;
   if wasaccepted then
     exit(RankFilter);
   if (Rank < 1) or (Rank > 9) then
    begin
     putmessage(concat('Rank has to be between 1 and 9 inclusive'));
      exit(RankFilter);
    end;
    getline(left, top, width, line);
   for i := left to right - 1 do
     line[i] := 0;                                    {Makes top row white}
    putline(left, top, width, line);
    getline(left, bottom - 1, width, line);
```

```
for i := left to right - 1 do
  line[i] := 0;                                  {Makes bottom row white}
 putline(left, bottom - 1, width, line);
 getcolumn(left, top, height, line);
for i := top to bottom - 1 do
  line[i] := 0;                                  {Makes left column white}
 putcolumn(left, top, height, line);
 getcolumn(right - 1, top, height, line);
for i := top to bottom - 1 do
  line[i] := 0;                                  {Makes right column white}
 putcolumn(right - 1, top, height, line);
for x := 0 to 1 do
 for y := 0 to 1024 do
 begin
    buffer[x, y] := 0;                           {Clears buffer}
 end;
 bp := 0;   { initialise buffer pointer }
 First := true;
for i := top + 1 to bottom - 1 do
 begin { all rows }
   getline(left, i, width, line);   { get original }
   buffer[bp] := line;
 for x := 0 to 255 do  { empty histogram }
    hist[x] := 0;
  Position := 128;  { guess middle }
  Leftof := 0;
 for y := left to left + 1 do    { first 2 cols }
  for x := i - 1 to i + 1 do
  begin   { 3 rows centred on i }
     pix := mygetpixel(y, x);
     hist[pix] := hist[pix] + 1;
  if pix < position then
    leftof := leftof + 1;
  end; { for x }
 for j := left + 1 to right - 2 do
 begin  { rest of columns }
  for x := i - 1 to i + 1 do
  begin { 3 rows, load new column }
     pix := mygetpixel(j + 1, x);
     hist[pix] := hist[pix] + 1;
  if pix < position then
    leftof := leftof + 1;
  end;
  repeat          { search up }
     leftof := leftof + hist[position];
   position := position + 1;
  until leftof >= Rank;
  repeat          { search down }
   position := position - 1;
     leftof := leftof - hist[position];
  until leftof < Rank;
     buffer[bp, j] := unsignedbyte(Position);
  for x := i - 1 to i + 1 do
  begin  { 3 rows, unload old column }
     pix := mygetpixel(j - 1, x);
     hist[pix] := hist[pix] - 1;
  If pix < Position then
    leftof := leftof - 1;
```

```
      end;
      end; { for j}
      bp := bp + 1;
      bp := bp mod 2;  {2 element 'ring' }
      if i - 1 <> 0 then
      begin
      if UseMask then
         PutLineUsingMask(left, i - 1, width, buffer[bp])
      else
         PutLine(left, i - 1, width, buffer[bp]);
      end;
      LineCount := LineCount + 1;
      if LineCount = LinesPerUpdate then
      begin
        pt.h := RoiRect.left;
        pt.v := i + 1;
        NewMark := pt.v;
      with RoiRect do
         SetRect(MaskRect, left, mark, right, NewMark);
       UpdateScreen(MaskRect);
       LineCount := 0;
        Mark := NewMark;
      if magnification > 1.0 then
        Mark := Mark - 1;
      if CommandPeriod then
      begin
      UpdatePicWindow;
      beep;
       PixelsRemoved := 0;
      if AutoSelectAll then
       KillRoi;
        exit(RankFilter);
      end;
      end;
    end; { for i }
    trect := frame;
    InsetRect(trect, 1, 1);
    ShowTime(StartTicks, trect, '');
  end; {with}
 if LineCount > 0 then
 begin
  with frame do
     SetRect(MaskRect, left, mark, right, bottom);
   UpdateScreen(MaskRect);
 end;
 if AutoSelectAll then
  KillRoi;
end;

{------------------------------  RangeFilter  ------------------------------}

procedure RangeFilter;        { Filter ranks 1..9}

 const
  PixelsPerUpdate = 5000;
 var
  i, j, x, y, pix, Mark, NewMark, LinesPerUpdate, LineCount: integer;
  StartTicks, PixelsRemoved: longint;
```

```
  k: char;
  pt: point;
   MaskRect, frame, trect: rect;
  line: linetype;
   position_h, position_l, leftof_h, leftof_l: integer;
  hist: array[0..255] of integer;
  buffer: array[0..1] of LineType; { buffer filtered rows so they can be }
                        { put back into original image to save memory }
  bp: byte;    { buffer pointer }
  first, AutoSelectAll, UseMask: boolean; { flag to stop row being modified until it isn't}
              { needed for processing subsequent rows }

begin
 if NotinBounds then
   exit(RangeFilter);
 AutoSelectAll := not Info^.RoiShowing;
 if AutoSelectAll then
 with info^ do
  begin
     SelectAll(false);
     SetPort(wptr);
    PenNormal;
     PenPat(pat[PatIndex]);
     FrameRect(wrect);
  end;
 if TooWide then
   exit(RangeFilter);
 ShowWatch;
 if info^.RoiType <> RectRoi then
  UseMask := SetupMask
 else
  UseMask := false;
 frame := info^.RoiRect;
 StartTicks := TickCount;
 with frame, info^ do
  begin
   changes := true;
   RoiShowing := false;
   if left > 0 then
     left := left - 1;
   if right < PicRect.right then
     right := right + 1;
    width := right - left;
   height := bottom - top;
   LinesPerUpdate := PixelsPerUpdate div width;
   LinesPerUpdate := LinesPerUpdate div 3;
   Mark := RoiRect.top;
   LineCount := 0;
   if wasaccepted then
     exit(RangeFilter);
   if (Rank_h < 1) or (Rank_h > 9) or (Rank_l < 1) or (Rank_l > 9) then
    begin
      putmessage(concat('Ranks have to be between 1 and 9 inclusive'));
       exit(RangeFilter);
       getline(left, top, width, line);
     for i := left to right - 1 do
       line[i] := 0;                                    {Makes top row white}
       putline(left, top, width, line);
```

