Digital image processing

While analyzing foils samples which were damaged or partially destroyed by ultrasonic cavitation it was evident that an accurate quantifiably means of analyzing the foils samples was needed. Essentially what was required was a value representing the area of the foil that was destroyed or damaged. The one of the best methods of doing this would be a digital image processor, as calculating these areas manually would be extremely time-consuming and error prone.

It was decided to use matlab's digital image processing toolbox to calculate these values as the language is easy to master and there are examples and help with similar problems on the Internet. Although these programs are very powerful it simplifies the programming to give the image processor data that can be easily manipulate, and for this reason the foils samples were placed with their dull side facing the scanner and on a black background. This black background was selected to ensure that there was a suitable intensity level difference between the intact foil and areas that had been destroyed. The reason for facing the dull side of the foil up was because the flat sections of the shiny surface in the scanner would tend to come out very white or bright and distorted images intensity level gradients. Although this document gives the code used together with explanations of what each piece of code does a basic knowledge of matlabs structure and syntax is needed for a full understanding. A basic introduction to matlab can be found at
http://space.tin.it/computer/gciabu/engl/matlab/matlab.htm

In order to explain the process of extracting a value for the damaged as well as the destroyed areas as a percentage of the total original foil as completely as possible, an example will be used.

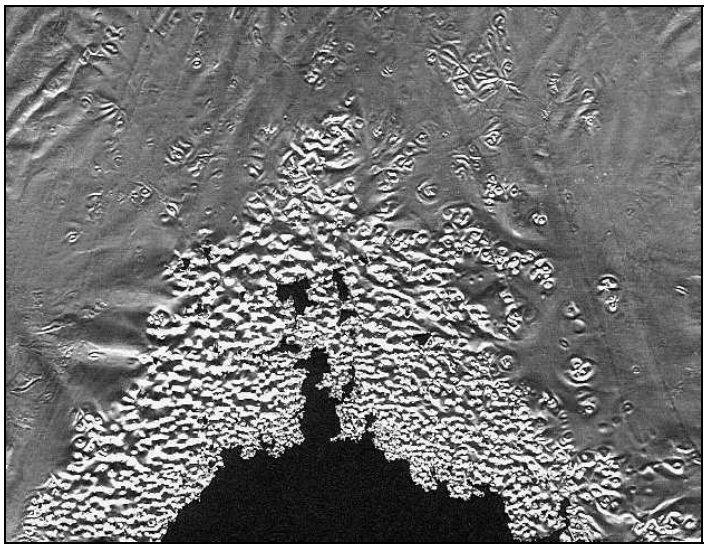

Figure 1 Original normalized image

The above foils sample used in this example was scanned in at 300 dots per inch (DPI) and stored as an 8-bit grayscale JPEG file (The scanning resolution is not crucial only that it need to remain constant for all samples in a specific test but reducing the DPI reduces the processing time). This image file was placed in directory called ‘c:\images’ and the path variable in matlab was up dated to include this path. The image was then read by the matlab program using the ‘imread’ function shown below which converts the image to an m * n matrix in this case called ‘Im’ which represents each pixel as a matrix element between 0 and 255 which can be mathematically manipulated. The ‘size’ function was then used to extract the number of rows and columns in the image matrix and store it in the variables ‘m’ and ‘n‘ for use in later computation.

```
path = (path,’c:\images’);
Im = imread(‘foil.jpg’);
[m,n] = size(Im);
```

The image was then normalized with the ‘imadjust’ function to insure that successive foil samples would yield consistent results. Normalizing is the process whereby a set or matrix of numbers or in this case intensity level's are scaled to insure that they covered the entire range i.e. at least one pixel will be of 0 intensity and at least one pixel of 255 intensity level. (To find out more about a function and what the input variables do type ‘help’ and the function name at the matlab prompt.
eg. ‘help imadjust’)

```
Iadj = imadjust(Im, [], [0 1]);
```

