Technical University of Crete

Department of Electronics Engineering and Computer Engineering

Letter Detection on Inscriptions via Optical Inspection and Image Processing

Antonis Papanicolaou

Chania, Crete

September 2000

Acknowledgements

I wish to extend my sincerest thanks to my advisor Dr. Michalis Zervakis for his invaluable guidance

To my parents for their love and support

To everyone for their contribution in completing this work

Antonis Papanicolaou

Contents

1. INTRODUCTION 5

1.1. General description 5

1.2. Problem definition 5

1.3. Related research 6

2. ALGORITHMS & TOOLS 8

2.1. Mathematical Morphology 8

2.1.1. Basic Morphological Operations 9

2.2. Edge Detection Algorithms 12

2.2.1. Gradient 12
2.2.2. Regularized Gradient 13
2.2.3. Blur and Minimum Operator 14
2.2.4. Alpha-Trimmed Multidimensional morphological edge detection 16
2.2.5. Histogram-based Morphological Edge Detection 17
2.2.6. Comments on the algorithms 18
2.2.6.1. Blurring 18
2.2.6.2. Structuring Elements 19

2.3. General Algorithms 20

2.3.1. Histogram Normalization 20

2.4. Post Processing Algorithms 21

2.4.1. Contour following 21
2.4.2. Raster tracking 22
2.4.3. Double thresholding 22

3. PROPOSED ALGORITHMS 23

3.1. Pre processing 23

3.2. Edge detection 25

3.2.1 Blurring operation 25
3.2.2. Morphological processing 29

3.3. Post processing algorithms 32

3.3.1. Labeling 32
3.3.2. Cleaning 33
3.3.3. Selective cleaning 33
3.3.4. Edge elimination 34
3.3.5. Edge linking 35

4. PROPOSED PROCEDURE 40

4.1. Preprocessing 42

4.2. Edge detection 42

4.3. Post processing 43

4.3.1. Double thresholding 43
4.3.2. Selective cleaning 45
4.3.3. Edge linking 46
4.3.4. Edge elimination 47

4.4. Other Examples ____48

5. CONCLUSION AND FUTURE WORK______________________________50

6. REFERENCES ____52

1. INTRODUCTION

1.1. General description

Ancient inscriptions are very important for archaeologists and historians because they provide valuable information about the activities and the achievements of human kind. During the last century many excavations all over Greece have brought to light a very large number of inscriptions. Most of these inscriptions are carved on marble or stone. However many of them are heavily corroded by water or by other elements in their long exposure. In attempts to extract useful information from them, archaeologists have found that for several inscriptions when lit from different angles, different letters or drawings can be distinguished. Even worse, there are rocks where it is known that inscriptions exist, but they are completely unreadable.

This thesis develops and studies an automated system that could exploit the various methods that people use to read the inscriptions, such as lighting, in order to provide a helpful and hopefully valuable tool for archaeologists.

1.2. Problem definition

In this thesis our efforts are focused on a first attempt to extract letters from inscriptions. Our goal is not to recognize the characters that are carved, but merely to isolate them from the noisy background in order to make them clearly visible to the human eye. We do not study the electromechanical system, which would consist of lighting and acquisition devices, but we rather focus on image processing tasks. For this reason we use as input images that were taken using a normal commercial digital camera. We consider images that are relatively easy to be read by the human eye, in order to trace the issues that must be considered in the real problem.

Moreover, we focus on the potential of image processing tools rather than pattern recognition or pattern matching tools.
Several problems that were confronted during this work were:

- Marbles or stones are not smooth and express variations on the color of their surface

- Non-uniform illumination degrades the images due to diffusion when acquiring the images

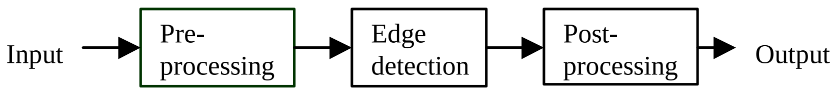
- There exist large cracks on the surface of the stones that interfere with the writings

- Erosion on the surface of the rocks makes the recognition of the carvings quite difficult

Taking under consideration these problems, the following tasks have to be performed towards effective recognition of the carvings:

- Intensity or histogram equalization of the input image
- Edge detection which can discard trivial information
- Post-processing techniques for eliminating the noise and for improving the quality of the edges, such as edge connection algorithms

A schematic diagram of these steps takes has the following form:



In the preprocessing part, we attempt to eliminate the problems we face due to the non-uniform lighting.
The second step is the edge detection phase, where as we will see, the corrosion will create many problems.
Finally we have the post processing part were we extract edges that correspond to letters from the image produced by the edge detector. This latter part will be based only on processing to reveal its potential in such tasks; we do not use pattern recognition methods, such as pattern matching that would be useful for character recognition.

The organization of this work is as follows. Chapter 2 deals with algorithms and methods that are known and have been found in literature. In Chapter 3 our approach and some proposed algorithms will be discussed.

1.3. Related research

While searching for related subjects and work in the literature we came across the following representative papers.

Hwang and Chang [1] have proposed a method for extracting the characters from document with non-uniform background. The method actually removes unwanted background noise from textual images. They have used the wavelet transform and they observed that the magnitude of the intensity variation of character boundaries differs from that of noise at various scales of their wavelet transform. Thus they extract the characters by thesholding.

This work is one of the few algorithms developed to deal with the extraction of characters from a noisy background. However, the background contains only noise due to the scanning procedure and does not involve random noise that can interfere with the characters. Other works that deal mostly with recognition are the following two.

Bourbakis and Goldman [2] have developed an approach for recognizing lines that have unevenness and are not smooth. This is very helpful in tasks such as fingerprint recognition. A graph-based method, using this approach, recognizes characters and fingerprints.

Kim et al. [3] worked on a system for on-line recognition of Korean characters. The tools they used were artificial neural networks and hidden Markov models.

The OCR problem (Optical Character Recognition) has been extensively studied. Nowadays OCR methods have been developed for almost all character sets used worldwide. The two previous papers are headed toward that direction. It is obvious that these are focused on a different direction than our study. Their goal is recognizing line patterns or characters, while we try only to extract the characters from a noisy background.

As a general conclusion, artificial intelligence and pattern recognition methods are mainly used for the character recognition problem, instead of image processing. Methods such as neural networks and fuzzy logic are studied extensively, along with hidden Markov models. This is a logical approach since these tools provide means of recognizing characters, whereas image processing alone cannot be used for recognition or identification.

We emphasize here that all these works, except from Hwang et al., deal with the problem of recognizing hand-written or printed characters on a uniform single-colored background. There is only limited work dealing with the extraction or recognition of characters that are carved on stone or marble, or generally printed on a non-uniform background, which can produce problems such as the ones we are facing in the present study. Some of the most difficult problems in our work are cracks on the surface of the rock that can be mistaken for characters and structural background noise that interferes with the characters.

2. ALGORITHMS & TOOLS

In this chapter we examine algorithms that have been already established and we focus on whether or not they can be applied to our application. For the reasons explained herein, we turn our attention to mathematical morphology, rather than on conventional edge detection algorithms.
Concerning the post processing, the algorithms in the literature are very general, dealing with edge following, edge linking etc.

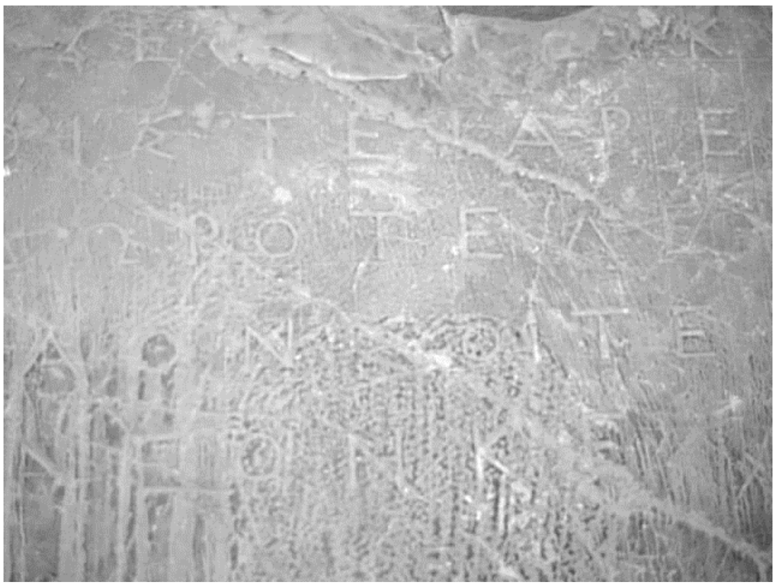


Fig. 2.1. The original image of the inscription

2.1. Mathematical Morphology

Edge detection is the most important part of our processing, since it provides the initial characteristics for description and/or recognition. If the edge detector discards information that is contained in the image, there is no way of retrieving it. On the other hand, if it over-detects edges, then it becomes difficult to get rid of the redundant information. Our aim is to preserve as much existing information as possible and use post-processing of the image to remove excessive edges.

We use mathematical morphology for edge detection due to its established potential in detecting vaguely defined edges. According to Lee, Haralick and Shapiro [4] mathematical-morphology edge detectors outperform most of the classical edge detection algorithms. Another reason is the fact that mathematical morphology has been around only for about 30 years. This means that there is plenty of work to be done in this field, contrary to the conventional techniques. A third reason is computational speed. These algorithms are faster than the complicated conventional edge detecting algorithms, although speed is not a matter of priority in our work.

Before the algorithms are presented, an introduction to mathematical morphology is required.

2.1.1. Basic Morphological Operations

Any given image can be represented as a two-dimensional set of pixels. Morphological operators work with two images. The first is the data image and the second is the structuring element. This is also an image and resembles the kernel of a convolution operator. The structuring element can have any shape, which can be considered as a parameter of the operator. The basic morphological operators are two, namely dilation and erosion.
First we consider the case of binary images. Let A be the set of points representing the binary “one” pixels of the original binary image *f* and B be the set of points representing the binary “one” pixels of the structuring element s.

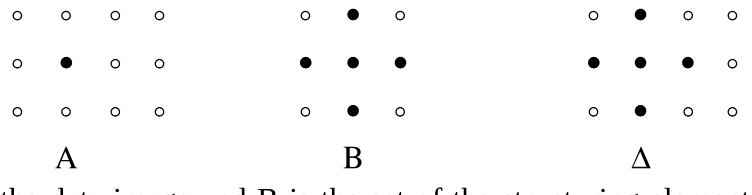
The dilation of the binary image *f* by the structuring element s, denoted by A⊕B, is defined by:

$$A \oplus B = \bigcup_{a \in A} \{b + a \mid b \in B\}$$

The erosion of the binary image *f* by the structuring element s, denoted by AÈB, is defined by:

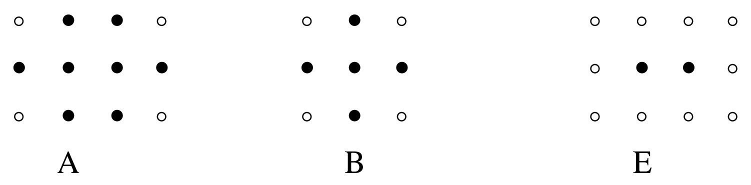
$$A \Theta B = \{p \mid B + p \subseteq A\}$$

To clarify the above, we can describe intuitively the operators. Let the structuring element have the shape of a cross of size 3x3 pixels. Morphological operations work very similarly to convolution. When performing a dilation on the input image the structuring element scans the entire image. If its reference pixel overlaps a binary “one” pixel it replaces all the pixels it covers with binary “one” value.
In the literature there are two different views regarding the reference point of the structuring element, some researchers use the center pixel, others use the top left pixel. We will use the center pixel for the purposes of this work.
The example below is characteristic of a binary dilation.



If A is the set of the data image and B is the set of the structuring element then Ä is the set of the image which results from the dilation of A by B.

Erosion works in a similar fashion like dilation. The difference is that instead of having to find a single binary “one” pixel in the original image, the structuring element has to find a group of binary “one” pixels exactly like itself. If that happens it assigns all of them a “zero” value, except from the pixel that is overlapped by the reference pixel of the structuring element. An example will make this more comprehensive.



If A is the set of “ones” of the original image and B the set of the structuring element, then E is the set that results from the erosion of A by B.
The E set has two binary “one” pixels because the structuring element “fits” on the original image on two different occasions.

Dilation has the effect of filling in small (relative to the size of the structuring element) holes and protrusions into the image, whereas erosion eliminates small components and extrusions of the image into its complement.

Apart from the two basic operations described above two other operations can be defined as a combination of the previous. These are the closing and the opening, which can also be considered as basic operations.

The closing of an image f by a structuring element s is a dilation of f by s followed by an erosion of the previous result by s.

$$A \bullet B = (A \oplus B) \Theta B$$

Intuitively, the closing smoothes the contours of the objects in the image, by blocking up the narrow *channels,* the small *lakes* and the long, thin *gulfs* [5].

The opening of an image f by a structuring element s is the erosion of f by s followed by the dilation of the result by s.

$$A \circ B = (A \Theta B) \oplus B$$

Intuitively, the opening smoothes the contours of the objects in the image, cuts the narrow *isthmuses*, suppresses the small *islands* and the sharp *capes* of the objects in the image [5]. The opening operation also eliminates single pixels, or small groups of pixels, that resemble the salt and pepper noise.

The extensions of the morphologic transformations from binary to gray-scale domain in the mid-80’s introduced a natural morphologic generalization of the dilation and erosion operations, along with their combinations.

The dilation of a gray-level image f by a gray-scale structuring element s is defined by

$$\mathrm{d}\,(r\,, c) = \max\,(f\,(r-i,\, c-j) + \mathrm{s}\,(i\,, j))$$

where, the maximum is taken over all (*i* , *j*) in the domain of s such that (*r-i, c-j*) is in the domain of *f*. The domain of d is the erosion of the domain of *f* with the domain of s [4].
The erosion of a gray-scale image *f* by a gray-scale structuring element s is defined by

$$e(r , c) = \min (f (r+i , c+j) - s (i , j))$$

where, the minimum is taken over all (*i* , *j*) in the domain of s. The domain of e is the domain of *f* eroded by the domain of s [4].

The definitions of the opening and closing operations do not change, rather they use the new definitions of the dilation and the erosion for gray-scale images.

Unfortunately, gray-scale morphology is not as easy to conceive, but it is possible to give examples like the ones given for the binary case.

Let assume we have a one dimension signal, where we want to detect edges. The structuring element used is a horizontal rod of size 3 and gray-level value of 20, which means that we are interested in the pixel under consideration and its two neighbors.
In the following figure we can see the original signal in the first row. This signal represents the step edge. The numbers on the left and the right are the gray scale values of the pixels. Sampling is performed on the vertical lines.