```
    getline(left, bottom - 1, width, line);
  for i := left to right - 1 do
    line[i] := 0;                              {Makes bottom row white}
    putline(left, bottom - 1, width, line);
    getcolumn(left, top, height, line);
  for i := top to bottom - 1 do
    line[i] := 0;                              {Makes left column white}
    putcolumn(left, top, height, line);
    getcolumn(right - 1, top, height, line);
  for i := top to bottom - 1 do
    line[i] := 0;                              {Makes right column white}
    putcolumn(right - 1, top, height, line);
 end;
for x := 0 to 1 do
 for y := 0 to 1024 do
 begin
    buffer[x, y] := 0;                         {Clears buffer}
 end;
 bp := 0;   { initialise buffer pointer }
 First := true;
for i := top + 1 to bottom - 1 do
 begin { all rows }
   getline(left, i, width, line);   { get original }
   buffer[bp] := line;
 for x := 0 to 255 do  { empty histogram }
    hist[x] := 0;
  Position_h := 128;
  Position_l := 128;   { guess middle }
  Leftof_h := 0;
  Leftof_l := 0;
 for y := left to left + 1 do   { first 2 cols }
  for x := i - 1 to i + 1 do
  begin   { 3 rows centred on i }
     pix := mygetpixel(y, x);
     hist[pix] := hist[pix] + 1;
  if pix < position_h then
    leftof_h := leftof_h + 1;
  if pix < position_l then
    leftof_l := leftof_l + 1;
  end; { for x }
 for j := left + 1 to right - 2 do
 begin  { rest of columns }
  for x := i - 1 to i + 1 do
  begin { 3 rows, load new column }
    pix := mygetpixel(j + 1, x);
    hist[pix] := hist[pix] + 1;
  if pix < position_h then
    leftof_h := leftof_h + 1;
  if pix < position_l then
    leftof_l := leftof_l + 1;
  end;
  repeat          { search up }
    leftof_h := leftof_h + hist[position_h];
   position_h := position_h + 1;
  until leftof_h >= Rank_h;
  repeat          { search down }
   position_h := position_h - 1;
    leftof_h := leftof_h - hist[position_h];
```

```
      until leftof_h < Rank_h;
      repeat          { search up }
        leftof_l := leftof_l + hist[position_l];
       position_l := position_l + 1;
      until leftof_l >= Rank_l;
      repeat          { search down }
       position_l := position_l - 1;
        leftof_l := leftof_l - hist[position_l];
      until leftof_l < Rank_l;
        buffer[bp, j] := unsignedbyte(Position_h - Position_l);
      for x := i - 1 to i + 1 do
      begin  { 3 rows, unload old column }
        pix := mygetpixel(j - 1, x);
        hist[pix] := hist[pix] - 1;
      if pix < Position_h then
       leftof_h := leftof_h - 1;
      if pix < Position_l then
        leftof_l := leftof_l - 1;
      end;
     end; { for j}
     bp := bp + 1;
     bp := bp mod 2;  {2 element 'ring' }
    if i - 1 <> 0 then
     begin
     if UseMask then
       PutLineUsingMask(left, i - 1, width, buffer[bp])
     else
       PutLine(left, i - 1, width, buffer[bp]);
     end;
     LineCount := LineCount + 1;
    if LineCount = LinesPerUpdate then
     begin
       pt.h := RoiRect.left;
       pt.v := i + 1;
       NewMark := pt.v;
     with RoiRect do
       SetRect(MaskRect, left, mark, right, NewMark);
      UpdateScreen(MaskRect);
      LineCount := 0;
       Mark := NewMark;
     if magnification > 1.0 then
       Mark := Mark - 1;
     if CommandPeriod then
     begin
     UpdatePicWindow;
     beep;
      PixelsRemoved := 0;
     if AutoSelectAll then
      KillRoi;
       exit(RangeFilter);
     end;
    end;
   end; { for i }
   trect := frame;
   InsetRect(trect, 1, 1);
    ShowTime(StartTicks, trect, '');
 end; {with}
if LineCount > 0 then
```

```
 begin
  with frame do
    SetRect(MaskRect, left, mark, right, bottom);
   UpdateScreen(MaskRect);
 end;
if AutoSelectAll then
 KillRoi;
end;

procedure Connectivity;{This procedure counts the number of voids per successive MAX
filters}
 var
  black_no: integer;

 procedure Blob;     {This procedure counts the number of voids in an image}
  var
  i, j: INTEGER;          { Variables used for scanning through the image.}
  equiv: INTEGER;         { Used for relabelling equivalent blobs with different labels.}
  b_next: INTEGER;        { Next available black label.}
   max_left: longint;     { Maximum label out of PL and PAL (PML)}
  new_val: INTEGER;       { New label value for a blob when relabelling a line.
              }
   labels: array[0..1000] of integer;
   temp: array[0..1, 0..1000] of integer;