Calculating the damaged area percentage

To calculate the area of damaged foil an edge detection using the ‘edge’ function was performed on the image giving the following image (see Figure 2), however the number of white pixels does not represents the entire area of damaged foil if one compares this to the original image.

```
Iedge = edge(Iadj, 'sobel', (2 * .11));
```

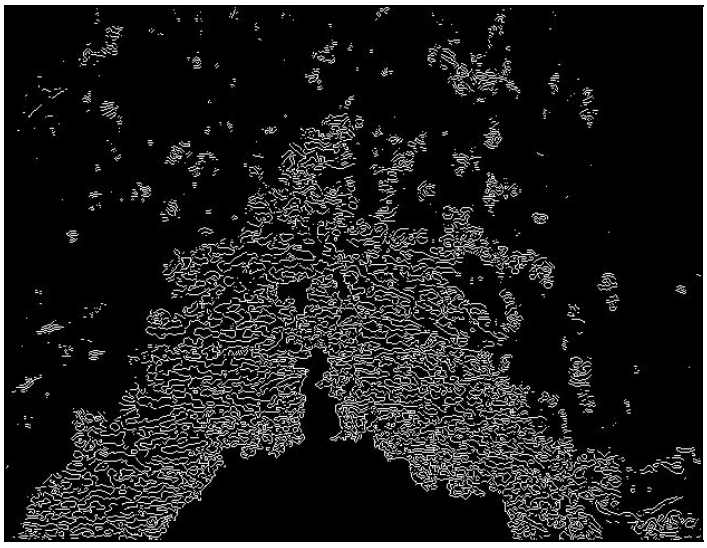

Figure 2 Edge detected image

The solution is to dilate or thicken each of the above lines until they join and form a solid area, however this tended to exaggerate the area and was corrected by simply carrying out the inverse of the above dilate process using the ‘erode’ function resulting in the image in Figure 3. It was found by trial and error that executing the ‘erode’ function twice gave better results however this might not always be the case.

```
SE = ones(6,2);                          % setting up dilate matrix
Iedgedil = dilate(Iedge, SE,'spatial',1);
```

```
seD = ones(3,1);                         % setting up erode matrix
Ifinal = erode(Iedgedil,seD);
Ifinal = erode(Ifinal,seD);
```

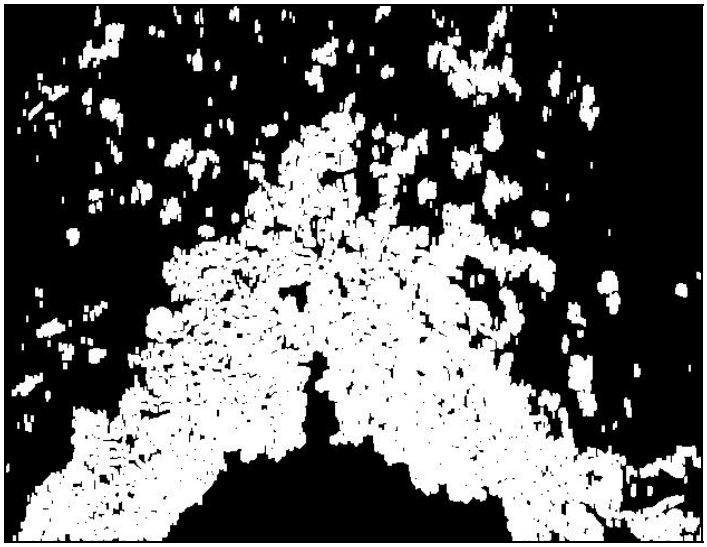

Figure 3 final damaged foil mask image

This gave a very good approximation of the damaged areas of the foil. The number of white pixels summed and divided by the total number of pixels gave a percentage of the area of the foil that was damaged.

```
Dam = 100*(sum((sum(Ifinal))'))/(m*n);
```

In the above example giving a 31.15 % damaged area result.