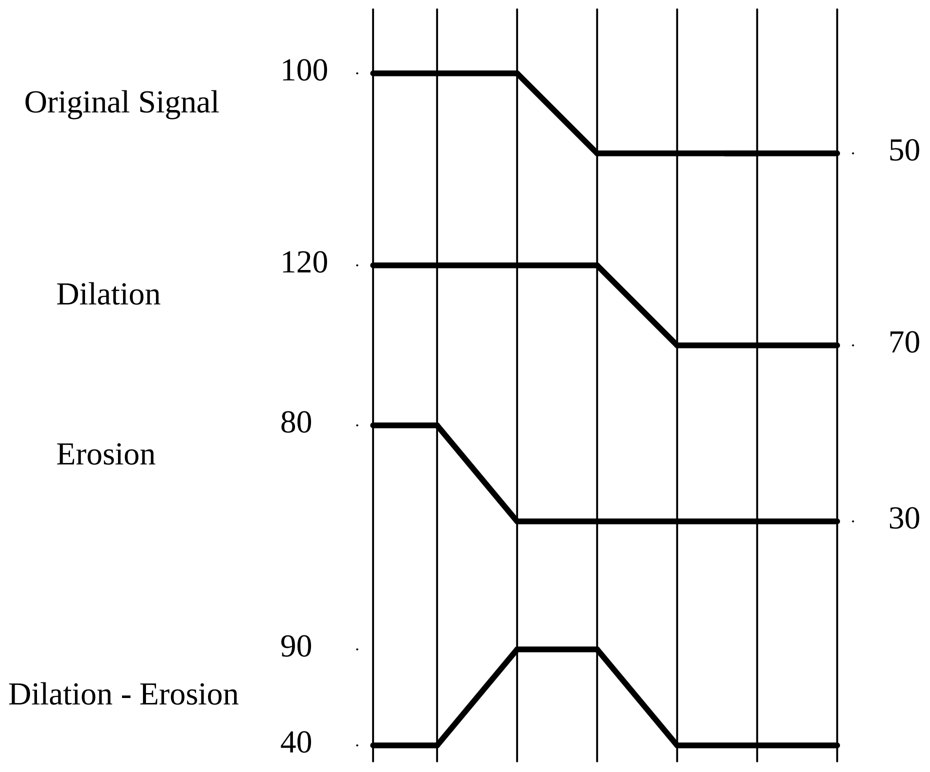


Fig. 2.2.

For example, in the original signal the first three pixels have a value of 100 and the next four 50.

The dilation of this signal is as shown in Fig. 2.2. The dilation operation adds to the values of signal the values of the structuring element and keeps the maximum value. Hence, the only change will be in the fourth pixel. The element covers pixels 3,4 and 5. But, because pixel 3 had an original value of 100, added up to the element makes it 120, it is the maximum of the three, so pixel 3 is assigned this value.
Similar, we can apply the erosion operation, where the value of the structuring element is subtracted from that of the original image and the minimum value is kept.
This figure also illustrates the simplest morphological edge detector. It is obvious from Fig. 2.2 that when subtracting the eroded signal from the dilated one, the edge stands out.

2.2. Edge Detection Algorithms

Several morphological edge detection techniques have been proposed in the past. The most promising were the following. We will omit to mention the structuring element in the course of the description of the algorithm and we will define in it the end of each description, since as we will see it does not play a major role as a parameter.

2.2.1. Gradient [6]

This is a very simple way to extract the edges of an image using mathematical morphology, as illustrated above, and is described by.

$$f_{\text{edge-strength}}(i,j) = \text{dilation}(f_{\text{input}}(i,j)) - \text{erosion}(f_{\text{input}}(i,j))$$

First an erosion and then a dilation are performed on the original image. By subtracting the eroded image from the dilated image, the edges of the original image remain, because dilation expands the objects in the image and erosion shrinks them.

This scheme works well for binary and gray scale morphology in case of step edges, but does not perform well in the images in our application, where the edges are vaguely defined and far from ideal. The result is shown in Fig. 2.3.

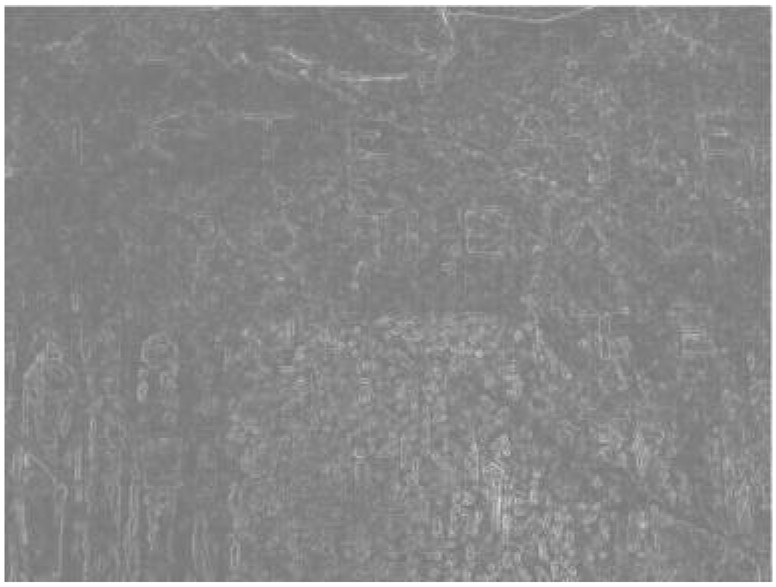


Fig. 2.3. Output of gradient algorithm

The structuring element used in this algorithm is a cross of size 3 and a gray level value of 60.

Increasing the gray level value of the structuring element results in an increase of the gray level value of the entire image, while decreasing the structuring element has the opposite effect. This is obvious since dilation adds the gray level values of the image and of the structuring element. When subtracting the eroded from the dilated image, the result increases when increasing the values of the structuring element and vice versa.

On the other hand, increasing the size of the structuring element decreases the effectiveness of edge detection and most edges are lost, because they have a small width and a large structring element is not able to locate the exact position of these edges. This is easy to comprehend if we assume the structuring element in Fig. 2.2 to have a width of 7 pixels. The final edge would be very wide and its exact location would be impossible to find.
Furthermore, the problem in this application is a little more complicated. The actual width of an edge is only the width of the transition from the background to the gray-level value of the carved part or the opposite and not the width across a letter itself. This means that the structuring element must be small enough to fit inside the carved part of the inscriptions, so that it can detect both these transitions.
As a result, instead of a single line for every letter that accounts for one edge, we get the perimeter of each letter; i.e. for the letter I we have an outside and an inside rectangle. It is emphasized that this is not a problem of the specific algorithm, but of the general concept of mathematical morphology.

2.2.2. Regularized Gradient [7]

Another algorithm, a little more complicated than the gradient, is the regularized gradient. It is described by the following equations.
First, the input image is blurred.

$$f_{blurred}(x, y) = \text{median}(f_{input}(x, y))$$

The blurred image is dilated and eroded using a cross structuring element (S_i) of size (1+2*i) for $0<i<4$.

$$d_i(x, y) = \text{dilation}(f_{blurred}, S_i)$$
$$e_i(x, y) = \text{erosion}(f_{blurred}, S_i)$$

The eroded image is then subtracted from the dilated image taken with the same structuring element.

$$h_i(x, y) = d_i(x, y) - e_i(x, y)$$

The gradient operator described above is the applied to this image.

$$g_i(x, y) = \text{gradient}(h_i(x, y))$$

Finally the minimum is taken of all the images taken as the output of the gradient.

$$f_{edge\text{-}strength}(x, y) = \min\{g_i(x, y)\}, \text{ for } 0<i<4$$

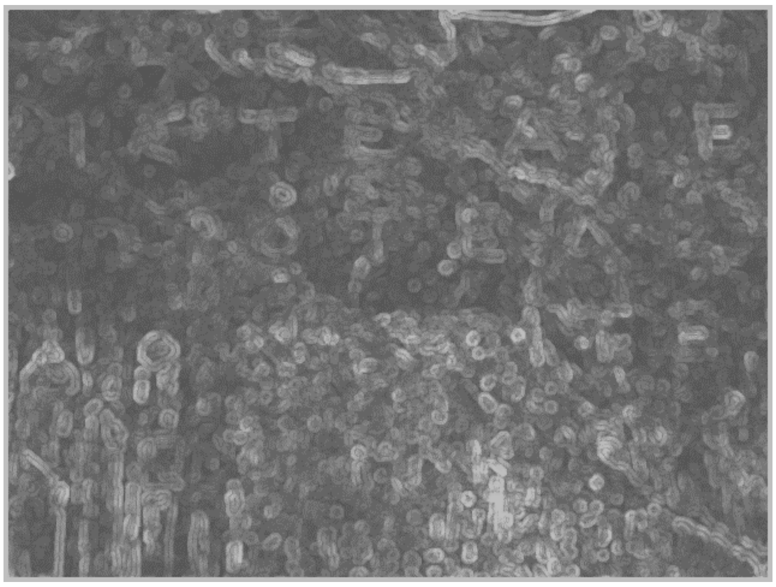


Fig. 2.4. Output of the regularized gradient algorithm

The result of this algorithm is a little better than that of the gradient algorithm, but due to the size of the structuring elements used (up to 7x7) the edges are almost lost.
Due to the large size of the structuring element, the detected edges appear very wide and blurred as discussed previously.

2.2.3. Blur and Minimum Operator [4]

This algorithm consists of the following steps.
The first step is to blur the original image.

$$f_{blurred}(i, j) = blur\{ f_{input}(i, j) \}$$

The blurred image is eroded and dilated. In the eroded image the object appears slightly smaller, whereas in the dilated image it appears slightly larger than the original.
The next step is to subtract the eroded image from the blurred one and to subtract the blurred image from the dilated one. This step produces two new images.

$$f_2(i, j) = f_{blurred}(i, j) - erosion(f_1(i, j))$$
$$\text{and } f_3(i, j) = dilation(f_1(i, j)) - f_{blurred}(i, j)$$

As a result, f_2 contains the inner boundary of the object and f_3 contains the outer boundary of the object. The edge pixels in the two images overlap and an OR combination would produce very thick lines. Instead, the minimum is taken to derive thin edge-lines, i.e.

$$f_{edge\text{-}strength}(i, j) = \min\{ f_2(i, j), f_3(i, j) \}$$

This algorithm was developed to find edges of bright objects on dark background. Although it is not very sensitive to additive noise, it does not perform well in our application. The algorithm does not succeed in separating letter edges from background edges, since gray-scale transitions are not clearly defined on the image.

The result of this algorithm on the input image is the following:

Fig. 2.5. Output of blur and minimum algorithm

In order to get this result an averaging 3x3 filter was used for blurring. The structuring element used was a cross of size 3 and gray level value 100.
The effects of the structuring element discussed for the gradient algorithm apply also to this approach.

2.2.4. Alpha-Trimmed Multidimensional morphological edge detection [8]

This algorithm also blurres the image as its first step, but introduces a new kind of blurring. This blur operation has similarities with the averaging filter. It takes the average of the pixels spanned by the structuring element, except from the á smallest and the á highest pixels values. This helps reducing salt and pepper noise. The equation that describes this operation is the following:

$$f_a = \sum_{i=a+1}^{k-a} \frac{\tilde{f}_i}{(k - 2 * a)}$$

where $\tilde{f}_i$ represents the ordered input values.

At the next step, an erosion and a dilation are performed on the á trimmed blurred image. Subsequently, an opening is performed on the eroded image and a closing on the dilated image. Then, the eroded image is subtracted from the opened and the closed from the dilated, creating two new images.

Finally the minimum of these two images is taken at every pixel. The following equations clarify the above operations:

$$e(i,j) = \text{erosion}(f_a(i,j)) \qquad d(i,j) = \text{dilation}(f_a(i,j))$$
$$o(i,j) = \text{opening}(e(i,j)) \qquad c(i,j) = \text{closing}(d(i,j))$$

$$f_{ATM}(i,j) = \min\{(o(i,j) - e(i,j)), (d(i,j) - c(i,j))\}$$

This algorithm yields good results on our input images. Its major problem is that it produces “double” edges, representing both the inner and the outer edges of the objects.

This effect can be easily explained on the basis of the effect of the individual operations employed. Lets consider an I that is a vertical line 5 pixels wide (letters on the inscription are actually wider.) Applying the dilation operation, using a 3x3 structuring element, expands the I by one pixel in each direction. Then the closing smoothes the curve of the letter. Subtracting the closing from the dilation produces an edge that fits the letter on the outside.

Similarly, the erosion shrinks the object by one pixel in each direction. The opening again smoothes the contour and the subtraction of the two produces an edge that fits the I from inside.

The algorithm also returns many small edges, which are caused by noise.

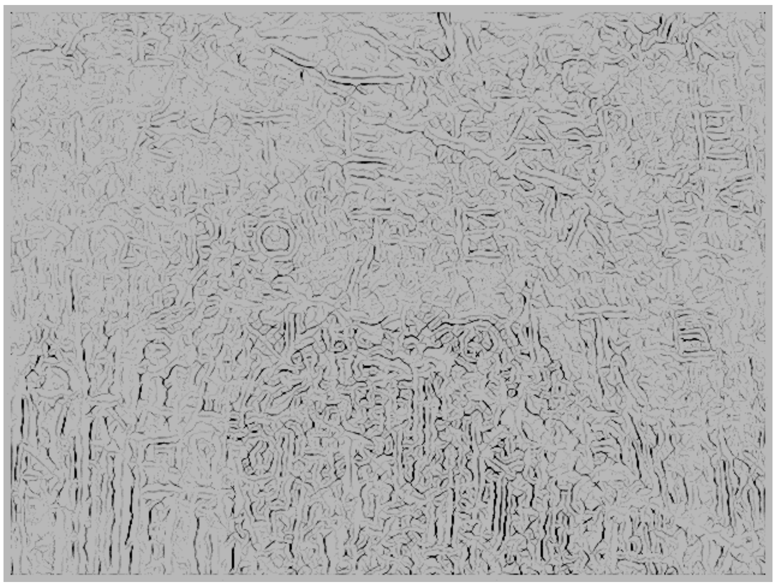


Fig. 2.6. Output of ATM algorithm

The image in Fig. 2.6 is obtained using a cross as a structuring element of size 3 and gray level value 60. The blurring filter used is 5x5 and alpha is 1. Experimenting with the gray level of the structuring element showed that the best value was 60, but if increased as much as 110 still returned good results. Outside this range (60-110) the results are not as good. The “double edge” effect is obvious in the above image.

An interesting fact is that contrary to the other edge-strength images, the result of this algorithm has dark edges on a bright background. The reason for this is that during the morphological operations some pixel values become negative. To avoid dealing with negative pixel values a histogram normalization is performed to bring the pixel values to their normal range (from 0 to 255). In the process of normalization, the background that is originally black (gray level 0) becomes white (gray level 255), whereas the edges that are originally negative become dark. We will further discuss this in the following chapter.

2.2.5. Histogram-based Morphological Edge Detection [8]

This algorithm introduces a new kind of morphological operators, namely histogramic erosion and dilation. These operators make use of the histogram of the image instead of the actual gray levels of individual pixels. Histogramic erosion is defined as the gray-level intensity at which the histogram height is the maximum of all histogram values at gray level intensities lower than the intensity of the pixel.

$$d_h(x, y) = \{ g_j \mid h(g_j) = \max [h(g_j), h(g_{j+2}), \dots, h(g_{l-1})] \text{ and } (i-1 < j < l) \}$$

Histogramic dilation is defined as the intensity at which the histogram height is the maximum of all histogram heights at intensities greater than the intensity of the pixel.

$$e_h(x, y) = \{ g_j \mid h(g_j) = \max [h(g_0), h(g_1), \dots, h(g_i)] \text{ and } (0 <= j < i+1) \}$$

The HMED operator, after blurring the input image, produces an edge strength image in which nonmaxima suppresion has to be performed. In order to take the edge strenght image histogramic dilation and histogramic erosion are used:

$$f_{HMED}(x , y) = \min (f_{blurred} (x , y) - e_h(x , y), d_h (x , y) - f_{blurred} (x , y))$$

Where e_h and d_h are histogramic eroded and dilated images. Having this image we can either threshold it, or perform nonmaxima suppresion to get the edges.
In the second case, we suppress a pixel as a non-edge pixel if there exists a group of pixels whose value is much greater than the pixel to be suppressed.
This operator was developed for edge detection in oceanographic images.

