University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master in Computer Sciences	Spring semester, 2010 Open / Restricted access
Writer: Min Wang	…… (Writer's signature)
Faculty supervisor: Professor Kjersti Engan; Associate professor Tom Ryen External supervisor(s):	
Title of thesis: Detection and analysis of rock cracks in meteor crater	
Credits (ECTS): 30	
Key words: Ritland crater; Rock crack; image process; Linear regression.	Pages: …… +enclosure: …… Stavanger, …….. Date/year

Acknowledgements

I would like to express my gratitude to my supervisor Kjersti Engan and Tom Ryen for their support and guidance on my master thesis. They have walked me through every stages of this thesis.

I feel grateful to all the teachers in the Department of Electrical and Computer Engineering in the University of Stavanger. I have learnt a lot from them during these two years.

At last I should extend my appreciation to my family and friends for their love and support.

Min Wang

June 10, 2010

Stavanger

Abstract

In 2000, a geologist Fridtjof Riis discovered a meteor crater in Ritland, Hjelmelan municipality in Rogland. This crater was formed by meteorite impact. The crater has areas with a lot of cracks in the rocks, and geologists think these cracks are very valuable information for them. By making photos with an ordinary camera, they want to get binary pictures where the cracks are shown as white lines on a black background. They can measure and quantify the length and direction of the cracks on the binary image.

The objective of this project is to extract cracks from the digital photos. In this project, statistical technique – linear regression, and image processing techniques like spatial filtering, morphological operations have been used.

Binary images with white lines showing cracks on black background are generated as result of this project. Geologists can use this result to analysis the cracks.

Index of Contents

Index of figures

Chapter 1. Introduction

1.1 Background

In 2000, the geologist Fridtjof Riis discovered a meteor crater (See figure 1.1) in the community of Hjelmeland, county of Rogaland (West Norway). The crater is about 2.5 km in diameter, about 350 m deep and was formed probably between 500 and 600 million years ago [1]. It was later buried by sediments, of which it has been partly recovered.

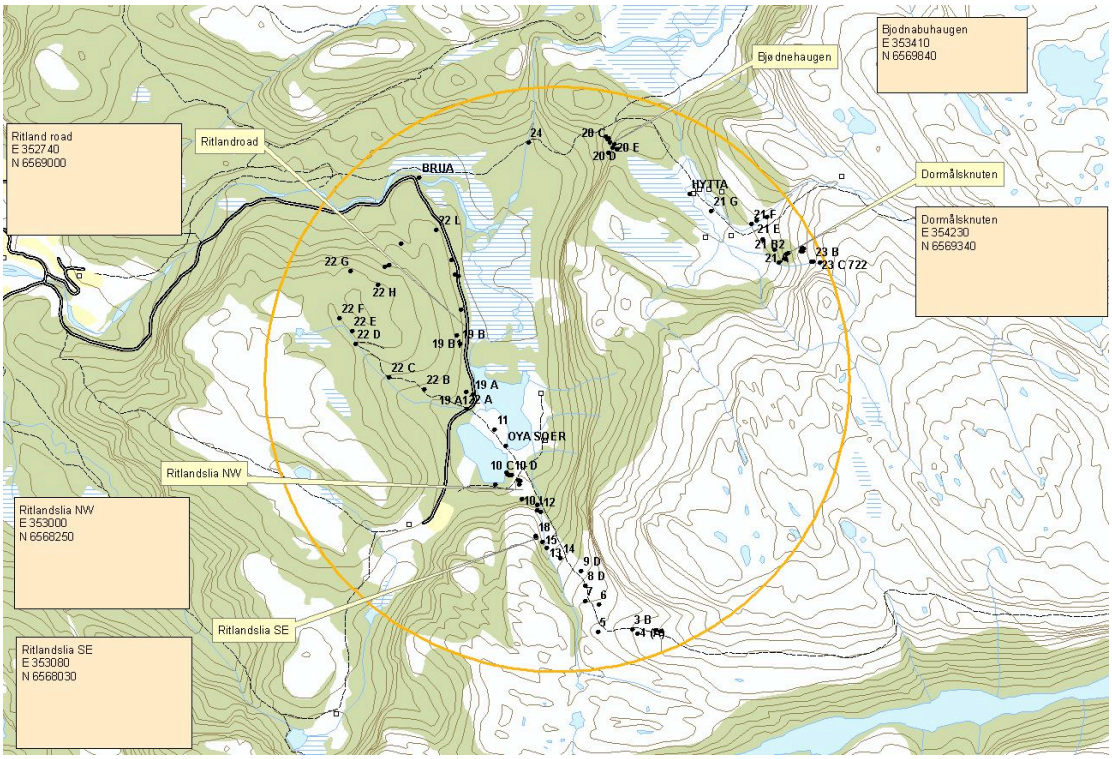


Figure1. 1 Map overview of the Ritland crater [2]