       {-------------------------------------------------------------------------}

 function MIN (x1: longint; x2: longint): longint;{Finds the minimum of x1 and x2}
  var
   minimum: longint;
 begin
  minimum := 10000000;
 if (x1 < minimum) then
   minimum := x1;
 if (x2 < minimum) then
   minimum := x2;
  MIN := minimum;
 end;

 function MAX (x1: longint; x2: longint): longint;{Finds the maximum of x1 and x2}
  var
   maximum: longint;
 begin
  maximum := 0;
 if (x1 > maximum) then
   maximum := x1;
 if (x2 > maximum) then
   maximum := x2;
  MAX := maximum;
 end;

 procedure add_label (      { Labels temp with a new label}
    var level,              { The next available label}
    count: INTEGER;         { The number of blobs thus far}
    c: INTEGER);            { Position in temp being labelled}
 begin
 with info^, info^.picrect do
  begin
```

```
    temp[1, c] := level;
    labels[level] := level;        { Fix label in the merge table}
   count := count + 1;             { New blob so increment count}
    level := level + 1;            { Next available level}
  end;
end;

procedure add_labelzero ( { Labels temp with a new label}
    var level,                     { The next available label}
    count: INTEGER;                { The number of blobs thus far}
    c: INTEGER);                   { Position in temp being labelled}
begin
 with info^, info^.picrect do
  begin
    temp[0, c] := level;
    labels[level] := level;        { Fix label in the merge table}
   count := count + 1;             { New blob so increment count}
    level := level + 1;            { Next available level}
  end;
end;

      {-------------------------------------------------------------------------}
procedure merge (                  { Merge two blobs together}
   var count: INTEGER;             { The number of blobs thus far}
   c: INTEGER;                     { Position in temp being labelled}
    l1, l2: INTEGER);              { The two labels being merged}
 var
   min_v: longint;                 { The labels being merged - sorted}
begin
 count := count - 1;               { Since two blobs are being merged together, decrement the}
                                                          {blob count.}
  min_v := MIN(labels[l1], labels[l2]);        { Sort the two labels to find the minimum and}
                                                          {maximum}
  temp[1, c] := labels[min_v];{Label image with the minimum label.}
  labels[l1] := labels[min_v]; { Fix other labels to that of the minimum.}
  labels[l2] := labels[min_v];
end;

begin
with info^, info^.picrect do
 begin
  black_no := 0;                   { Initialise counters - no blobs have been found yet}
  b_next := 1;                     { Black labels go from 1 to 999.  The first available label is 1.}

 if mygetpixel(left, top) >= 128 then      { Add a label for the first pixel}
   add_labelzero(b_next, black_no, left)
 else
    temp[top, left] := 1000;

 for j := left + 1 to right - 1 do
  begin                                    { Label first row}
   if mygetpixel(j, top) >= 128 then
    if mygetpixel(j - 1, top) >= 128 then
      temp[0, j] := temp[0, j - 1]         { P = BLK, PL = BLK  - use same label}
    else
      add_labelzero(b_next, black_no, j)   { P = BLK, PL = WHT  - new black label}
   else
      temp[0, j] := 1000;
```

```
    end;

   for i := top + 1 to bottom - 1 do          { Count blobs on a row by row basis}
    begin
     if mygetpixel(left, i) >= 128 then       { First pixel in new row}
      if mygetpixel(left, i - 1) >= 128 then
        temp[1, left] := temp[0, left]        { P = BLK, PA = BLK  - use same label}
      else
       add_label(b_next, black_no, left)      { P = BLK, PA = WHT  - new black label}
     else
        temp[1, left] := 1000;

     for j := left + 1 to right - 1 do
     begin
       max_left := MAX(temp[1, j - 1], temp[0, j - 1]);{ Maximum label of PL and PAL (PML).}
                                                       {This is the label}
     if mygetpixel(j, i) >= 128 then
     if mygetpixel(j, i - 1) >= 128 then
     if mygetpixel(j - 1, i) >= 128 then
     if labels[temp[1, j - 1]] = labels[temp[0, j]] then
       temp[1, j] := labels[temp[1, j - 1]]    { P = BLK, PL=PA = BLK  - use same label.}
     else
      merge(black_no, j, temp[1, j - 1], temp[0, j])  { P = BLK, PL<>PA - merge PL and PA.}
     else
       temp[1, j] := labels[temp[0, j]]        { P = BLK, PL = WHT, PA=BLK - use PA label.}
     else if temp[1, j - 1] < 1000 then
       temp[1, j] := labels[temp[1, j - 1]]    { P = BLK, PL = BLK, PA = WHT - use PL label }
     else
      add_label(b_next, black_no, j)   { P = BLK,  PL = WHT, PA = WHT - new black label.}
     else
       temp[1, j] := 1000;
     end;  {for j}
{ Second scan across line to relabel pixels.  Blobs thus far (on this line) are completely
relabelled to reflect merges.  A label is marked as relabelled in the label table by pointing to a
level of the opposite type - ie black blobs are indicated by white in the table.  This is used as a
translation table for relabelling pixels.  The label table is then corrected for the next row
after relabelling is complete.}

     b_next := 1;                        { Reset first available label counters. }
    for j := left to right - 1 do
     begin
     if temp[1, j] <> 1000 then
     if ((temp[1, j] < 1000) and (labels[temp[1, j]] < 0)) then
       temp[1, j] := -labels[temp[1, j]]
     else
     begin
      equiv := temp[1, j];       { Search through mergers in the table until a base label,}
     repeat                              { or one which has been reallocated is found}
      equiv := labels[equiv];
    until (equiv = labels[equiv]) or (labels[equiv] < 0);
{ Base label - minimum label after a merge Label has been reallocated.}
     if equiv = labels[equiv] then
     begin
      new_val := b_next;      { Black base label was found - allocate a new black label}
      b_next := b_next + 1;
     end
     else
       new_val := -labels[equiv];        { Part of blob already relabelled, so use same label}
```