Figure 2.7 is the result image of the HMED algorithm. Although the nonmaxima suppression was not applied, the histogram content of the image shows that it does not contain any useful information regarding the edges and it is obvious that it does not work in our application.

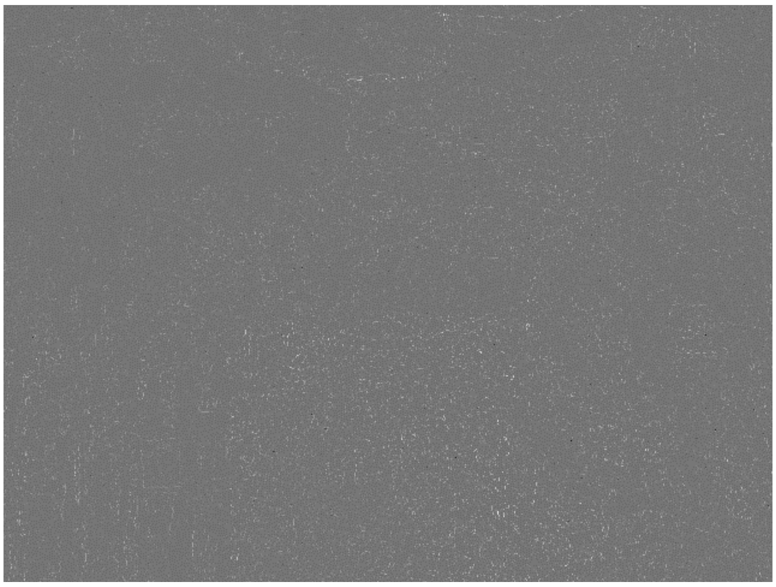


Fig. 2.7. Output of HMED algorithm

We mention here that apart from the above algorithms some combinations of algorithms were tried without any noticeable results.

2.2.6. Comments on the algorithms

2.2.6.1. Blurring

As we have seen above, all the morphological edge detection algorithms use blurring as a first step before applying any morphological operators. The reason is that these label a pixel as an edge pixel if it has gray level between two extremes in the area spanned by the structuring element centered at the given pixel. Therefore, if an ideal

step edge is encountered the algorithm will not work properly because neither the smallest valued pixel nor the highest valued pixel has neighbors with higher and lower gray level values [4]. Thus in the case of a step edge, the blurring will transform this edge to a ramp, satisfying the above criteria.
However, blurring has a negative effect on the image and the edges produced, because it spreads the edges. When they are not ideal step ones, the algorithm tends to further spread them resulting in a loss of the information of the actual location of the edge, as well as of its intensity.
In our application this disadvantage is actually desirable in a small extent. Due to this effect, the small gray-scale variations on the surface of the rock that are caused by the corrosion are smoothed. Thus the edge detector does not recognize them as edges; this is a first stage of noise elimination.
We conclude then that only a small amount of blurring should be introduced in order for the detected edges to resemble the actual ones as close as possible.

2.2.6.2. Structuring Elements

Some issues regarding the structuring elements can be also summarized. The optimal size was found to be 3x3, which is expected in our application. The edges in our input images are quite thin, since the letters are small. This means that a small structuring element must be used to involve only a small neighborhood around the edge. If the element is large, many pixels that do not belong to the edge are taken into account and the algorithm cannot determine the exact position of the edge. The reason is that the larger the structuring element is, the more the objects will be expanded or shrank, see Fig. 2.2. If the element is 7x7 pixels, then the objects will be expanded by 3 pixels in case of a dilation. Subtracting the dilation from the original image an edge 3 pixels wide is produced, even in the case of a binary image input. A good example to illustrate this is the regularized gradient, where large structuring elements were used. It is easy to see that the algorithm could not find the exact position of the edges, resulting in very thick edges.
Concerning the gray-level value of the structuring elements, most algorithms work very well using a value of about 60, although some work optimally with the value of 100. The optimal value used is very much dependent on the lighting and the gray level of the image itself. For most algorithms a good value is one that when added to and subtracted from the gray level of the image gives results in the 0-255 range. The reason is that gray scale morphological operations add and subtract the values of the images and the structuring element. As a result, if an image is very dark or very bright a structuring element with smaller gray level value will have to be used.
One could mention that larger gray-levels can be used for the structuring element and histogram normalization can be applied to bring the image back to the visible value range (0 to 255). Although the use of larger values is possible, there is no evident advantage to their use. The output of the algorithm is not be improved, since the transitions from edge to background are not altered and a histogram normalization has to be applied, adding to the complexity of the algorithm.
The shape of the structuring element is also important. The shape that worked best for the algorithms is the cross, because it can detect edges on all directions. If a horizontal rod is used, only vertical edges would be detected, since the algorithm has to locate a change in gray level along the element in order to detect edges. Bearing in mind that

we have a 3x3 limitation in size, due to the size of the letters in our input images, a 3x3 cross is the best compromise.

2.3. General Algorithms

2.3.1. Histogram Normalization [9]

Histogram normalization is used extensively during this work. It is very simple, but also very important for improving the uniformity of contrast within the image.
This algorithm takes an image as input and normalizes its histogram to span the entire 0 to 255 range. The input image's histogram might be located in a subset of this range, in a region larger than 255, or even in the negative gray scale range. The original histogram is shifted, shrank or expanded so that the output image's histogram lies exactly in the 0 to 255 range.
The transformation that achieves this normalization is given by:

$$a = \max \{ f(x, y) \}$$
$$b = \min \{ f(x, y) \}$$

$$g(x,y) = 255 * \frac{f(x,y) - b}{a - b}$$

where f is the original image and g is the output image whose histogram is normalized.
The main use of this algorithm in our work is to bring the histogram of each image in the visible range (0 to 255). Having negative pixels or pixel values of over 255 is not a major problem for processing purposes. The problem is these images cannot be seen correctly, because images are stored and viewed as arrays of 8 bit characters, 8 bits are allocated for each pixel. This way, if a pixel is negative it will be assigned a zero value and if it is larger than 255, it will be assigned 255 as its value. Therefore important information is lost, because pixel values are truncated to this range without any care taken for the rest of the pixels. That results in a poor contrast either locally or globally.

In Fig. 2.8 the histogram of the original picture of the inscription is shown. The pixel values are concentrated, and as a result the contrast of the image is very low, which can easily be observed.

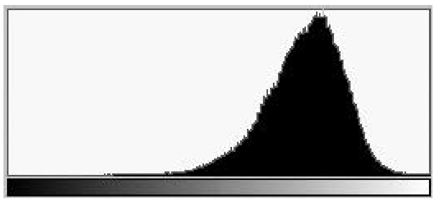


Fig. 2.8. Histogram of original image

If a histogram normalization is applied to this image the resulting histogram will be that of Fig. 2.9.

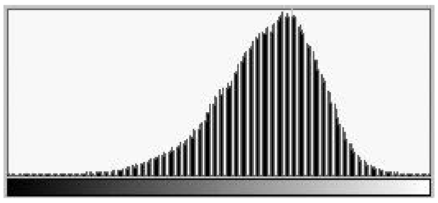


Fig. 2.9. Normalized histogram of original image

The new histogram is spread, in order to expand from 0 to 255. The lower value of the first histogram, which was not zero, has been dragged to zero and the entire histogram has expanded. This results in an increase of contrast, since the value of the pixels are more clearly separated.

2.4. Post Processing Algorithms

2.4.1. Contour following [10][11]

This is a well known algorithm for contour following and the extraction of closed edges from the background.
Assuming that edges follow smooth curves, it uses the direction as a criterion for connecting edge pixels. It examines the neighbors of every pixel along the direction of the edge. If one of them (on either side) has an edge direction that matches the slope of the edge pixel in consideration, the neighbor is linked to the edge, else it is discarded. If no such pixel is found it can skip one pixel, or even two, and continue the search beyond that point. This can also help in filling small gaps that might exist along the edge curve. In order to find the slope of each pixel the gradient is used.
Contour following can successfully extract curves that are perimeters of bright objects on a dark background and vice versa. In order to trace the curve, it requires that pixels on one side of the edge are dark and on the other they are bright, so that the algorithm can clearly assign a gradient value and determine whether a pixel lies on the boundary of the two areas by the direction criterion alone.
This algorithm can be extended to include the strength (the gray level value) of the edge-pixel as an additional criterion. That means that a pixel is linked to the edge only if its slope is close to the edge’s slope and if its (gradient) magnitude exceeds a certain threshold. This version of the algorithm can be used for our application after the edge detection phase. Having the edge-strength image, we could follow each edge and add another pixel if its gray level value exceeds a threshold and if its slope is similar to that of the curve. The problem in applying this to our application is that the curves we want to detect are by no means smooth since many angles exist on letters.

2.4.2. Raster tracking [10]

Another algorithm by Rosenfeld and Kak is raster tracking. This can be used for cleaning small edges that are caused by noise rather than actual edges in the image.
This algorithm scans the image row by row, assuming that the edges to be traced are thin, dark and continuous curves, whose slopes never differ far more than 90°. If a given pixel (x,y) exceeds a relatively high threshold T_h, we accept it as an edge pixel. We also accept any neighboring pixel that lies in the previous row (y-1), which exceeds a lower threshold T_l.
As a result, if an edge has a single pixel with a gray level value exceeding T_h, the line will be tracked from there and downwards until one of the edge pixels is lower than T_l, in which case the edge will break. This procedure can lose the entire edge or can detect only parts of the main edges of the image, because parts will be lost if the uppermost pixel does not exceed T_h.
This algorithm is very sensitive to the thresholds and especially $T_{h.}$

2.4.3. Double thresholding

A slightly different version of the previous algorithm is double thresholding.
Suppose that the input image is a black background and a gray edge. Select two thresholds T_1 and T_2, where $T_2=T_1+$å. Create f_1 and f_2 two new images by thresholding the original image twice using these thresholds, f_1 contains pixels larger than T_1 and smaller than T_2, whereas f_2 contains pixels larger than T_2. This way f_2 contains few edge pixels that have large values and are very likely to belong to an actual edge. f_1 contains more pixels, ideally every pixel that belongs to an actual edge and inevitably some noise pixels.
The algorithm can now proceed in the following manner. After thresholding, f_2 is scanned. If an edge pixel say $f_2(x,y)$ has a neighboring edge pixel in f_1 then assign this pixel in f_2 as an edge. This process is repeated until no change is made in f_2.
This way we achieve to keep every pixel (larger than T_2) that definitely belongs to an edge and to get rid of edges that have pixels of low gray level value, which are probably noise pixels.

3. PROPOSED ALGORITHMS

In this chapter we describe all algorithms we developed for this application. Some of the algorithms have been mentioned in the previous chapter. They have been adjusted so as to have optimal performance, in accordance to the requirements of our application. In other areas, we propose new algorithms designed to match the requirements of the application. Along with the description, the reasons that led us to these algorithms will be investigated.

3.1. Pre processing

This is a procedure by Gonzalez and Woods [9] that is extensively used before the edge detection, in order to improve the quality of the image fed to the edge detector. The effect of image equalization is that the background pixels are assigned values near a reference gray scale value and, at the same time, the contrast between the foreground and the background pixels becomes more uniform. The main reason that can cause variations on the values of the background pixels is the lighting conditions when taking the images. For instance, excessive lighting may result in an image such as the one shown in Fig. 3.1(c), with large variations at the incident light distribution.
In our application the reason we use this algorithm is the uniformity of the edges, rather than the correction of the background. Morphological edge detectors detect edges based only on spatial information in the neighborhood of each pixel. As a result our primary concern is the contrast between foreground and background pixels and not the background level itself. The reason is that, as we saw in the previous chapter, most morphological edge detectors involve a subtraction between images which means that background values in on image are subtracted from similar values of the other. However, it is important to us that letters across the image have the same contrast with the background regardless of the original lighting conditions of the image. This is very important since different contrast across the image between letters and background will result in edges that have different gray levels in the edge-strength image. If the contrast is large the output edge will have a larger gray level than if it were smaller. As a result applying a threshold becomes more difficult, because the edges do not have similar gray levels across the image in the output of the edge detector. The assumption of uniform contrast does not hold for the original image, where very bright areas have larger contrast (in variance of gray levels) than darker areas.

A quite simple algorithm for equalization is the following:
First of all the original image is blurred using the blurring operation of the ATM algorithm, described in chapter 2. A 7x7 filter is used with alpha equal to 1.
Let f be the original image. Then:

$$g(x,y) = \text{ATM blur}\{ f(x,y) \}$$

Subsequently the mean gray level value of the blurred image is calculated.

$$mean = \frac{\sum_{\substack{x=1 \\ y=1}}^{\substack{height \\ width}} g(x,y)}{height * width}$$

Finally the original image is divided by the blurred image pixel by pixel and the result is multiplied with the mean value we have calculated.

$$e(x,y) = mean * \frac{f(x,y)}{g(x,y)}$$

As a last step, a histogram normalization is applied to the resulting image, so that the pixel values are clearly separated.

In Fig. 3.1. an example of applying this equalization algorithm to an image with large variations in the background level is presented.