(The yellow line indicates approximately outline of the crater)

Fridtjof Riis started a three years' Ritland project, which is supported by the Norwegian Research Council, for analyzing the crater in 1. July 2009. Until now, the geologists have proved that the crater is formed by meteor cretaceous impact, and they have collected many rock samples [3]. Thousands of meteorites have hit the earth during the 4.5 billion years. So far, only 176 of these have been identified as impact crater. There are in-depth studies in the next steps by the geologists for the Ritland crater, like reconstruction when the crater was formed. Studying the cracks in the rock formations is of great importance for the geologists, and in the current project. They want to get the information of the cracks automatically with aids of computer science.

1.2 Purpose and importance

The crater has areas with a lot of cracks in the rocks (See figure 1.2). The geologists believe that the rock cracks are very important and valuable for the following research. They want to measure and quantify directions and lengths of the cracks, and get valuable data for their analysis. Some of the cracks were formed by enormous pressure, which occurs only when meteor cretaceous impact, meanwhile some of them are formed in the following long period, because of the earth's moving. So the cracks' information is very useful for the reconstruction. For this purpose, by making photos with an ordinary camera, they wish to remove the grass and other things recovering on the rocks and to get binary pictures where the cracks are shown as white lines on a black background.

Figure1. 2 Rock cracks caused by meteorite fall

The focus of this project is to use image-processing techniques like edge detection, filtering, morphology, etc., in order to find cracks in the pictures of rocks. (See figure 1.2). The plants and rubble covering on the rocks are defined as "noise" in this case. The object of this project is to remove these noises and generate binary images showing the cracks as white lines on a black background. And then, the cracks' lengths and directions are automatically measured and quantified. In this project, Matlab image processing toolbox will be used.

1.3 Related research

Line detection is a classical subject in image processing field with many applications, such as vessel detection in medical image, and road detection in remote images [6].

Applying line detection can reduce the amount data to be processed and can filter out information that may be regarded as less relevant, meanwhile preserving the important structural properties of an image. Some traditional way to detect lines, such as Canny algorithm, is based on abrupt changes in intensity.

Hough transform (a special case of the Radon transform) has been widely used to detect straight lines. The basis of this method is: a fixed point (xi, yi) in the xy-plane on a straight line $y = kx + b$ is a single line in the parameter space (kb-plane). This method is always used to determine the location and orientation of straight lines in image. [7,8]

Wavelet transform can also be used in edge detection. Lines in image are mathematically defined as local singularities. Wavelet analysis is a local analysis; it is suitable for time-frequency analysis, which is essential for singularity detection. [9, 10]

Recently, grid cell analysis (GCA) has been used for inspection of the cracks on asphalt surface image and pavement image. In this way, a pavement image was divide into gird cell of 8 × 8 pixels and each grid cell was classified as a crack or non-crack cell using the gray-scale information of the border pixels. [11 - 13]

It is hard to obtain the ideal lines and edges from real life images. In different case, we have to choose a different suitable way to deal with that. All the lines/ cracks detection methods, as we mentioned above, are for some specific applications, but not suitable for our project. In this project, we find different ways. All the approaches are based on analysis the different features between lines and other information in the image.

1.4 Preproject

There is a project started from September 2009 as a preproject to this thesis. We want to generate a binary image with white lines showing cracks on a black background. So in the preproject, we focus on how to remove useless information and get the skeletons of the cracks. The skeleton of the cracks has been obtained successfully last year. (See figure 1.3). This is the input for this thesis.