```
        equiv := temp[1, j];              { Hang onto old label for correcting the label table}
        temp[1, j] := new_val;            { Relabel image with the label obtained}
       repeat
        max_left := labels[equiv];
        labels[equiv] := -new_val;        { Set equivalent labels in the table to label obtained.}
        equiv := max_left;
       until ((equiv < 1000) and (labels[equiv] < 0));
       end;
       end; {for j}

      for j := 1 to b_next do             { Fix the labels in the label table for the next line}
        labels[j] := j;

      for j := left to right - 1 do
        temp[0, j] := temp[1, j];
     end;                                 {for i}
     User2^[mCount] := black_no;
   end;
  end;

begin
 Rank := 9;{Sets rank of rank filter to 9 - MAX filter}
 repeat
  Measure;
  Blob;          {Counts the number of voids}
  RankFilter;    {Adds a layer of pixels to voids using a MAX filter}
 until black_no = 1;
 ShowResults;
end;

procedure WeightedArea;
 type
  areatype = array[0..8000] of real;
  areaptr = ^areatype;
 var
  Area, diffarea: areaptr;
  i, j: integer;
begin
 Area := AreaPtr(NewPtr(sizeof(areatype)));
 diffArea := AreaPtr(NewPtr(sizeof(areatype)));
 blOD := true;
 Rank := 1;                           {Sets rank of rank filter to 1 - MIN filter}
 j := 1;
 Area^[0] := 1.0;
 repeat
  Measure;
                                      {Measures the area of the voids in the image}
  Area^[j] := (mean^[j] / 255) * mArea^[j];          {Finds the area}
 if (j <> 1) and (Area^[j - 1] > 0.0) then
  begin
    diffArea^[j - 1] := Area^[j - 1] - Area^[j];
{Finds the area difference due to removed layer of pixels}
    User2^[j - 1] := (j - 1) * diffArea^[j - 1];
   User2^[j] := 0.0;                  {Puts results into User2 array for display to results window}
 end;
 If Area^[j] > 0.0 then
  RankFilter;                         {Applies MIN filter}
 j := j + 1;
```

```
 until Area^[j - 1] = 0.0;  {Continue until all the voids have disappeared}
 ShowResults;
  DisposPtr(Ptr(Area));
  DisposPtr(Ptr(diffArea));
end;

procedure DoNeighbour;
{This procedure finds the distances to the four nearest neighbours for each void}
{and calculates the change in effective area due to the ignored voids on the edges}
 type
mintype = array[0..8000] of real;
  minptr = ^mintype;
 var
  minmin: minptr;
  i, j: integer;
  dx, dy, sqrdx, sqrdy, total, min, dist: real;
  disttop, distbottom, distleft, distright, maxmin: real;
  newtop, newbottom, newleft, newright: extended;

 procedure FindMin; {Finds the distance to the nearest boundary for each void}
 begin
  minmin^[i] := 1e10;
 if disttop < minmin^[i] then
   minmin^[i] := disttop;
 if distbottom < minmin^[i] then
   minmin^[i] := distbottom;
 if distleft < minmin^[i] then
   minmin^[i] := distleft;
 if distright < minmin^[i] then
   minmin^[i] := distright;
end;

 procedure FindMax1;{Finds the maximum distance from the nearest boundary}
{at which no first nearest neighbour existed}
{This is used in finding the area change due to the outer voids being ignored}
  var
i: integer;
 begin
  maxmin := 0;
 for i := 1 to mCount do
  begin
   if (User2^[i] = 0) and (minmin^[i] > maxmin) then
     maxmin := minmin^[i];
  end;
 end;

 procedure FindMax2;{Finds the maximum distance from the nearest boundary}
{at which no second nearest neighbour existed}
{This is used in finding the area change due to the outer voids being ignored}
  var
   i: integer;
 begin
  maxmin := 0;
 for i := 1 to mCount do
  begin
   if (User3^[i] = 0) and (minmin^[i] > maxmin) then
     maxmin := minmin^[i];
  end;
```

```
end;

procedure FindMax3;{Finds the maximum distance from the nearest boundary}
{at which no third nearest neighbour existed}
{This is used in finding the area change due to the outer voids being ignored}
 var
i: integer;
begin
 maxmin := 0;
for i := 1 to mCount do
 begin
  if (User4^[i] = 0) and (minmin^[i] > maxmin) then
    maxmin := minmin^[i];
 end;
end;

procedure FindMax4;{Finds the maximum distance from the nearest boundary}
{at which no fourth nearest neighbour existed}
{This is used in finding the area change due to the outer voids being ignored}
 var
i: integer;
begin
 maxmin := 0;
for i := 1 to mCount do
 begin
  if (User5^[i] = 0) and (minmin^[i] > maxmin) then
    maxmin := minmin^[i];
 end;
end;

begin
 minmin := minptr(NewPtr(sizeof(mintype)));
with info^, info^.picrect do
 begin
  AnalyzeParticles;
       {Finds the X-Y positions of all the voids in the image}
 for i := 1 to mCount do
  begin
   min := 1e10;
  if SpatiallyCalibrated then
   begin     {finds the distances all the boundaries if a scale has been set}
     newtop := top / xSpatialScale;
     newbottom := bottom / xSpatialScale;
     newleft := left / ySpatialScale;
     newright := right / ySpatialScale;
     disttop := abs(ycenter^[i] - newtop);
     distbottom := abs(newbottom - ycenter^[i]);
     distleft := abs(xcenter^[i] - newleft);
     distright := abs(newright - xcenter^[i]);
   end;
  if not SpatiallyCalibrated then
   begin     {finds the distances all the boundaries if a scale has not been set}
     disttop := abs(ycenter^[i] - top);
     distbottom := abs(bottom - ycenter^[i]);
     distleft := abs(xcenter^[i] - left);
     distright := abs(right - xcenter^[i]);
   end;
  FindMin;
```