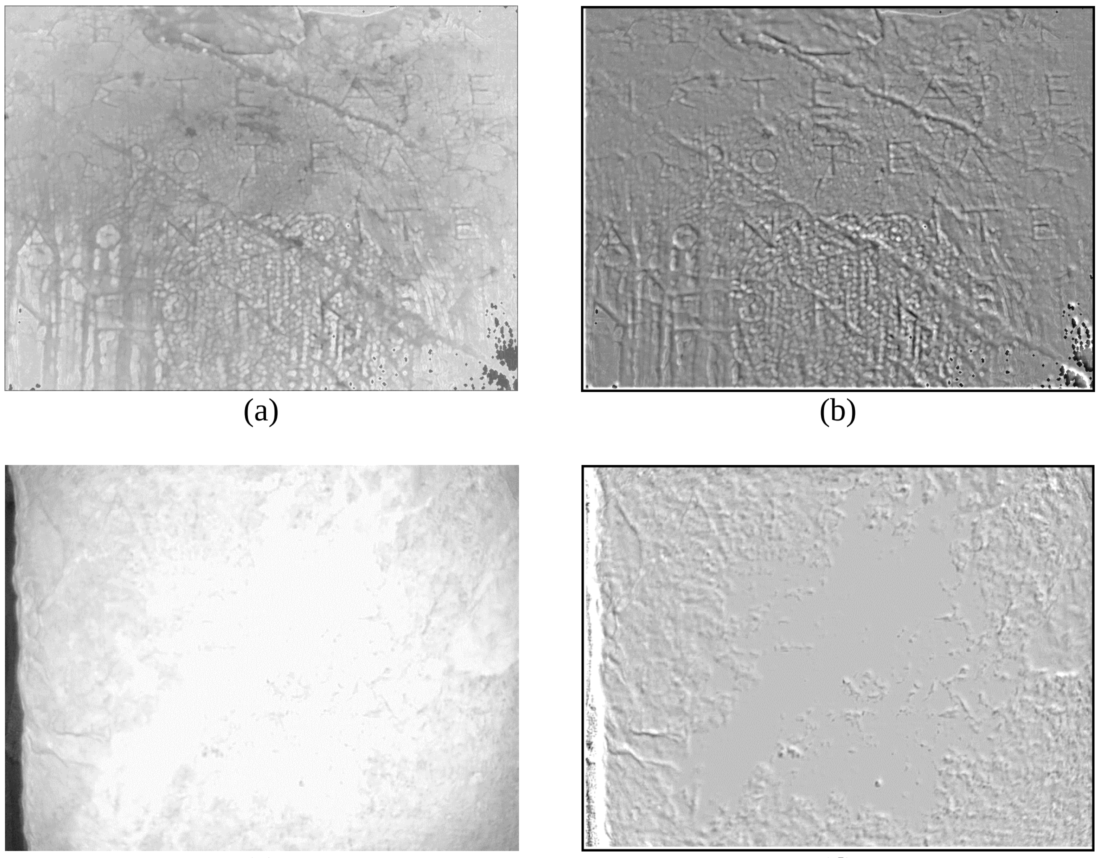

(a) (b) (c) (d)

Fig. 3.1. (a) Image with large variations in background, (b) the result of equalizing (a), (c) very bright image, (d) the result of equalizing (c)

As a conclusion, the equalization algorithm manages to suppress the difference in the background in the various parts of the image, while retaining the contrast locally. The difference between background and foreground pixels changes slightly, but the background pixels are forced to a reference gray level. This is clearly illustrated in Fig. 3.1. The letters that can be identified in the left hand side images can be identified in the right hand side images also, but the background in the second set of images is almost uniform. Moreover, histogram equalization manages to enhance the contrast between letters and background.

3.2. Edge detection

It is obvious from the previous chapter that the algorithm yielding the best results in edge detection for our application is the ATM algorithm. However, in spite of its quite good results many noise edges are detected among the letters. This is an important problem along with the fact that “double edges” are detected.
For these reasons we experimented with the algorithm to improve its performance. As input to the ATM algorithm in these experiments, the equalized image shown in Fig. 3.2 is be used.

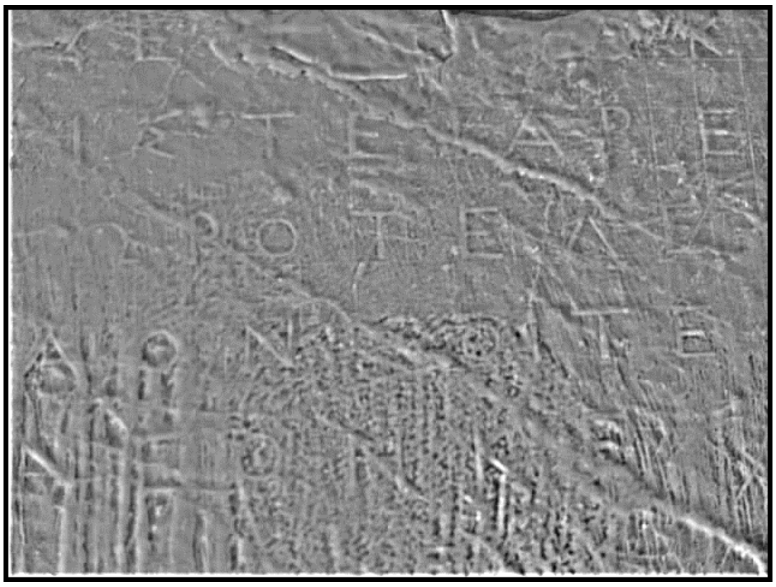


Fig.3.2. Input image for the edge detector

The ATM algorithm has two different parts; the first is the blurring operation that is applied to the input image and the second is the morphological operators that extract edges.

3.2.1 Blurring operation

In this phase we attempt to optimize the blurring operation of the ATM algorithm.
First of all we replace the averaging (blurring) operation with a median filter. This is a natural consequence from the characteristics of input image. The letters have a

slightly higher gray scale value (lighter color) that the background, not taking the shadows under consideration. The averaging filter used by the ATM algorithm spreads the edges of the letters, which are already quite wide. This is an undesirable side effect of the algorithm in our case. The median filter, contrary to the averaging, does not average pixels but keeps the median value within its neighborhood. This means that large deviations from the average gray scale values are eliminated, whether positive or negative. As a result, when applied to an area with constant value, whether background of letter, it merely suppresses large differences and eliminates the salt and pepper noise. When applied to an area on a border of a letter it keeps the value that the majority of the pixels have. This way the edges are not spread and deviations of values in smooth areas are slightly reduced as seen in Fig. 3.3.

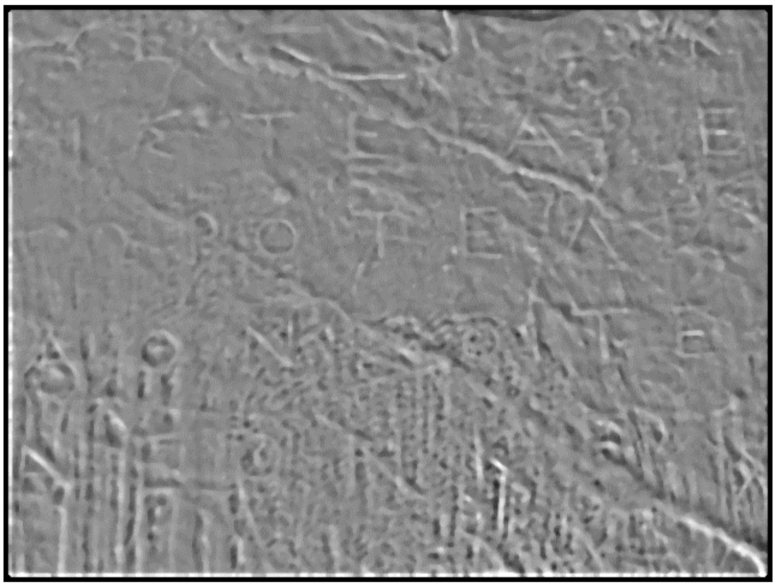


Fig 3.3. Equalized image filtered using 5x5 median filter

Substituting the ATM blurring operation with this median filtering and applying the algorithm we obtain the result of fig. 3.4.
It is easily seen that a major improvement has been achieved. The unwanted noise edges are reduced and the edges that belong to the letters are enhanced. This image is obtained with a median blurring 5x5 filter prior to the morphological operators.

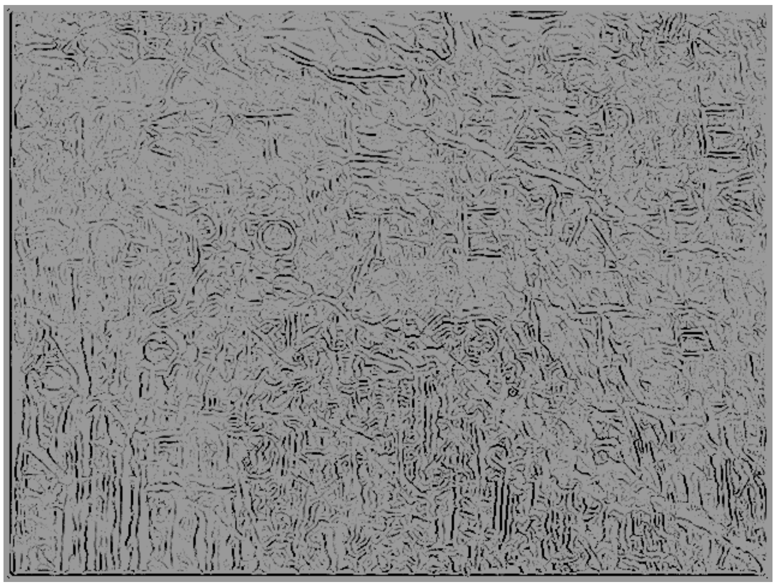


Fig. 3.4. Output of ATM using 5x5 median blurring

If a 3x3 filter is used instead of a 5x5 median filter we expect an output that is less blurred but preserves more noise effects. The corresponding image of the edge detecting algorithm is shown in Fig. 3.5.

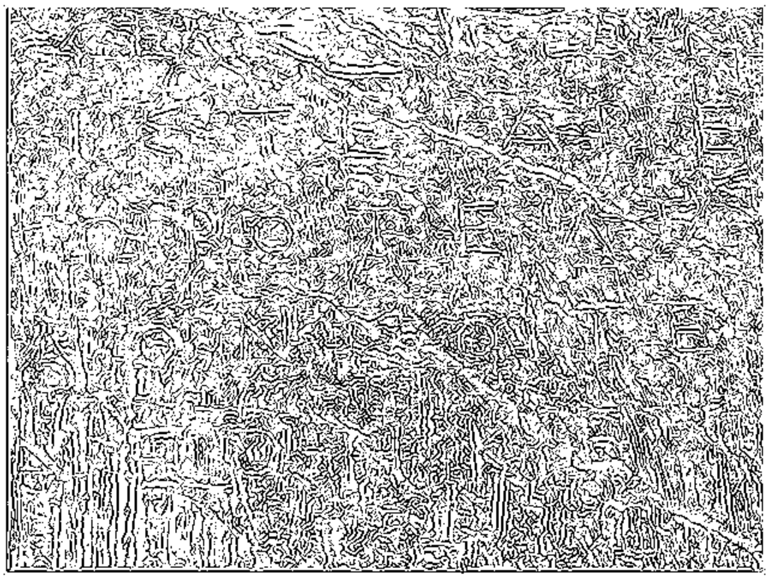


Fig. 3.5. Output of ATM using 3x3 median blurring

The edges here are much stronger as a result of less severe blurring. Although the edges we want to keep are further enhanced, the noise edges are also emphasized, creating problems to subsequent processing.

The 5x5 median filtering reduces the noise edges and slightly enhances the edges that belong to letters. To further improve this step of the edge detection, we use a combination of blurring filters.

The advantages of the median filter have been already pointed out. On the other hand the averaging filter also has some characteristics that are desirable to our application. The main one relates to its effectiveness in averaging large portions of the image, thus eliminating dark spots and shadows, caused by inadequate lighting, as in Fig. 3.1. These effects are clearly visible in the lower and central part of the equalized image. The surface of the rock is very rugged and lighting produces intense shadows that pose severe problems to the edge detection algorithm. If a large averaging filter is applied, then such spots are eliminated along with useful information in the image, since all the letters are averaged with the background.

Bearing in mind that the important information, i.e. the letters, are lighter in color than the background, a potential averaging combination that could be used involves the maximum of the two filters; the median and the mean-value filters. This way in the region of a letter the median will have a larger value than the averaging, so that it will be preserved. In the case of background, the averaging will probably have a value slightly larger, because it also averages brighter pixels belonging to letters. Dark spots will not present a problem, because the averaging will eliminate them, replacing them with larger values of the background. A possible shortcoming is the fact that the gap between letters and background is slightly decreased, but from the result it is observed that this effect is not crucial.

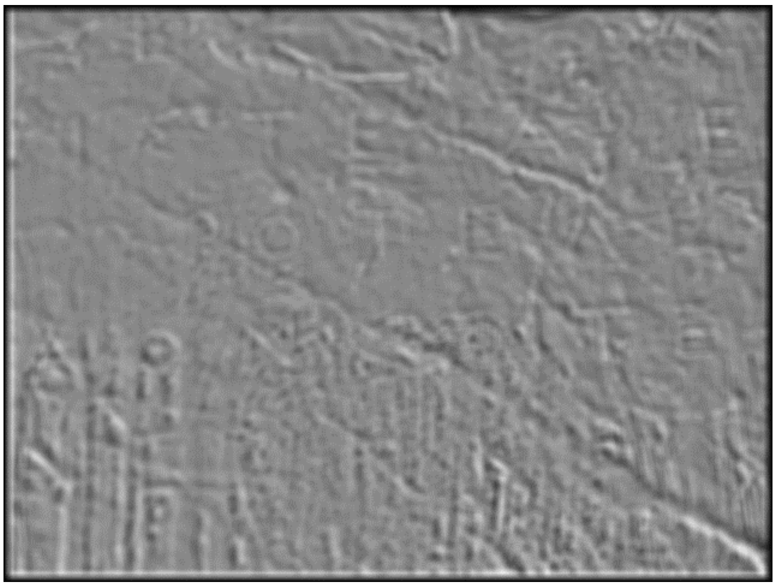


Fig. 3.6. Equalized image blurred using 7x7 averaging filter.

A 5x5 median and a 7x7 averaging filter are used on the equalized image. The averaging filter is larger in size because it should create a gray scale level of reference, which ideally falls between the values of background and letters in the median blurred image. Then, when the maximum of the two at every pixel is taken, only useful information is preserved.

The result of feeding this blurred image to the rest of ATM algorithm is shown in Fig. 3.7.

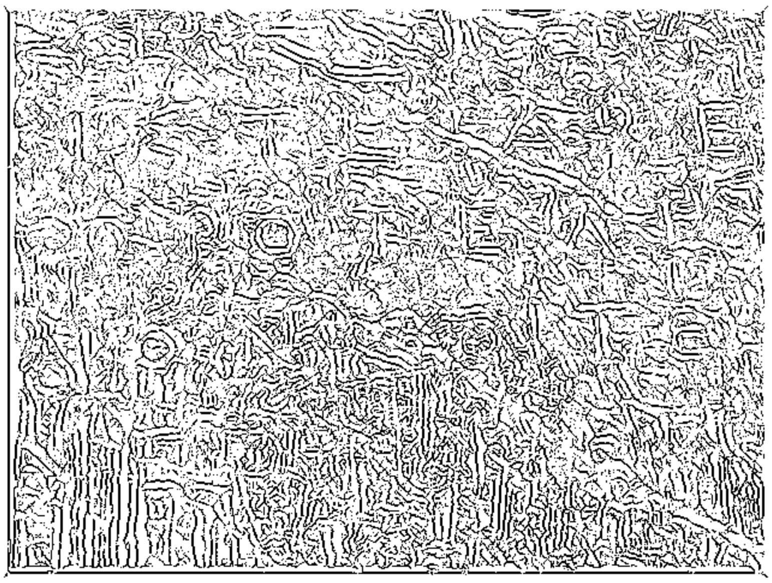


Fig. 3.7. Output of ATM algorithm using combination of blurring filters.

This is the best result we achieve in our application. It is obvious that the unwanted noise edges are efficiently eliminated, especially in the central and lower parts of the image. On the other hand, the edges that belong to letters are quite strong and they can be easily distinguished without further processing. As a result, this blurring operation is preferred and forms the first step of our edge detection algorithm.

3.2.2. Morphological processing

Following the specification of the blurring operation, we have to deal with the "double-edges" effect. This is caused by the fact that the ATM algorithm takes into account all edges produced by the erosion and dilation operators. In fact, the ATM algorithm works with two different images from which the minimum is kept at every pixel to produce the edge-detection output. The first image is the subtraction of the erosion of the blurred image from the opening of the eroded image (I_1). The second comes from the subtraction of the closing of the dilated image from the dilation of the blurred image (I_2).