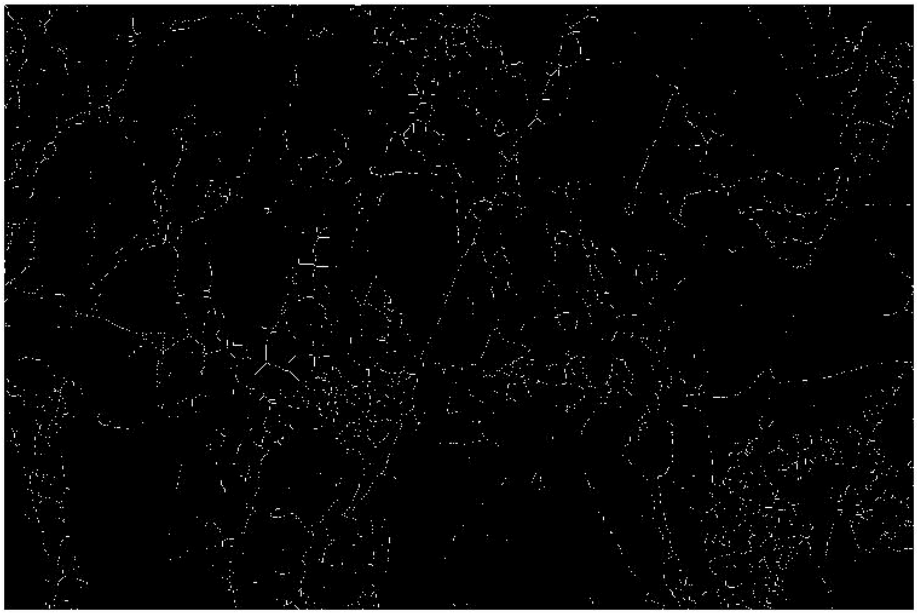


Figure1. 3 Result from previous work, skeleton of cracks.

The whole process in the pre-project is: (See figure 1.4)

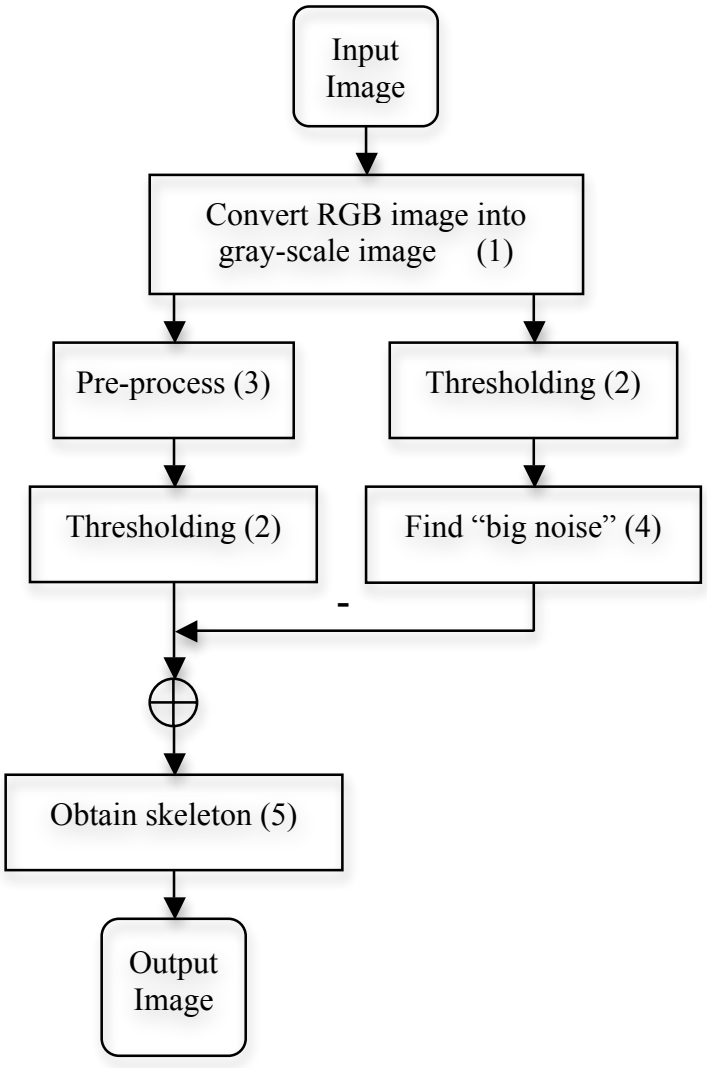


Figure1. 4 Workflow of previous work

(1) Convert the color image to a grayscale image. We want to get a binary image as result, and color factor is very complex. So it is ignored at first.