```
    for j := 1 to mCount do
     begin                              {Finding the first nearest neighbour distances}
      if i <> j then
      begin
        dx := xcenter^[j] - xcenter^[i];
        dy := ycenter^[j] - ycenter^[i];
       sqrdx := dx * dx;
       sqrdy := dy * dy;
       total := sqrdx + sqrdy;
        dist := sqrt(total);
      if dist <= min then
       min := dist
      end;  {for i<>j}
     end; {for j}
    if min > minmin^[i] then
     min := 0;
     User2^[i] := min;
   end; {for i}
   FindMax1;
  for i := 1 to mCount do
   begin
    if minmin^[i] < maxmin then
      User2^[i] := 0;
{Sets first nearest neighbour distance to 0 if closer to boundary than nearest void}
{Also puts the first nearest neighbour distances into the User2 array for display}
   end;
  if SpatiallyCalibrated then
{Calculates area change due to outer voids being ignored for first neighbour level}
              {and stores in mArea array for display}
    mArea^[1] := (abs(newtop - newbottom) - (2 * maxmin)) * (abs(newright - newleft) - (2
*  maxmin));
  if not SpatiallyCalibrated then
    mArea^[1] := (abs(top - bottom) - (2 * maxmin)) * (abs(right - left) - (2 * maxmin));
  for i := 1 to mCount do
   begin
    min := 1e10;
   if SpatiallyCalibrated then
    begin
      newtop := top / xSpatialScale;
      newbottom := bottom / xSpatialScale;
      newleft := left / ySpatialScale;
      newright := right / ySpatialScale;
      disttop := abs(ycenter^[i] - newtop);
      distbottom := abs(newbottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - newleft);
      distright := abs(newright - xcenter^[i]);
    end;
   if not SpatiallyCalibrated then
    begin
      disttop := abs(ycenter^[i] - top);
      distbottom := abs(bottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - left);
      distright := abs(right - xcenter^[i]);
    end;
   for j := 1 to mCount do
    begin                               {Finding the second nearest neighbour distances}
     if i <> j then
     begin
```

```
        dx := xcenter^[j] - xcenter^[i];
        dy := ycenter^[j] - ycenter^[i];
       sqrdx := dx * dx;
       sqrdy := dy * dy;
       total := sqrdx + sqrdy;
        dist := sqrt(total);
     if (dist > User2^[i]) and (dist <= min) then
      min := dist
     end;  {for i<>j}
    end; {for j}
   if min > minmin^[i] then
    min := 0;
   if User2^[i] = 0 then
{if first nearest neighbour level does not exist then the second cannot}
     min := 0;
     User3^[i] := min;
   end; {for i}
   FindMax2;
  for i := 1 to mCount do
   begin
   if minmin^[i] < maxmin then
     User3^[i] := 0;
{Sets second nearest neighbour distance to 0 if closer to boundary than the second nearest}
{void. Also puts the second nearest neighbour distances into the User3 array for display}
   end;
  if SpatiallyCalibrated then
{Calculates area change due to outer voids being ignored for second neighbour level}
                    {and stores in mArea array for display}
    mArea^[2] := (abs(newtop - newbottom) - (2 * maxmin)) * (abs(newright - newleft) - (2
* maxmin));
  if not SpatiallyCalibrated then
    mArea^[2] := (abs(top - bottom) - (2 * maxmin)) * (abs(right - left) - (2 * maxmin));
  for i := 1 to mCount do
   begin
    min := 1e10;
   if SpatiallyCalibrated then
    begin
      newtop := top / xSpatialScale;
      newbottom := bottom / xSpatialScale;
      newleft := left / ySpatialScale;
      newright := right / ySpatialScale;
      disttop := abs(ycenter^[i] - newtop);
      distbottom := abs(newbottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - newleft);
      distright := abs(newright - xcenter^[i]);
    end;
   if not SpatiallyCalibrated then
    begin
      disttop := abs(ycenter^[i] - top);
      distbottom := abs(bottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - left);
      distright := abs(right - xcenter^[i]);
    end;
   for j := 1 to mCount do
    begin                          {Finding the third nearest neighbour distances}
     if i <> j then
     begin
       dx := xcenter^[j] - xcenter^[i];
```