The first attempt to overcome this problem was combination of the two images reflecting the edges. Instead of taking the minimum, we apply thresholds on these two images and accept a pixel as an edge pixel only if its gray values are smaller than the corresponding thresholds. In fact, we use a threshold of 240 for I_1 and 200 for I_2. If both values are smaller than the corresponding threshold, we keep the pixel from the second image. Alternatively, we preserve the pixel from the first image. In this way, if the pixel is an edge pixel in both images, we keep the lower value of the two (the one that accounts for the larger contrast). If it is not an edge pixel in both images, we keep the higher value, which would be very close to the background level. The result is shown in Fig. 3.8. It is easily seen that this approach has a large impact, resulting in complete elimination of the "double edges" effect.
The reason we use a larger threshold for I_1 than for I_2 is because we mainly want to keep the edges produced by the opening minus erosion part (I_1). This part produces the inner edge for each letter (the skeleton), rather than the outer (the perimeter).

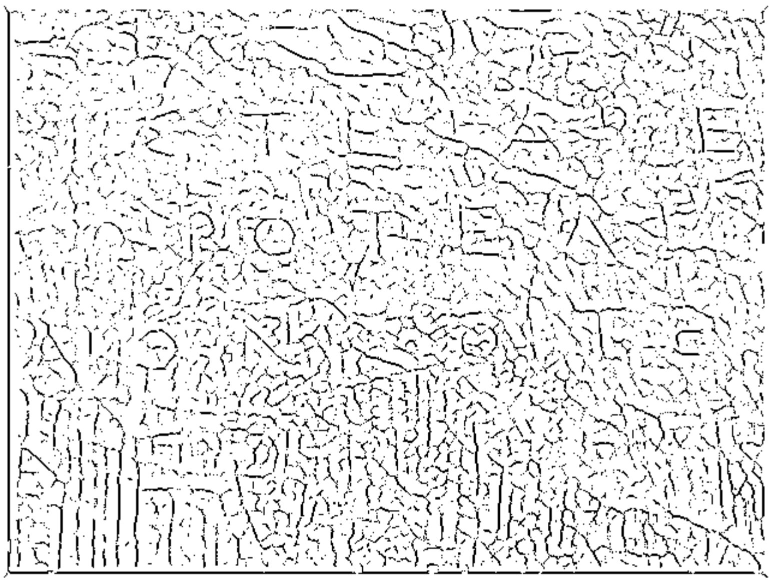


Fig. 3.8. Output of edge detection algorithm using thresholds

However, despite the fact that its output is very good compared to the previous results this method has one major disadvantage. We must assign the thresholds heuristically and a strategy for estimating them for each image must be developed. This is a very difficult task since the histogram of the output image is very concentrated and does not form any valleys to assist the process of threshold selection, as seen in Fig. 3.9.

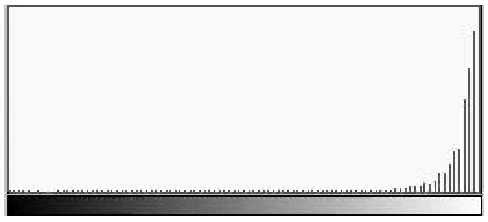


Fig. 3.9. Histogram of thresholded ATM

One possible approach for threshold selection would be to implement a strategy that found a threshold based on the percentage of pixels on each side of the histogram. For example, a candidate threshold could be the value that separates the 10% of the pixels as edges and the rest as background.
Even with a selection method, this edge detection scheme remains quite heuristic.

Another way to overcome the “double-edge” problem is to consider only one part of the algorithm; either the erosion and opening part, or the dilation and closing part. As we have discussed in the previous chapter each part of the algorithm returns either the inner edge or the outer edge of the letters, respectively. In order to get a single edge for each letter, we may select the opening minus erosion part. In this case, an edge will be obtained that fits the letter from inside. Hence, since the letters are not very wide, an edge resembling the skeleton of each letter will be produced. If we use the closing minus dilation part, then the edge would resemble the perimeter of each letter.

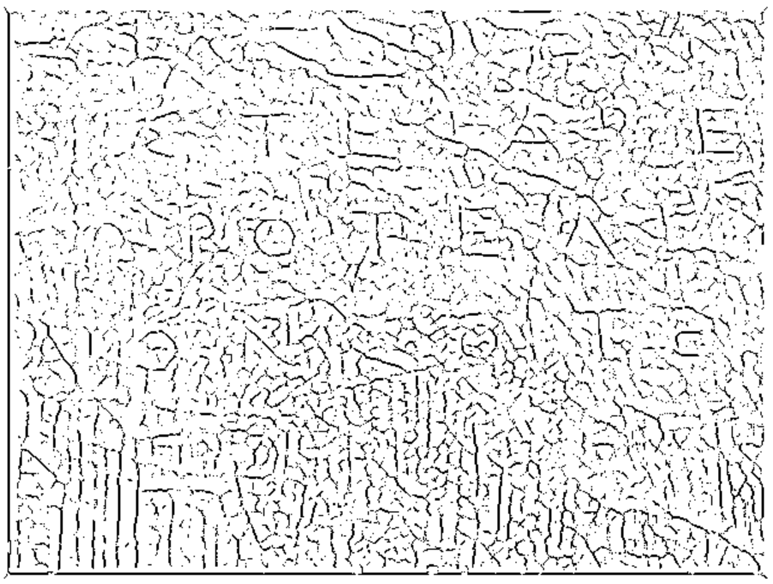


Fig. 3.10.

The quality of this image is as good as that we get using the two thresholds technique. Thus, we use this approach for two main reasons. First of all, it does not require any threshold selecting strategy. Secondly, although not as important, this second approach is computationally cheaper, since almost half of the morphological operations are executed.

Overall, we propose the following edge detection algorithm. The input image is the output of the equalization algorithm.

The blurring step is described by:

$$f_m(x,y) = \text{median5x5}(\ f_{input}(x,y)\ )$$
$$f_a(x,y) = \text{averaging7x7}(\ f_{input}(x,y)\ )$$
$$f_{blurred}(x,y) = \max\{\ f_m(x,y), f_a(x,y)\ \}$$

The morphological operations applied subsequently are given by:

$$e(x,y) = \text{erosion}\{\ f_{blurred}(x,y)\ \}$$
$$o(x,y) = \text{opening}\{\ e(x,y)\ \}$$
$$f_{edge\text{-}strength} = o(x,y) - e(x,y)$$

Although this algorithm is quite simple, it is the one that yields by far the best results in our application.

Here, we must further analyze the negative pixels phenomenon that was mentioned in the previous chapter when discussing the ATM algorithm.

The erosion, which is the first step after the blurring operation, subtracts the value of the structuring element from that of the image and keeps the minimum of these two values. Thus, the local maxima are lost since only the minima are kept. Let I_1 be the resulting image. Following the erosion, an opening is applied. The opening is an erosion followed by a dilation. This operation has no effect when applied to an area

with uniform gray scale values since no local minima and maxima exist. On the other hand, when transitions are present, the opening behaves differently on minima and maxima. First the erosion is applied and the local minima are kept reduced by the gray scale value of the structuring element, we will refer to this image as I_2. Then when the dilation is applied it is applied over I_2, so even though it keeps the local maxima increased by the value of the structuring element it does not reach the values of the pixels in I_1. The output of the opening will be referred to as I_3. All these operations use the same structuring element. Concluding, the opened image is expected to have smaller gray levels in areas of the image where transitions (edges) exist.
Also, as we have seen in the previous chapter, the opening smoothes the contours. This means that due to the fact that the edges are far from smooth the opening will produce a smoother edge contour than that of the erosion. This results in a different shape of the edges in the two images I_1 and I_3.
From all the above we conclude that by subtracting I_1 from I_3 the result is zero in areas of uniform gray scale values and negative in areas where transitions exist, on the perimeters of the letters.

As a result, a histogram normalization (from 0 to 255) is applied following the edge detection. By normalizing the histogram no information is lost, which is important for post processing and the image can be handled by a normal viewer.

3.3. Post processing algorithms

Following the edge detection phase, we obtain an image that contains many edges, many more than those carrying useful information, due to small cracks and other irregularities on the surface of the stone. Our target in this stage of the processing is to eliminate as much as possible these unwanted edges so as to preserve and, if possible, enhance the edges that belong to letters. At this stage enhancement refers to the connection of small (broken) edges that are parts of larger segments. This task becomes difficult, due to the lack of special characteristics of noise and/or useful edges.
Several algorithms were developed for this task. In the following, we discuss each of the algorithms developed and the reasons that led to their selection.

3.3.1. Labeling

A labeling technique is necessary at this stage on, so as to process edges as distinct and separate objects. Labeling implies that all pixels belonging to a single edge have a unique label and pixels belonging to different edges have different labels. A single edge is a group of foreground pixels that are 8-connected.
The algorithm developed for labeling has two steps. First of all, it scans the image once and assigns each edge pixel a different label, unless a neighboring pixel already has a label, in which case it assigns this label to the pixel under consideration. This ensures that all edge pixels have a label and the ones connected have the same label. In the next step, the labels on a single edge's pixels are forced to a unique label. The algorithm checks each pixel's neighbors. If one neighbor has a label l_2 different than

that of the pixel, it scans the entire image and replaces all labels l_2 with that of the pixel under consideration. For example, let a pixel with label 3 be examined. If it has a neighbor with a label 4, the entire image is scanned and all the pixels having label 4 are assigned label 3 as their new label. This second step is repeated until no change is made to the labels and assures that all the pixels that belong to a single edge have a unique label. The first step of the algorithm makes sure that no different edges have the same label, since only connected pixels are assigned the same label. The second step further improves the labeling process by assigning the same label to all neighboring pixels.
This technique is time-consuming, since the image needs to be scanned many times in order to derive the final values of the labels.

3.3.2. Cleaning

A simple idea that can be applied towards eliminating unwanted edges is to eliminate those having size less than a number of pre-specified pixels, i.e. 5. This will clear the image of all the noise edges that look like salt and pepper noise. Although it is very simple as an idea, it yields good results in cleaning many small edges that existed in rather “empty” parts of the image. When referring to “empty” parts, we imply areas of the image where there are few edges, or where the density of the edge pixels is low. An area meeting these criteria in Fig. 3.10 is the upper left part of the image, where only the edges that belong to letters are long and connected. The other edges are very small and are caused by irregularities on the surface of the marble, such as corruption. The algorithm manages to clean that area rather well.
On the other hand, we observe that there are many small edges in the center and lower part of the image which involve more than a few pixels and cannot be cleaned using a small size threshold. Furthermore, some of these edges are almost as long as some of the not-connected parts of the edges that we want to preserve. If we eliminate these noise edges by means of a size threshold, we will also eliminate edges that belong to letters and are not connected, such as the vertical line of T at the first row of letters. This conflict makes the use of this simple cleaning approach quite inefficient and leads to a form of a selective cleaning algorithm.

3.3.3. Selective cleaning

This algorithm forms an extension of the previous cleaning scheme that avoids the problem stated above to a satisfactory degree. In order to eliminate an edge it uses criteria relating not only to the number of pixels, but also to the density of edge pixels in the surrounding area. If this area has few edges the threshold is decreased, whereas the threshold is increased in areas of dense edges. This scheme can eliminate larger edges in areas where we face noise contamination leading to increased edge activity.
The algorithm for selective cleaning scans the image and defines a window for every pixel that belongs to an edge centered at that pixel. Then, the number of edge pixels in this window is measured. The density of edges in this window controls the size threshold applied. If the density is smaller than the lower accepted density then the size threshold is decreased, so that only very small edges are eliminated. If it is higher than a higher accepted density, then the threshold is increased, so that larger edges can

be eliminated from places where the edge density is large. If the density falls in between these two values no change is made to the threshold.
According to the above we apply the algorithm using an initial threshold of 10. In areas of the image that are quite “empty”, this threshold is reduced to 5, so only edges that have less than 5 pixels are eliminated. In areas of the image where there are a lot of edges the threshold is increased to 15 to eliminate longer edges. In all other parts of the image the threshold remains unchanged and edges of size 10 are eliminated. It should be mentioned that all numbers above are set experimentally. The ideal value of the threshold depends heavily on the output image of the edge detector and the threshold used to obtain the binary image. The same apply for the threshold changes. Under the assumption that the size of the letters in the image does not differ drastically and depends only on the actual size of the letters on the inscription and the distance at which the picture was taken, such threshold levels can be easily set.

3.3.4. Edge elimination

This step was developed to face the problem of the existence of very long and randomly spread small edges in the image.
If the length exceeds the maximum expected length of a letter then this edge is obviously caused by a large crack on the surface of the rock, or by corrosion and is certainly not part of a letter. Edges like this are removed from the image without any further examination. The only side effect that can arise is when a letter is placed on a crack. Then, we throw a large edge caused by the crack, but part of this edge could have been devoted to a letter. This is not a rare case, as we have seen in our images, but it is very difficult to overcome it.
Implementing the above we eliminate any edge that is longer than 70 pixels, for example, regardless of their angle. This number is estimated taking in mind that letters are about 50 pixels long and certainly no more than 70 in any direction.

Furthermore, we must eliminate some of the small edges that exist in the image and are mainly noise edges. The criterion we will use in this case is the angle information we have obtained for the edges.

The procedure followed in our approach is the following.
First, the labeling algorithm is applied. The majority of lines that make up the letters are vertical or horizontal. Other regular directions are hard to be defined, since ancient inscriptions were carved by hand and we do not expect regularity of line directions. For example, consider the letter N. The diagonal line can be assumed to be in the vicinity of 135 degrees in printed text. In hand carved characters, though, it can have any angle from about 100 to 150 degrees.
Another conclusion that is easy to make is that noise edges appear in random angles. Noise edges are rather small ones that are caused by the small anomalies on the surface of the rock in any direction. As a result we cannot extract any conclusions for small edges that are not nearly vertical or horizontal.
Vertical and horizontal small edge segments have a large probability of belonging to letters, because of their regular orientation. On the other hand all other edges have a large probability of being noise.

Therefore in the elimination phase we preserve all edges that are horizontal or vertical and we eliminate all other edges if their length is below a threshold, so as to eliminate only very small ones.

3.3.5. Edge linking

Edge linking is the procedure of connecting line segments. To decide which segments should be connected several criteria are used. In the literature we came across criteria such as the gray scale value of the pixels under consideration and their gradient direction. However, these are not applicable to our application since gray levels are very similar in all edges, regardless of whether they belong to letters. The gradient direction criterion is also inappropriate, since it uses only the direction of the two pixels to be connected and not of the entire two edge segments. It is adequate for images where the edges are smooth and the gradient direction of the extreme pixels correspond to the direction of the segment.
This led us to search for other criteria, which would be effective for our application. These criteria are size, proximity and direction of the segments and are structural criteria. In this paragraph we discuss two algorithms. The first utilizes only the criterion of proximity and merely connects edges that are separated by small gaps. The second one is more sophisticated and uses all the mentioned criteria to connect segments that have a large probability of belonging to letters.