Calculating the destroyed ***area percentage***

To calculate the area of the destroyed foil, a threshold detection using the 'im2bw' function was carried out on the normalized image i.e. setting intensity level's above the threshold value to 255 and those below it to 0. The threshold value was experimentally set 0.2 to extract only the black background that was visible where the foil was destroyed, however there were a few outlier pixels which were removed with the dilate and erode functions similar to but inverse operation to the above example.

```
Ith = im2bw (Im,0.2);
```

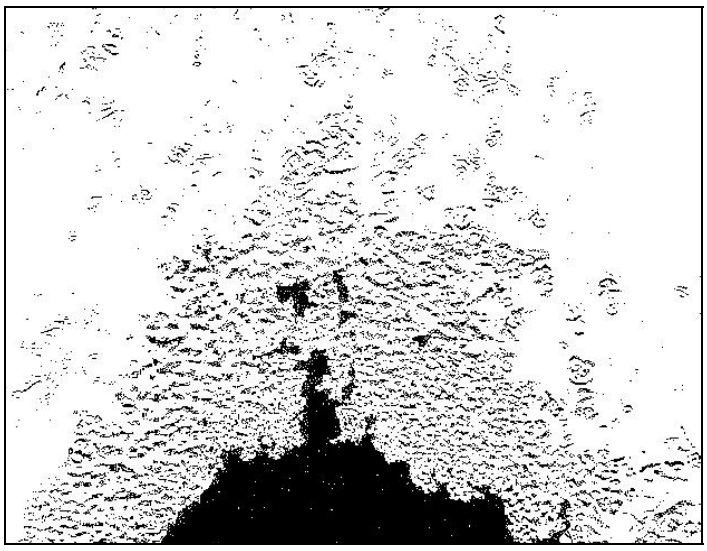

Figure 4 Threshold image

```
seD = ones(3,1);
Itherode = erode(Ith,seD);
Itherode = dilate(Itherode, SE,'spatial',1);
```

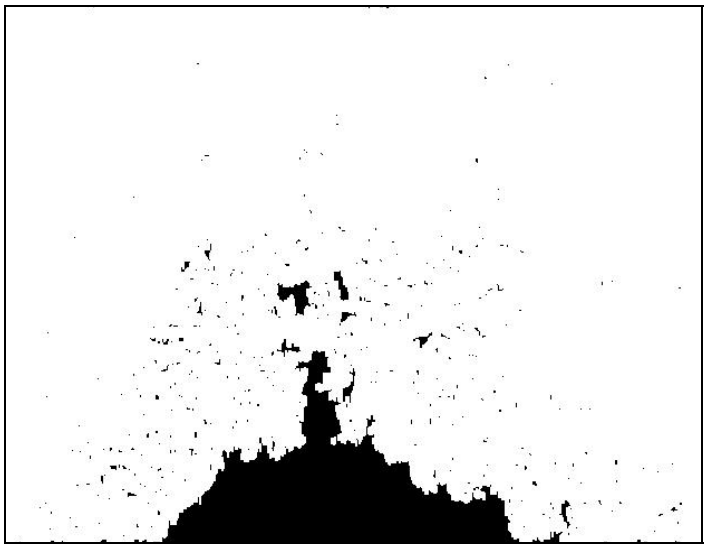

Figure 5 Final destroyed mask image

Once again the pixels were summed, however this time the back pixels were summed and divided by the total number to give a percentage of the area of the foil that was destroyed.

Des = 100*((m*n)-(sum((sum(Itherode))')))/(m*n);

In the above example giving 7.95% destroyed area.

In conclusion it is obvious that this method of quantifying the amounts of damaged and destroyed foil far out perform previous subjective methods of visual inspection and enables a greater degree of accuracy and consistency between samples.

References

www.mathworks.com/products/demos/imagetlbx/examples/morph/morph2.html