```
        dy := ycenter^[j] - ycenter^[i];
        sqrdx := dx * dx;
        sqrdy := dy * dy;
        total := sqrdx + sqrdy;
        dist := sqrt(total);
      if (dist > User3^[i]) and (dist <= min) then
       min := dist
      end;  {for i<>j}
     end; {for j}
    if min > minmin^[i] then
     min := 0;
    if User3^[i] = 0 then
{if second nearest neighbour level does not exist then the third cannot}
     min := 0;
     User4^[i] := min;
   end; {for i}
   FindMax3;
  for i := 1 to mCount do
   begin
   if minmin^[i] < maxmin then
     User4^[i] := 0;
{Sets third nearest neighbour distance to 0 if closer to boundary than the third nearest void}
{Also puts the third nearest neighbour distances into the User4 array for display}
   end;
  if SpatiallyCalibrated then
        {Calculates area change due to outer voids being ignored for third neighbour level}
                              {and stores in mArea array for display}
    mArea^[3] := (abs(newtop - newbottom) - (2 * maxmin)) * (abs(newright - newleft) - (2
* maxmin));
  if not SpatiallyCalibrated then
    mArea^[3] := (abs(top - bottom) - (2 * maxmin)) * (abs(right - left) - (2 * maxmin));
  for i := 1 to mCount do
   begin
    min := 1e10;
   if SpatiallyCalibrated then
    begin
      newtop := top / xSpatialScale;
      newbottom := bottom / xSpatialScale;
      newleft := left / ySpatialScale;
      newright := right / ySpatialScale;
      disttop := abs(ycenter^[i] - newtop);
      distbottom := abs(newbottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - newleft);
      distright := abs(newright - xcenter^[i]);
    end;
   if not SpatiallyCalibrated then
    begin
      disttop := abs(ycenter^[i] - top);
      distbottom := abs(bottom - ycenter^[i]);
      distleft := abs(xcenter^[i] - left);
      distright := abs(right - xcenter^[i]);
    end;
   for j := 1 to mCount do
    begin                    {Finding the fourth nearest neighbour distances}
     if i <> j then
     begin
      dx := xcenter^[j] - xcenter^[i];
      dy := ycenter^[j] - ycenter^[i];
```

```
        sqrdx := dx * dx;
        sqrdy := dy * dy;
        total := sqrdx + sqrdy;
        dist := sqrt(total);
      if (dist > User4^[i]) and (dist <= min) then
       min := dist
      end;  {for i<>j}
     end; {for j}
    if min > minmin^[i] then
     min := 0;
     if User4^[i] = 0 then
{if third nearest neighbour level does not exist then the fourth cannot}
     min := 0;
     User5^[i] := min;
   end; {for i}
   FindMax4;
  for i := 1 to mCount do
   begin
    if minmin^[i] < maxmin then
      User5^[i] := 0;
{Sets fourth nearest neighbour distance to 0 if closer to boundary than the fourth nearest
void. Also puts the fourth nearest neighbour distances into the User5 array for display}
   end;
  if SpatiallyCalibrated then
        {Calculates area change due to outer voids being ignored for fourth neighbour level}
                    {and stores in mArea array for display}
    mArea^[4] := (abs(newtop - newbottom) - (2 * maxmin)) * (abs(newright - newleft) - (2
* maxmin));
  if not SpatiallyCalibrated then
    mArea^[4] := (abs(top - bottom) - (2 * maxmin)) * (abs(right - left) - (2 * maxmin));
  ShowResults;
 end;
 DisposPtr(Ptr(minmin));
end;

procedure DoAreaFactor;
{This procedure finds the area factor for each void in the image}
 var
  i, j: integer;
  dx, dy, sqrdx, sqrdy, total, sum, dist, areafactor: real;
begin
 AnalyzeParticles;     {Finds the area and the X-Y position for every void in the image}
for i := 1 to mCount do
 begin
  sum := 0.0;
  for j := 1 to mCount do
   begin
     dx := xcenter^[j] - xcenter^[i];
     dy := ycenter^[j] - ycenter^[i];
    sqrdx := dx * dx;
    sqrdy := dy * dy;
    total := sqrdx + sqrdy;
     dist := sqrt(total);
   if dist <> 0 then
    begin
      areafactor := (mArea^[j] * 100) / total;
     sum := sum + areafactor;
    end;
```

```
    end;
    User2^[i] := sum;    {Puts the area factor for each void into the User2 array for display}
  end;
  ShowResults;
end;

procedure DoDistanceFactor;
{This procedure finds the distance factor for each void in the image}
 var
  i, j: integer;
  dx, dy, sqrdx, sqrdy, total, sum, dist, distfactor: real;
begin
 AnalyzeParticles;    {Finds the X-Y position for every void in the image}
for i := 1 to mCount do
 begin
  sum := 0.0;
 for j := 1 to mCount do
  begin
     dx := xcenter^[j] - xcenter^[i];
     dy := ycenter^[j] - ycenter^[i];
    sqrdx := dx * dx;
    sqrdy := dy * dy;
    total := sqrdx + sqrdy;
     dist := sqrt(total);
   if dist <> 0 then
    begin
       distfactor := 100000 / total;
      sum := sum + distfactor;
    end;
  end;
   User2^[i] := sum;    {Puts the distance factor for each void into the User2 array for
display}
 end;
 ShowResults;
end;