First algorithm
A simple step that improves the results is the linking of edges that have gaps up to two pixels between them. It is very common for gaps of 1 or 2 pixels to exist between edges and it is mainly caused by the thresholding that is applied in the binarization process. During the thresholding, no care is taken on whether a pixel lies on an edge. As a result, it is considered as background if its gray level value does not exceed the threshold even when it obviously belongs to an edge. For this reason, some edges appear with small gaps. The edge linking algorithm fills these gaps to obtain larger edges, by looking in the neighborhood of each pixel for other edge pixels. To fill gaps of 1 pixel a neighborhood with radius of 2 is searched; for 2 pixel gaps a radius of 3 is used, etc. If another edge pixel is found in this neighborhood, it is connected to the pixel under consideration. This linking, however, must be performed carefully, especially when the gap is 2 pixels wide. In this case, it is not obvious which pixels should be linked to existing edge pixels. To overcome face this problem we devised the following.
If the gap is only one pixel wide then the solution is easy. Lets assume we want to connect two pixels that have coordinates (x_1,y_1) and (x_2,y_2). It is obvious that the pixel that would fill the gap has the following coordinates:

$$\left(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2}\right)$$

If the division produces numbers that are not integers, i.e. 10.5, the decimal part is discarded.
The filling of two pixel gaps is not as trivial. We calculate the mean values of the x and y coordinates of the pixels to be connected in this case also. At least one of the mean values will not be an integer, since the gap is two pixels wide in one axis. For

example, if the gap is two pixels wide along the x axis, then x_1 and x_2 cannot be both even or both odd, so the mean value will be not an integer.
Let's assume P_1 and P_2 are the two pixels we are looking for to fill the gap. One of these two pixels is calculated the same way as for the one pixel gap case. The other pixel is calculated by adding 0.5 to the coordinate, which was not integer.
To illustrate the above, the possible coordinates of the gap filling pixels are shown below:

$$P_1 : \left(\frac{x_1 + x_2}{2} - 0.5, \frac{y_1 + y_2}{2} \right) \quad P_2 : \left(\frac{x_1 + x_2}{2} + 0.5, \frac{y_1 + y_2}{2} \right)$$

If the mean value of the x coordinates is not an integer.

$$P_1 : \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} - 0.5 \right) \quad P_2 : \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} + 0.5 \right)$$

If the mean value of the y coordinates is not an integer.

For example, if we are to connect (0,3) to (3,5) the procedure is the following, as in Fig. 3.11. The mean values of the x and y coordinates are (0+3)/2=1.5 and (3+5)/2=4 respectively. The mean value of the x coordinate is not an integer. This means that the two pixels we want have coordinates (1,4) and (2,4).

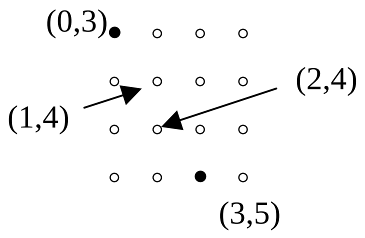


Fig. 3.11. Edge linking example

This algorithm has other side effects. It links any small noise edge to other edges on the basis of proximity, instead of discarding it. This, however, is easy to overcome if a cleaning operation is applied before the linking to eliminate very small edges. Moreover, it connects any two pixels that have a direct gap even if they are connected along other longer paths. This results in a thickening of edges.

Second algorithm

The existence of many broken small line segments along the same direction that belong to the same edge, led us to the development of this algorithm, which connects segments based on all the criteria mentioned, size, direction and proximity.
First of all, labeling is applied so that edges are separated and features are extracted from them. These features include the center of gravity of the edge and the direction of the line that fits the specific edge, as well as its length. The center of gravity and length are relatively easy to calculate; they are the mean values of the x and y coordinates and the diagonal of the bounding box, respectively. To find the bounding box two points are needed; the upper right and the lower left point. The upper right coordinates are the maximum x and y taken over all the pixels that belong to the edge segment, whereas the lower left coordinates are the minimum x and y, respectively.
In order to find the parameters of the line, two different approaches were tested, namely the linear regression [12] and the Hough transform [9].

Linear regression is a numerical method based on least squares approximation. It attempts to find the line, which has minimum distance from all the points that compose the edge. The equation of a line is:

$$y = a * x + b \quad \text{Eq. 3.1}$$

The algorithm returns the slope of the line, a, and the distance from the crossing of the x axis to the origin, b.
Suppose we have N points and x_i and y_i be the coordinates of these points. For each point we form the error measure:

$$s_i = (y_i - a * x_i - b)^2$$

If s_i is zero then the i-th point is on the line, else s_i is a measure of the distance of the point from the line in Eq. 3.1 with the specific parameters a and b. Summing up the above measure for all points we obtain the overall error measure:

$$s = \sum_{i=1}^{N} [y_i - a * x_i - b]^2$$

If all the points lie on this line (a,b), then s is zero. Since we want to find the line that best fits these points we need to find the parameters a and b for which s is minimum. The optimization process results to the parameters values:

$$a = \frac{N * (\sum x_i * y_i) - \sum y_i * \sum x_i}{(N * \sum x_i^2) - (\sum x_i)^2}$$

$$b = \frac{\sum y_i * \sum x_i^2 - (\sum x_i * y_i) * \sum x_i}{(N * \sum x_i^2) - (\sum x_i)^2}$$

This method is used to find the line that fits a number of points that compose an edge label.

The Hough transform is based on an entirely different idea. The basic idea of this technique is to find curves that can be parameterized like straight lines, polynomials, circles, etc., in a suitable parameter space. We use the following representation of a line:

$$x * \cos J + y * \sin J = r$$

To understand how the Hough transform works, let's assume we have N points (x_i,y_i) forming a line and we want to find the equation of this line. Using the above equation and substituting the coordinates of every point we get a sinusoidal equation for ρ and θ on a parameter space (ρ,θ). After repeating this procedure for all the points, we obtain many sinusoidal lines in the parameter space. The crossing of these lines give us (ρ,θ) pairs that fit many equations. The point where the most sinusoidal lines cross gives the parameters of the equation of the line we are looking for.
Since we are working on digital space we must quantize the parameter space into accumulator cells, forming a two dimensional array. The range of ρ is from 0 to the

length of the diagonal of the image and that of θ is from 0 to 360 degrees. For computational reasons, we quantized θ every 2 degrees and ρ every pixel.

The results of linear regression and Hough transform are by no means the same. If we have an L shaped edge, linear regression returns a line with a slope of about 135 degrees, whereas the Hough transform yields a vertical line, the line that passes through most points of our edge.
Linear regression has one major shortcoming. The problem with using the Eq. 3.1. to represent a line is that slope and intercept approach infinity as the line approaches vertical and horizontal direction, respectively. As a result this equation cannot be used as is, because it does not provide a uniform way of dealing with lines. Even if we managed to overcome this problem, the approach used by the Hough transform suits better our purposes. The reason we need these line equations is to compare the slopes and the intercepts of different edge segments in order to decide whether to connect them. The linear regression approach returns the slope and the intercept, while the Hough transform returns the angle and the perpendicular distnce from the origin of the plane. It is far easier to compare the output of the Hough transform between two edge segments, than comparing the slope and intercept resulting from linear regression. For example, nearly vertical lines might have similar slopes, but the difference of their intercepts might be very large. Although they should be connected, if they meet the rest of the criteria, the difference of their intercepts makes that impossible. On the other hand, angle and perpendicular distance are measures that give similar values for similar lines regardless of their values. This makes the comparison fairly easy using the Hough transform.
It is obvious from the above that, Hough transform has certain advantages to be used in our application.
Having calculated the features we need, we organize them making a record for each edge segment. The features stored are the coordinates of the center of gravity, the length and the line fitted on the edge segment, along with the edge label.

After the features have been calculated, the next step is to find which, if any, edge segments must be connected. The criteria that must be met to connect two edge segments are three and are verified in the following order:

- Angle. The angles returned by the Hough transform are compared to determine whether the two segments are parallel or almost parallel.
- Distance from origin. The other parameter that is returned by the Hough transform is the perpendicular distance from the origin. If these distances are equal or very close for the two segments, their centers of gravity lie on the same circle perimeter. Combined with the above angle criterion, if both are met, the lines fitted on the edges are the same or almost the same.

  In the final implementation of the algorithm this criterion is substituted with another very similar but more reliable one. Instead of comparing the distances provided by the line fitting process, we find the perpendicular distance from the center of gravity of one edge to the line fitted to the other. The equation defining this distance from a point P (x_i,y_i) to a line ε: $A*x+B*y+C=0$ is the following:

$$d(P,e)=\frac{|A*x_i+B*y_i+C|}{\sqrt{A^2+B^2}}$$

This method is a little more reliable, because, as we have seen, the Hough transform is not a line fitting algorithm. This distance gives us a better estimation of the actual distance between the two edge segments.

- Distance between edges. If the above criteria are met the two segments lie approximately on the same line. The final check that must be made is how far they are from each other; if they are on opposite sides of the image they should not be connected. The distance from the center of gravity of the first to the second is:
  $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \, .$$
  If this distance is smaller than the sum of the lengths of the segments then they are connected. The sum of the lengths is used as a threshold since if the two edges are far from each other and are short then they are probably noise edges or belong to different letters if their distance is very large. If they are quite long, then we connect them, since they are likely parts of the same letter.
  We must emphasize that all the above apply only for edge segments that are not too long. For example, if we know that the maximum length of a letter is 20 and we find a segment 40 pixels long, then it is obviously caused by corrosion and we do not connect it with any other segment.

Finally, if all the above criteria are met we connect the two segments by drawing a line from the center of gravity of the first to the center of gravity of the second.
A problem exists here when one of the two or both are not line-shaped. Then the center of gravity lies outside the edge, is not an edge pixel, and when the segment is connected the connecting line does not coincide with the edge. This may mean that in a later labeling operation they might not be considered as a single edge. This occasion, though, was rarely met in our work and did not present major problems. A solution to this problem is to connect only segments that are line-shaped and ignore segments that form angles or arcs. In the case of arcs it is possible to connect some, small arcs that have large radii resemble straight lines and can be connected.

4. PROPOSED PROCEDURE

In this chapter we present the actual procedure we developed and applied to our problem. We discuss how all the algorithms described in the previous chapter are used in our overall approach.

The overall algorithm is divided into three major steps, see Fig. 4.1. First of all we preprocess the original image, to facilitate the edge detector phase. Secondly, the edge detection algorithm is applied to the resulting image. The final step is the post processing procedure, which takes the edge strength image and keeps only the information that is important for this application. From chapter 3 we have concluded on the algorithms to be applied in the first two parts. On the contrary the post processing is a combination of previously presented algorithms that will require further analysis.

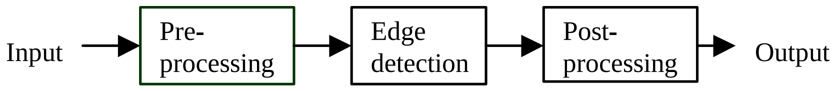


Fig. 4.1. Block diagram of the procedure

The pre-processing stage applies a fairly simple algorithm as discussed in chapter 3.1. Below we will show block diagrams of the edge detection and the post processing phases, which are more complicated.

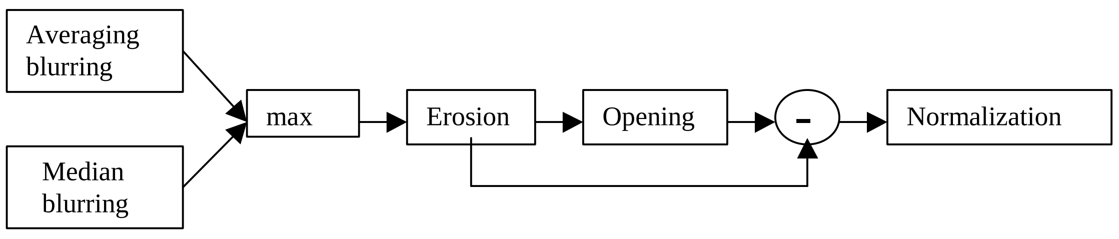


Fig. 4.2. Block diagram of the edge detection algorithm

For further details on the edge detection algorithm see chapter 3.2. A block diagram of the post processing is shown in Fig. 4.3. So far we have only discussed the algorithms we are going to use. Here we will show the sequence of the algorithms and how they interact.

Details on each of the algorithms are presented in the previous chapter and later on in this chapter.

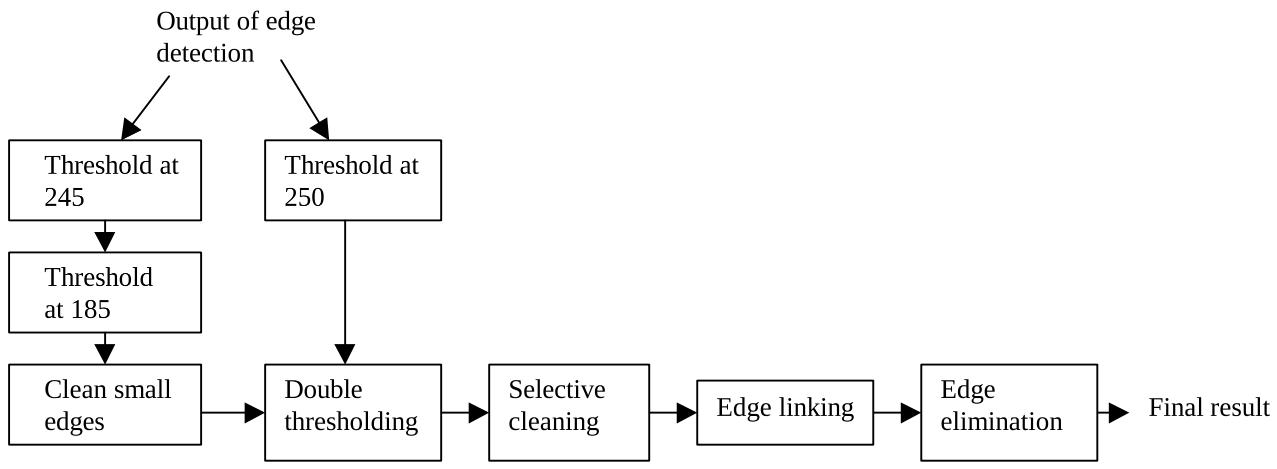


Fig. 4.3. Block diagram of post processing phase

The image shown in Fig. 4.4. is the one that will be used as an input to the entire processing and the effect of each step of the procedure will be illustrated using intermediate results.

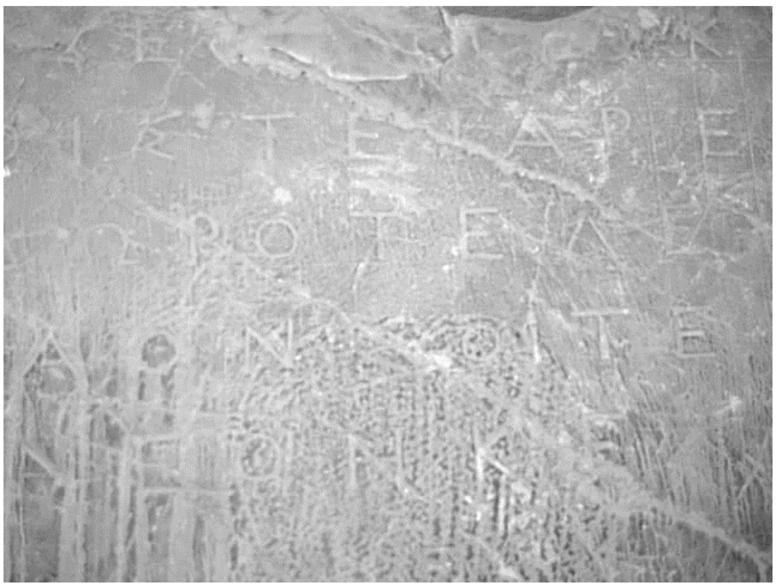


Fig. 4.4. The input image

4.1. Preprocessing

The image equalization algorithm presented in the previous chapter (section 3.1) gives good results in the test images it was applied to. When the above image is fed to this algorithm the result is shown in Fig. 4.5. It is obvious that differences in illumination have been eliminated and the edges have been emphasized.

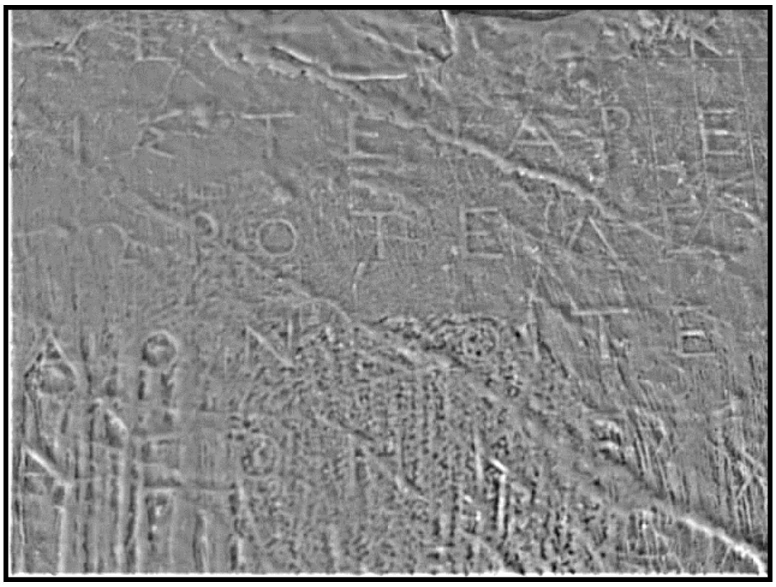


Fig. 4.5. The equalized image

4.2. Edge detection

The algorithm developed in the previous chapter (section 3.2) is used without any modifications. The result is shown in Fig. 4.6.

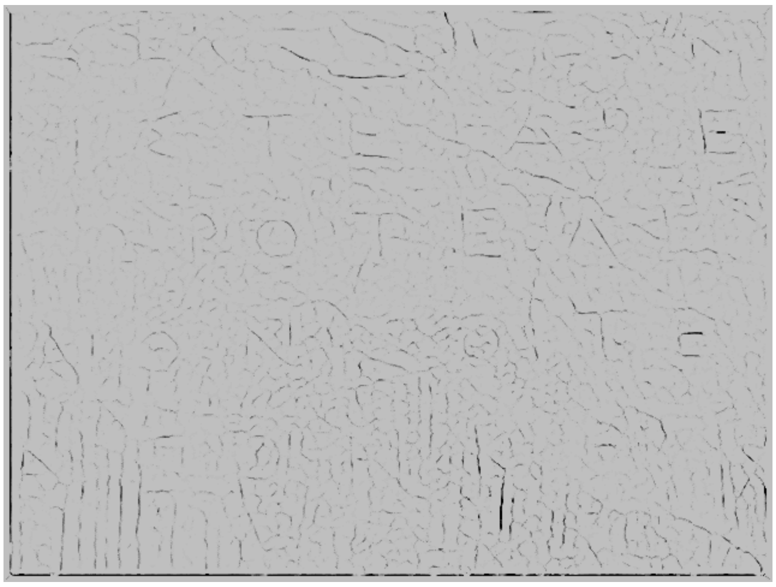


Fig. 4.6. The output of the edge detection algorithm

In order to obtain a complete idea of the edges that are detected in the image above we threshold the image at a level of 254. This thresholding is done purely for purposes of understanding the output of the edge detector and is not a part of the processing. The output is shown in Fig. 4.7.

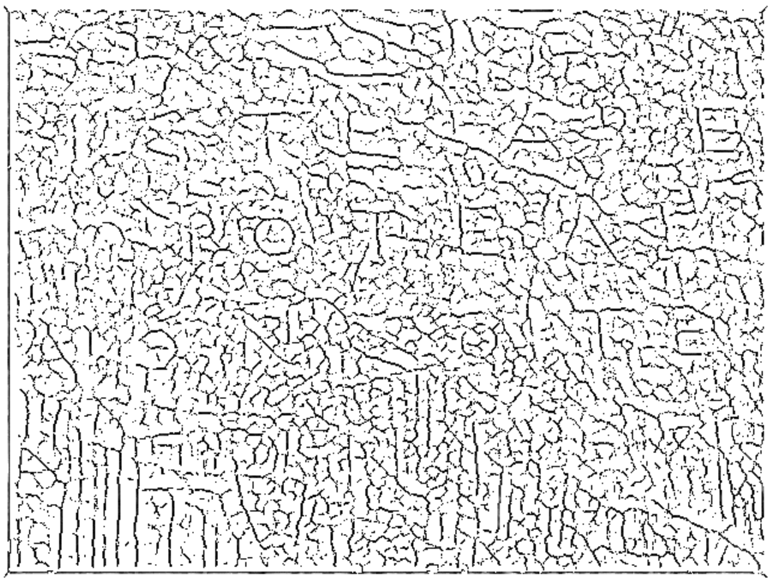


Fig. 4.7. Output of edge detection thresholded at 254

It is obvious from Fig. 4.7 that the edge detector finds many edges, many more than desirable. The post processing phase must eliminate most of these edges that do not belong to letters and to keep the letters intact, as possible.

4.3. Post processing

The post processing that will be applied to the output of the edge detector is a combination of the algorithms described in the previous two chapters.

4.3.1. Double thresholding

A variation of the algorithm discussed in chapter 2.4.3 is developed to meet the requirements of our application. The original algorithm takes one image and thresholds it twice, creating two images and continues using them. Instead of this, we modified the algorithm to take the two thresholded images as input. This way we can do some further processing on them, instead of a simple thresholding.
The determination of the images that are supplied to the double thresholding algorithm is very crucial. Any information that will be lost from now on at every step of the processing will not be recoved. Therefore care must be taken so as to discard only trivial edges.
One of the two images that will be supplied to the algorithm must have few edges and desirably only edges that are parts of letters. This is used as a basis for expanding

edges using the second image. This second image must have a lot of edges and must contain entire letter-edges to preserve all useful information.
The images that we actually use are:

- The first image in Fig. 4.8. is taken using two thresholds and an edge cleaning procedure.
- The second image in Fig. 4.9. is the result of a simple thresholding at 250.

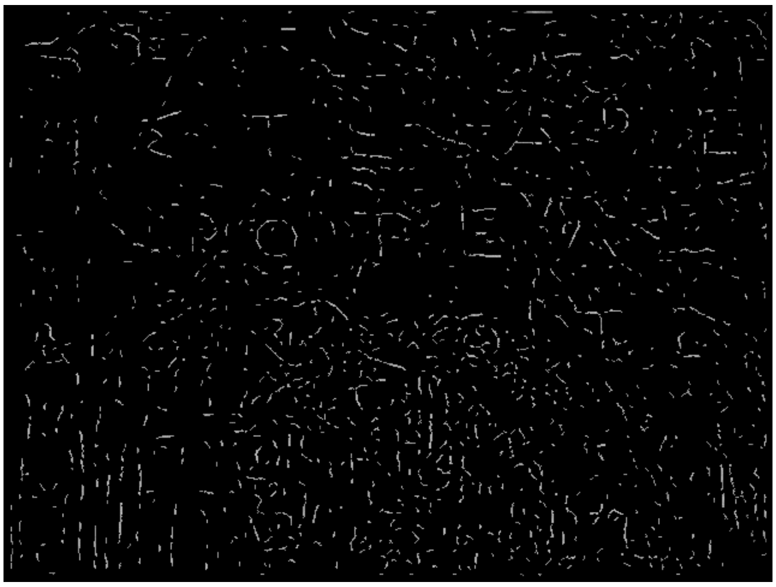


Fig. 4.8. Thresholding the edge strength twice.

The image in Fig. 4.8. was first thresholded at 245 and anything larger than this threshold was set to 0. Every non-zero pixel was then assigned the value it had in the equalized image, so as to keep the brightness information of the original image. Then, the final image was obtained by thresholding, which sets to 0 anything above 185 (in the equalized image). The first threshold is applied to keep only the stronger of the edge pixels after the

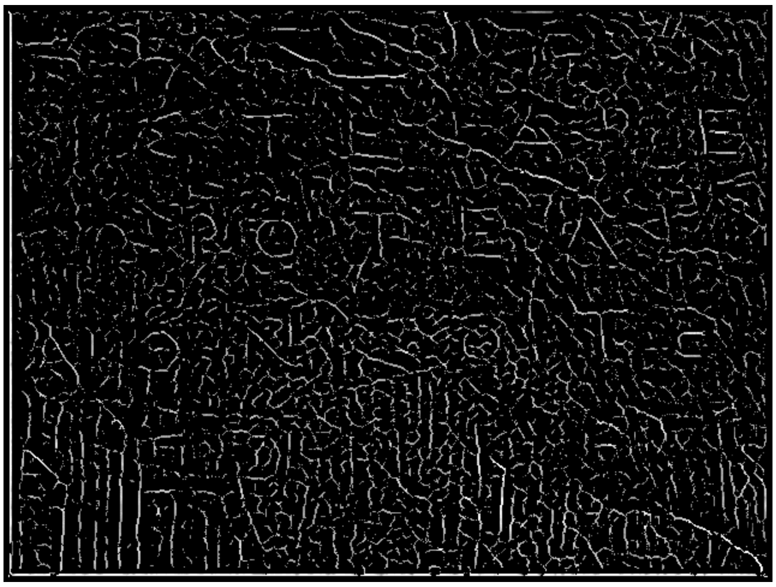


Fig. 4.9. Thresholding edge strength image at 250.

edge detection and the second one to eliminate some very bright pixels that are caused by lighting irregularities in the original image. Finally the edge cleaning algorithm is applied to the output of the two thresholds, to eliminate the edges that are composed of less than 4 pixels.

Having these two intermediate edge images the algorithm continues by scanning the edges on the first image that contains fewer edge pixels. If an edge pixel in this image has an edge neighbor then this new pixel is kept as an edge pixel. If it has none then the neighbors in the second image are searched. If a neighbor in the second image is found then it is added to the first image as an edge pixel. The algorithm repeats the same procedure until no change is made to the first image, which is the final output of the algorithm.

The resulting image contains all edges of Fig. 4.8. extended with parts of the edges from Fig. 4.9.

The output of the double thresholding algorithm is the following image.

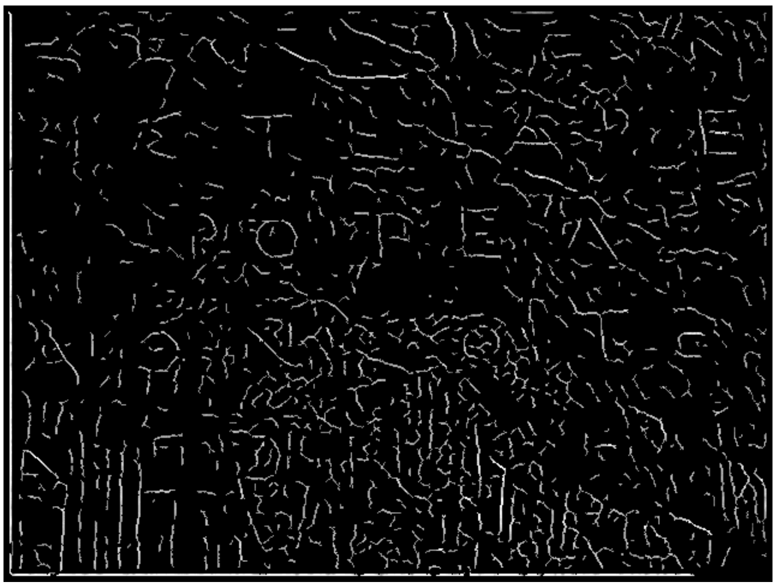


Fig. 4.10. Output of the double thresholding algorithm.

It is easily seen that most of the very small edges that existed in Fig. 4.9. have now disappeared and the large edges have almost remained intact. Some small pats of letter edges have actually been lost, but the whole procedure poses a trade off. Trying to eliminate a large portion of the noise edges, some useful edges are also eliminated.

4.3.2. Selective cleaning

The next step in post processing is the selective cleaning algorithm. The image we have at this stage has most of the letter edges, but also contains many small or larger edges that are caused by cracks or corrosion on the surface of the rock. For example, in the upper central and the lower left parts of the image, some large edges can be distinguished that are definitely not caused by the carved letters. Furthermore, many small edges are spread all over the image. The problem with those small edges is that in some parts of the image they are really quite small, while in other parts their length is comparable to that of the letter edges. This is the problem we attempt to overcome by using the selective cleaning. The result of applying this algorithm to the output image of the previous step is shown in Fig. 4.11.

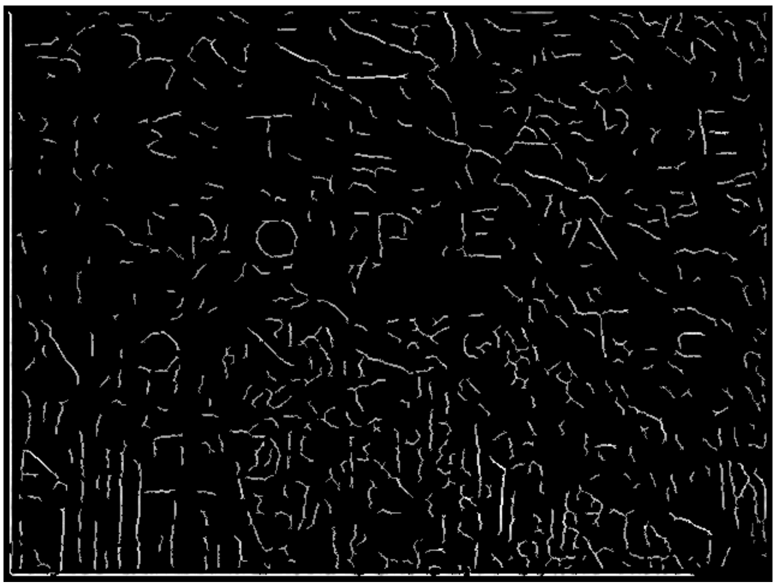


Fig. 4.11. Output of selective cleaning algorithm

The result of this algorithm is exceptionally good if we consider the input image. It is obvious that it has removed almost all small edges. The noise edges that remain are a big problem because they cannot be classified as different than the letters edges. Indeed, the remaining edges have similar characteristics and it is impossible to find some way of clearly determining which are noise and which are letter edges.
This algorithm, although it works very well, has a major shortcoming. It uses many parameters that have to be estimated. More specifically, it needs a threshold, the variation of this threshold and two limits for the edge density of the image. The threshold is the minimum number of pixels each edge must have in order to be preserved. The variation of the threshold is the value that is added to or subtracted from the threshold depending on the edge density on every part of the image. Finally, the limits for the density are the upper limit over which the threshold is increased and the lower limit under which the threshold is reduced. Assuming that the original images used do not vary a lot and have similar letter sizes, then it is possible to find values for these parameters that can work optimally for a wide variety of images.