procedure Circularity;{Finds the circularity of every void in the image}
 var
  i, j: integer;
  circ: real;
begin
 AnalyzeParticles;    {Finds the major and minor elliptical axes for every void in the image}
for i := 1 to mCount do
 begin
    circ := MinorAxis^[i] / MajorAxis^[i];
   User2^[i] := circ;
 end;
 ShowResults;
end;
```

APPENDIX C

Table of Critical Values in the Kolgomorov-Smirnov Two-Sample Test

(Large Samples: two-tailed test)

Level of Significance (%)	Critical Value
90.0	$1.22\sqrt{\frac{n_1+n_2}{n_1 n_2}}$
95.0	$1.36\sqrt{\frac{n_1+n_2}{n_1 n_2}}$
97.5	$1.48\sqrt{\frac{n_1+n_2}{n_1 n_2}}$
99.0	$1.63\sqrt{\frac{n_1+n_2}{n_1 n_2}}$
99.5	$1.73\sqrt{\frac{n_1+n_2}{n_1 n_2}}$
99.9	$1.95\sqrt{\frac{n_1+n_2}{n_1 n_2}}$

where n_1 and n_2 are the two sample sizes being tested.

(Source: Adapted from S. Siegel. *Nonparametric Statistics for the Behavioural Sciences.* McGraw-Hill, New York, 1956. 279p.)

APPENDIX D - Texture Equations

SGLDM Features:

(1) Energy

$$E = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}[f(i,j,d,a)]^2$$

(2) Entropy

$$ENT = - \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}[f(i,j,d,a).\log_{10}(f(i,j,d,a))]$$

(3) Inertia

$$INERTIA = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}[(i-j)^2.f(i,j,d,a)]$$

(4) Homogeneity

$$HOMOG. = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}[f(i,j,d,a)/(1+(i-j)^2)]$$

(5) Correlation

$$COR. = \sum_{i=0}^{Ng-1}\sum_{j=0}^{Ng-1}[(i-\mu x)(j-\mu y).f(i,j,d,a)/(\sigma x.\sigma y)]$$

Where

$$\mu x = \sum_{i=0}^{Ng-1} i \sum_{j=0}^{Ng-1}[f(i,j,d,a)]$$

$$\mu y = \sum_{j=0}^{Ng-1} j \sum_{i=0}^{Ng-1}[f(i,j,d,a)]$$

$$\sigma x = \sum_{i=0}^{Ng-1} (i - \mu x)^2 \sum_{j=0}^{Ng-1} [f(i,j,d,a)]$$

$$\sigma y = \sum_{j=0}^{Ng-1} (j - \mu y)^2 \sum_{i=0}^{Ng-1} [f(i,j,d,a)]$$

GLDM Features:

(1) Contrast

$$CON = \sum_{i=0}^{Ng-1} [i^2 P'(i)]$$

(2) Angular Second Moment

$$ASM = \sum_{i=0}^{Ng-1} [P'(i)]^2$$

(3) Entropy

$$ENT = - \sum_{i=0}^{Ng-1} [P'(i).\log_{10}(P'(i))]$$

(4) Mean

$$MEAN = \left(\frac{1}{Ng}\right) \sum_{i=0}^{Ng-1} [i.P'(i)]$$

(5) Inverse Different Moment

$$IDM = \sum_{i=0}^{Ng-1} [P'(i) / (i^2 + 1)]$$

GLRLM Features:

(1) Long Run Emphasis

$$LRE = \sum_{i=0}^{Ng-1} \sum_{j=1}^{Nr} [j^2 P(i,j)] / \sum_{i=0}^{Ng-1} \sum_{j=1}^{Nr} P(i,j)$$

(2) Short Run Emphasis

$$SRE = \sum_{i=0}^{Ng-1}\sum_{j=1}^{Nr}[P(i,j)/j^2]/\sum_{i=0}^{Ng-1}\sum_{j=1}^{Nr}P(i,j)$$

(3) Gray Level Nonuniformity

$$GLNU = \sum_{i=0}^{Ng-1}\left[\sum_{j=1}^{Nr}P(i,j)\right]^2/\sum_{i=0}^{Ng-1}\sum_{j=1}^{Nr}P(i,j)$$

(4) Run Length Nonuniformity

$$RLNU = \sum_{j=1}^{Nr}\left[\sum_{i=0}^{Ng-1}P(i,j)\right]^2/\sum_{i=0}^{Ng-1}\sum_{j=1}^{Nr}P(i,j)$$

(5) Run Percentage

$$RPC = \sum_{i=0}^{Ng-1}\sum_{j=1}^{Nr}P(i,j)/P''$$

Where P'' is the Number of pixels in the image.

NGLDM Features:

(1) Small Number Emphasis

$$SNE = \sum_{k=0}^{K}\sum_{s=0}^{S}[Q(k,s)/s^2]/R$$

Where R is the normalising factor

(2) Large Number Emphasis

$$LNE = \sum_{k=0}^{K}\sum_{s=0}^{S}[s^2Q(k,s)]/R$$

Where R is the normalising factor

(3) Number Nonuniformity

$$NNU = \sum_{s=0}^{S}\left[\sum_{k=0}^{K} Q(k,s)\right]^2 / R$$

Where R is the normalising factor

(4) Second Moment

$$SM = \sum_{k=0}^{K}\sum_{s=0}^{S}[Q(k,s)]^2 / R$$

Where R is the normalising factor

(5) Entropy

$$ENT = - \sum_{k=0}^{K}\sum_{s=0}^{S}[Q(k,s).\log_{10}(Q(k,s))] / R$$

Where R is the normalising factor

APPENDIX E - Comalco Contracts

Anode Macroscopy Image Analysis Research Proposal

Comalco Aluminium Limited have developed an optical macroscopy technique for assessing the pitching level and degree of compaction in baked anodes. The technique currently requires a subjective interpretation of the anode macrostructure.

Massey University are experienced in image processing. It should be possible to use this experience to develop an automatic qualitative analysis of anodes based on the macroscopy technique. It is, therefore proposed that Massey University develop image analysis algorithms which will indicate the level of porosity in anode samples or photographs of samples.

The final software should be able to detect:

- the number of pores in a sample and their position (eg in a coke grain, in the binder-fines matrix)
- the shapes of pores and cracks in the sample
- the total porosity of the surface
- an unusual distribution of pores in the anode and their position

This information should lead to the development of an expert system (either at Massey University or at CRC) which will indicate the presence of:

- incorrect pitching levels
- compaction faults
- baking cracks
- coke porosity
- macropores
- delamination cracks

CAL will supply to the university, macrographs of common anode faults and of laboratory and plant samples produced under a range of conditions. A list of features to be detected will accompany each macrograph.

At the completion of the project (June 1993), Massey University will supply to CAL, subject to the standard terms and conditions, a disk containing the working software

package written in C including a documented copy of source code, a user's manual and any training required to run the software. Knowledge of C should not be necessary to run the package. The University should also provide a progress report six months after the project commencement and a copy of the student thesis at the completion of the project.

Restatement of the Aim/Methodology of the Image Analysis Project

As requested, this letter restates what CRC expect from the Massey macroscopy project and contains CRC's suggestions on how to achieve it.

Deliverables

CRC expects from Massey the deliverables outlined in the original Research Proposal, which were:

1. A software package containing prototype image analysis / sample preparation techniques to describe anode macrostructure (4 to 5 are suggested)