4.3.3. Edge linking

At this stage we attempt to connect some edges that probably belong to single larger edges. Applying the algorithm that was described in chapter 3.3.5 (second algorithm) gives the result shown in Fig. 4.12.

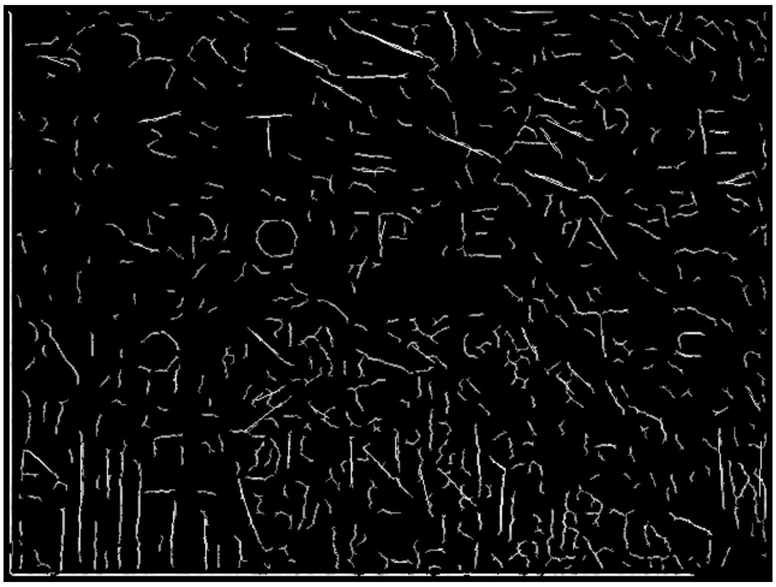


Fig. 4.12. Output of the edge connection algorithm

The effect of the algorithm is clearly illustrated in this image. Many segments that were parts of larger edges have been connected, building up longer edges. It is possible now to eliminate some of them based on their length. Very long edges that were broken into smaller parts by the thresholding are connected again and can be now detected and discarded. Furthermore, edges that belong to letters are connected also so that they will not be treated as small noise edges.

4.3.4. Edge elimination

The final step in this process is the edge elimination described in chapter 3.3.4.

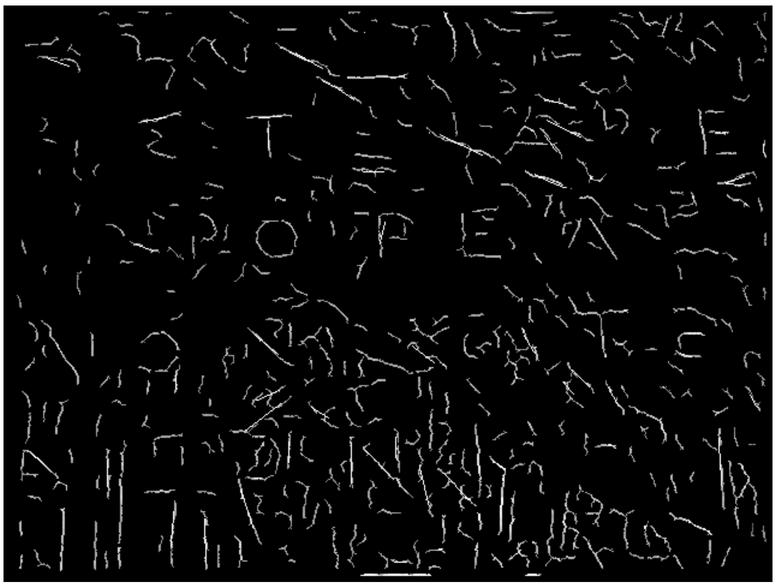


Fig. 4.13. Output of the edge elimination algorithm

The result of this algorithm to the image in Fig. 4.12. is shown in Fig. 4.13. All edges that were longer than the maximum length of a letter are no longer present in this image.
This is the best output we have managed to achieve. The noise is obviously not completely removed. The reason is that the remaining edges all have similar characteristics to useful letter-edges and there is no way to separare noise from letters. Image processing can get only this far in this application. In order to get a clear description of the letters some kind of recognition should be applied,

4.4. Other Examples

The procedure, described in this chapter, yielded good results not only for the image used as input in the previous sections, but for a variety of images. The only restrictions are that the letters in the images have roughly the same size. This is important because in the course of the processing we assume that edges larger than a certain threshold cannot be parts of letters due to their large size. If an image, which has very large letters, is used as input the parameters of the algorithms may require a slight modification.

In Fig.4.14. we see an image from a different region of the same inscription as the image previously used. It suits our purposes, since it was taken from almost the same distance as the previous, thus having same letter sizes. We must emphasize here that our algorithms are expected to work equally as well in any image of an inscription, provided the letter sizes do not change significantly.

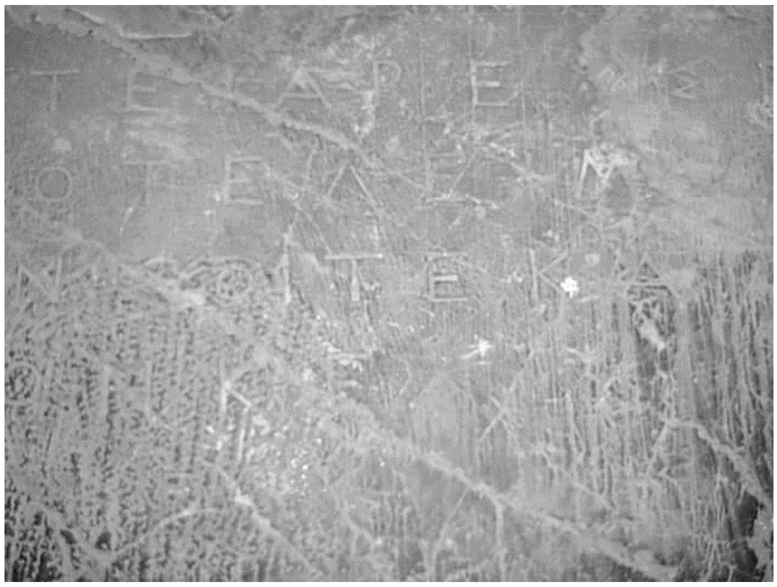


Fig. 4.14. Another image taken from the inscription

Applying the procedure discussed in this chapter we obtain the result shown in Fig. 4.15.

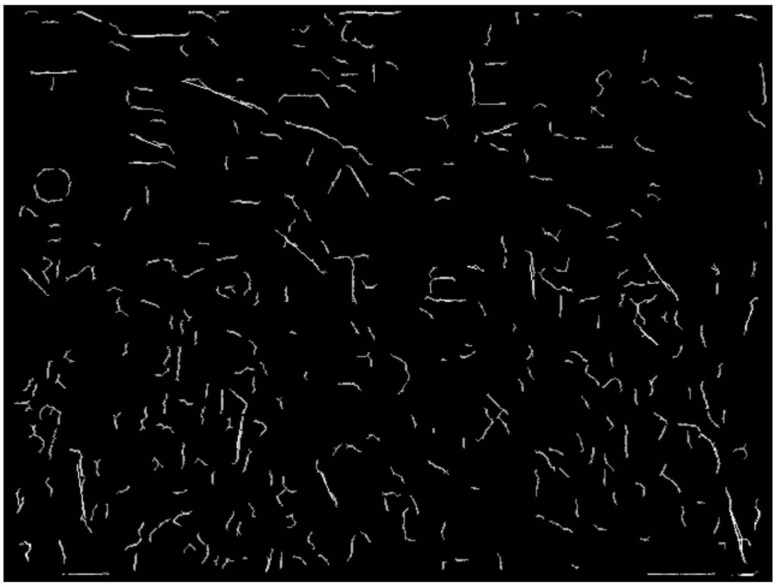


Fig. 4.15. Output of the entire procedure

We can that in the Fig. 4.13. most of the edges belonging to letters are preserved and most of the edges due to noise have been eliminated. Some noise edges that still exist in the image cannot be eliminated because they have very similar characteristics to letter edges.

5. CONCLUSION AND FUTURE WORK

The recognition of objects in a complex background is generally a complicated task. Various algorithms have been developed to achieve such a task. In their majority, the algorithms are Pattern Recognition algorithms that try to match exact or almost exact occurrences of the object to be recognized. These algorithms generally require a uniform background, as in OCR algorithms. Noisy background increases the difficulty of the entire problem and makes the PR algorithms much more complicated. Furthermore, lack of knowledge of the objects to be detected can cause major problems. In this case the classical PR algorithms cannot be applied since they require good knowledge of the characteristics of the object to be found.

In the course of this work we tried to extract the letters from the noisy background of ancient inscriptions. We focused on image processing, since our goal was merely to make these inscriptions readable and not to recognize the letters carved on them. This is the first step before any pattern recognition can be applied. Our work was further complicated, because we applied no special characteristics that could discriminate the letters from the anomalies on the surface of the rock or marble.

Another difficulty we faced was the lack of an adequate edge detection algorithm in the literature. We implemented a number of existing algorithms, but only one of them yielded a somewhat promising result, the ATM algorithm. The images we process have very faint edges and most of the existing algorithms could not detect them. This led us to the modification of the ATM algorithm. To further improve the result of the edge detection algorithm we pre-processed the image, to remove lighting anomalies and enhance the contrast of the image.
After the edge detection phase, post processing had to be applied in order to preserve only the edges belonging to letters and not those caused by anomalies on the surface of the rock. Few algorithms were found that could be effective in our application. One of them was used almost unmodified, the double thresholding algorithm, but for the rest of the post processing we had to develop algorithms such as the edge linking or the selective cleaning algorithms to better suit our application.

As we have mentioned in the first chapter this work is just the first attempt in building an automated system for reading ancient inscriptions. Further work to extend this thesis can be separated in two categories. First, the processing might be continued with pattern recognition applied to the images. Secondly an image acquisition system should be developed in order to obtain better pictures from the inscriptions than the ones we already have.

Processing & pattern recognition

In this work we have tried to extract the letters from the noisy background of the rock region. The result of our work is an image with uniform background that contains a good portion of the letters. This means that, although we have lost parts of the letters, the larger part of the information is still available, so that a pattern recognizer can be applied. As we have seen in the first chapter, the problem of pattern recognition, especially that of optical character recognition, is extensively researched.

Bourbakis and Goldman [2] have developed a pattern recognizer that has a feature very useful in our application. It can handle lines that are not smooth, lines with unevenness. This will be effective in our work since in our results most of the lines are not smooth and other PR algorithms will have problems recognizing them as straight lines. However, the recognition part of the algorithm uses a graph-based method, which will probably give poor results, due to the lack of information in our image, the letters are broken or parts are missing.
Artificial Intelligence is nowadays heavily used to confront these problems. Methods such as Hidden Markov Models, Artificial Neural Networks and Fuzzy Logic have been utilized lately.
For example, Lazzerini and Marcelloni [13] have developed a fuzzy system for recognizing handwritten characters. They describe the shape of each character by linguistic expressions derived from a fuzzy partition of the character image. Then this linguistic description is compared with others in a database.

The problem of broken characters might yet have another solution. Wang and Yan [14] have proposed a macrostructure analysis method for mending broken handwriting. This can prove very useful, because most of the letters in our output image are either broken or parts of them are missing. This algorithm does not recognize the characters, it simply connects the broken ones, thus making the recognition task easier.

This work even if it is done, will not be totally successful without the development of an adequate lighting and image acquisition system.

Image acquisition system

Such a system is necessary for two reasons. First of all, the images captured under controlled conditions will be better than the ones we had using a commercial digital camera. Lighting conditions, which are very important, the entire preprocessing step deals with them, should be controlled in order to acquire images that will be uniformly illuminated. This will make the edge detector produce better results. Secondly, from a brief discussion with the archaeologists we have found out that the major contribution of a system like the one we are beginning to develop would be to help in reading some inscriptions that are not readable now. The archaeologists have noticed that when lighting these inscriptions from different angles they can distinguish some parts of their carvings.
A possible solution for an automated system would be to get several captures of one inscription in different lighting conditions, for example lighting the inscriptions from various angles. Having many images and the information spread in these images we can apply the algorithms that were developed in this work to get some first experimental results.

6. REFERENCES

[1] W.L. Hwang, F.Chang. “Character extraction from documents using wavelet transforms”
Image and Vision Computing, vol. 16, 1998, pp. 307-315

[2] Nikolaos G. Bourbakis and David Goldman. “Recognition of line segments with unevenness used in OCR and fingerprints”
Engineering Applications of Artificial Intelligence, vol. 12, 1999, pp. 273-279

[3] S.K. Kim, S.M. Park, J.K. Lee and H.J. Kim. “On-line recognition of Korean characters using ART neural network and hidden Markov model”
Journal of Systems Architecture, vol. 44, 1998,pp. 971-984

[4] James S.J. Lee, Robert M. Haralick and Linda G. Shapiro. “Morphologic Edge Detection”
IEEE Journal of Robotics and Automation, vol. RA-3, no. 2, April 1987, pp.142

[5] Jean Serra. ”Image Analysis and Mathematical Morphology” Volume 1
Academic Press, 1982

[6] Edward R. Dougherty. “An Introduction to Morphological Image Processing”
SPIE Press, 1992

[7] X. Yu, S. Beucher, and M. Bilodeau. “Road tracking, lane segmentation and obstacle recognition by mathematical morphology”

[8] S. Krishnamurthy, S.S. Iyengar, R.J. Holyer and M. Lybanon. “ Histogram – based morphological edge detection”
IEEE Transactions on Geoscience and Remote Sensing,v32, n4, July 1994, pp. 759 – 767

[9] Rafael C. Gonzalez and Richard E. Woods. ”Digital Image Processing”
Addison-Wesley, 1992

[10] A.Rosenfeld and A. C. Kak. “Digital Picture Processing”

Second Edition, Academic Press, 1982

[11] David Vernon. "Machine Vision, Automated Visual Inspection and Robot Vision"
Prentice Hall, 1991

[12] W.H. Press, S.A. Teukolsky, W.T. Vettering and B.P. Flannery. "Numerical recipes in C. The art of scientific computing"
Second Edition
Cambridge, 1985

[13] B. Lazzerini, F. Marcelloni. "A linguistic fuzzy recognizer of off-line handwritten characters"
Pattern Recognition Letters, vol. 21, 2000, pp. 319-327

[14] J. Wang, H. Yan. "Mending broken handwriting with a macrostructure analysis method to improve recognition"
Pattern Recognition Letters, vol. 20, 1999, pp. 855-864

[15] R.M. Haralick, S.R. Sternberg, X. Zhuang. "Image analysis using mathematical morphology"
IEEE Transactions on Pattern Anal. and Machine Intelligence, vol. PAMI-9, no. 4, July 1987, pp. 532-550