2. Verification that these techniques can detect pore distribution, association of pores, the location of pores (inside coke grains or in the binder matrix), cracks in the matrix and the aspect ratios of these cracks.

Comalco (possibly in conjunction with Massey) would then evaluate these techniques at its smelters, to detect the anode structures listed in the proposal.

It is suggested that you first aim to assess the anode binder-fines matrix so that pitching level and compaction efficiency can be determined.

Other aims such as:

- measuring coke/butt particle porosity so that the effect of raw coke quality (especially porosity) on pitching level can be monitored, and
- detecting delamination cracking and large bake cracks

are more difficult to meet as large samples are required to obtain a representative measurement. I would consider attempting these goals after techniques to assess the binder-fines matrix have been developed and tested.

Methodology

A four step process is suggested:

1. Develop image analysis routines and sample preparation techniques which recognise anode macrostructures which Comalco uses to assess anode quality (pitching level etc). These structures have been thoroughly discussed in a CRC

report (TM/86/89) written by Barry Sadler. Don Bailey summed up well the type of structures which should be detected in his NZAS trip report (17-18 November 1992).

Match the samples which you already have and which CRC/NZAS have ranked with the macrostructures in Barry's report. Then work from these samples.

CRC would not, however, put a great deal of effort into trying to equate all these samples and all the surface of the samples with the ranking because:

- the ranking is subjective, not a robust analytical procedure
- even three or four photographs will not cover a 50mm diameter sample (20 or 30 would be required at a 10x magnification)

The progress reports indicate promising algorithms and sample preparation techniques which CRC recommend evaluating before developing more.

2. Evaluate the potential of the image analysis techniques by assessing the 20mm x 50mm diameter NZAS compaction samples which you have. These samples have been formed under a range of process parameters which effected the physical properties of the material. Strong correlations with:

- forming compaction force (high, low, normal)
- forming time (25s, 45s, 75s) and
- forming temperature (may effect pitch penetration, pore distribution)

would indicate promising techniques. In this way subjective judgements would be avoided (I have included a history of these samples in case you do not have all the information).

As you have pointed out the aim of the project is not to develop analysis techniques which solely replicate other physical tests. It is expected, however, that the binder-fines matrix structure would be sensitive to some of these process parameters and the usefulness of the project would be proven immediately if the techniques:

- were more sensitive to process parameters than the physical tests
- detected differences in the binder-fines matrix (pore size distribution, cracking etc) which could be attributed to the forming conditions
- explain differences which the physical tests could not (eg variations between mixer rounds)

The techniques could then be used to evaluate anodes of different pitching levels etc, for which the tests give ambiguous results.

I take your point, that a 20mm slice of a 50mm diameter core may not characterise an anode! especially if a large butt particle is present, however, I make the following comments:

- These small samples have been shown to correlate with process conditions
- the principle aim of the project is to be able to differentiate between large coke/butt particles and the binder-fines matrix and assess the matrix
- the sample size we can routinely take out of the plant anodes is restricted

3. Assess larger samples with a range of pitch levels. CRC can provide you with larger samples, rectangles 100mm by ≈200mm. These samples are 100mm diameter cores (sliced in half) from the BSL plant in Queensland formed with a variety of pitching levels (measured by slump of the anode). Please let me know if you require them.

   CRC recommend that a lower magnification image analysis technique should not be evaluated solely because more of the sample can be assessed by eye and used to evaluate the technique. This would not avoid the problem of subjectivity and anode structure may be overlooked. Altering the magnification should be based on increasing the resolution between samples and test the robustness in steps 1 and 2.

4. Provide Comalco with software which incorporates the more useful techniques and allows the user to calibrate them by altering settings (eg gray level, number of associated pixels averaged etc).

Conclusion

I hope these comments are of assistance. I do not believe that Comalco can provide you with samples ranked accurately enough by visual inspection to calibrate your image analysis techniques, that is why we are trying to automate macrostructure analysis! Macrostructure analysis, like each of our physical tests, can only be evaluated by assessing its sensitivity to process changes.

APPENDIX F - Sample Descriptions

SAMPLE ID	Description
12M	Slightly overpitched, but good compaction. Larger pores are mostly inter-particle porosity
4M	Overpitched anode. Small to medium pores form "rings" around aggregate particles to allow excess pitch volatiles to escape during baking
3110	A well pitched and formed anode. Consistent structure throughout the binder/fines matrix of an evenly distributed fine porosity that is typical of an acceptable anode structure
748	Underpitched anode. Medium to large pores scattered throughout binder/fines matrix. No "rings" around aggregate particles also indicate underpitching