(2) Convert grayscale image (see figure 1.5) into binary image. The simplest method to extract cracks from original image is thresholding. In this way, we have to choose a probable threshold value by analysis the intensity histogram (see figure 1.6) of the input grayscale image. In the grayscale image (figure 1.4), we can find that the intensity values of the pixels, which are represented cracks, must be very low. (In 8 bits intensity image there are 256 intensity levels. 0 is usually associated with completely black and 255with perfect white). The intensity levels of the pixels, which represent the rocks must be higher. There is a minimal value in the histogram at intensity value T (T is 41 in figure 1.6, and T is different for every images). Choosing value T as threshold value, we can extract the cracks in the image.

(3) Apply morphological grayscale opening operation as a preprocessing step. After thresholding, some very slight cracks will disappear, and there is much noise in the binary image. So we have to apply preprocessing before thresholding to keep more cracks' detail and denoise of the objects. For this purpose, we found that opening of the morphological gray-scale image is the best method.

(4) Find "big noise". The plants, which cover parts of the rocks, and the rubbles are unwelcomed information. And most of this kind of information are larger than the information we want (cracks), so we call them "big noise", which should be removed. Observing the "big noise", we can find that the cracks are thinner and longer objects, and there are less pixels used to represented them. So we define the objects n as "big noise" if

$$\begin{cases} Object\,n.Area > \tau_1 \\ \dfrac{Object\,n.MajorAxisLength}{Object\,n.MinorAxisLength} < \tau_2 \end{cases}$$

Where, $MajorAxisLength$ and $MajorAxisLength$ are properties of objects in the image. $MajorAxisLength$ is the length (in pixels) of the major axis of the ellipse, and $MajorAxisLength$ is the length (in pixels) of the minor axis of the ellipse.

(5) Eliminate the "big noise" from the preprocessed image and obtain the skeleton of cracks. The geologists want to measure and quantify the length and direction of the cracks, and the width of the cracks can be ignored. So we

represent the cracks by one pixel width. This can be implemented by morphological operation called skeletonization.

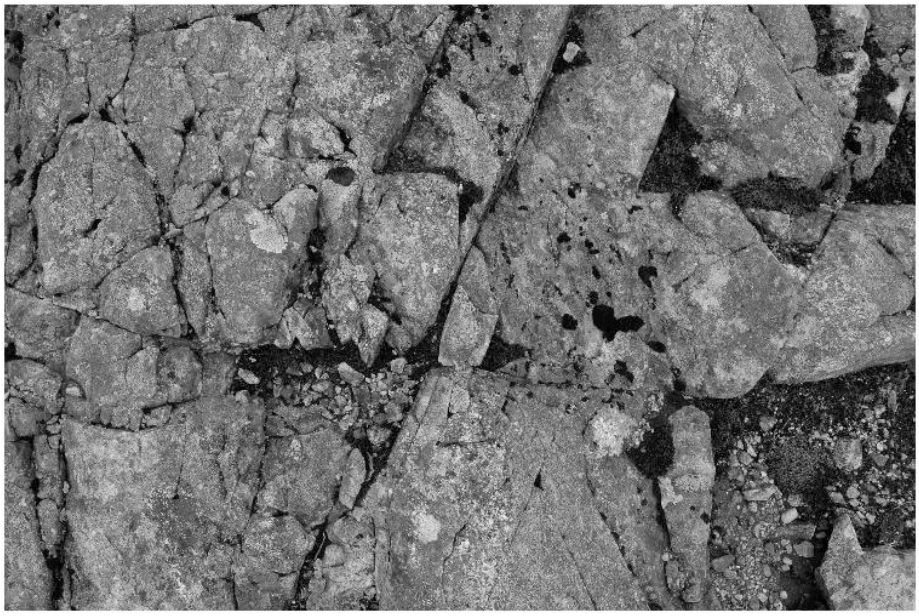


Figure1. 5 Grayscale image

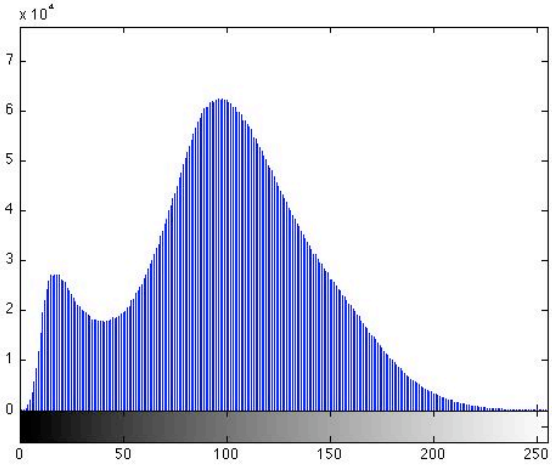


Figure1. 6 Histogram of figure 1.5

1.5 Thesis outline

This report focus on designing some algorithms to represent the rock cracks by straight vectors. The input of this project is the image with skeleton of the cracks, which is the output of previous work. In that image, one crack is represented by several disconnected lines, and many braches on the crack lines. We want the output of this project to be an image with straight white lines representing cracks on black

background. In the report, algorithm design and implementation will be discussed in detail. The project is implemented in Matlab. The following chapters constitute the report:

Chapter1 gives a background of the Ritland crater project, and presents the purpose and importance of this project. Then, we also look at the earlier research about how to get the skeleton of the cracks.

Chapter 2 gives an introduction to the background theory of this project, involving image processing and statistics. This basal knowledge has been used in the algorithm design.

Chapter 3 presents the details of algorithm design for cutting the short branches and representing cracks by straight vector. This chapter explains why and how some methods are used for in these algorithms.

Chapter 4 gives a brief introduction of our tool—Matlab, and describes how to implement the project by Matlab in detail, including user interface and algorithm implementation.

Chapter 5 & **Chapter 6** summarize the major contributions and conclusions of our work. We give some results from our work and point out the direction for further study.

Chapter 2. Background theory

Digital image processing refers to processing digital images by computer to extract information from an image. In this chapter, the background theories on the image process themes that will be used in later chapters are described.

2.1 Basic relationship between pixels [16, 17]

For a given pixel p at coordinate (m,n), it has four horizontal and vertical neighbors whose coordinates are given by

$$(m-1,n),(m+1,n),(m,n-1),(m,n+1)$$

This set of pixels is called 4-neighborhood of p, which can be denoted as $N_4(p)$

Meanwhile, it has four diagonal neighbors of p, whose coordinates are given by

$$(m-1,n-1),(m-1,n+1),(m+1,n-1),(m+1,n+1)$$

The diagonal neighbors are denoted by $N_D(p)$.

The 8-neighborhood of p, denoted as $N_8(p)$, is the both 4-neighbors and diagonal neighbors.

$$N_8(p) = N_4(p) + N_D(p)$$

In the binary image, we define two kinds of adjacency:

4-adjacency: Two pixels p and q with value 1 are 4-adjacency, if q is in the set $N_4(p)$.

8-adjacency: Two pixels p and q with value 1 are 8-adjacency, if q is in the set $N_8(p)$.

If S is a subset in the image,

4-connected component: The subset S is defined as 4-connected component, if each pixel has 4-adjacency pixel in S.

8-connected component: The subset S is defined as 8-connected component, if each pixel has 8-adjacency pixel in S.

For example, there is a binary image as shown in figure 2.1. Each square denotes a pixel, and values in squares with dots are logical 1. There are three 4-connected components in the image. And there is only one 8-connected component.

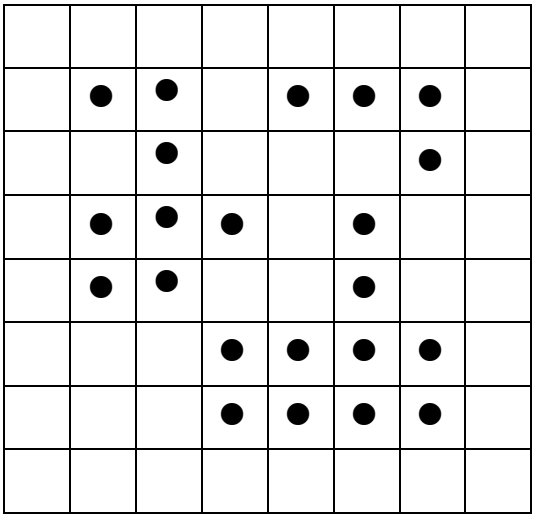


Figure2. 1 An example for connected component

2.2 Linear spatial filtering

Spatial filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood, and whose value is the result of the filtering operation. There are two kinds of filtering operation: *correlation* or *convolution.* In these two operations, the value of output pixel is computed as a weighted sum of neighboring pixels. Correlation is the process of moving a filter mask over the image and computing the sum of products at each location. The mechanics of convolution are the same, except that the filter is rotated 180^o firstly.

The output of the filtering operation of a filter mask $h(m,n)$ with an image $f(m,n)$ is $g(m,n)$ (See figure 2.2).

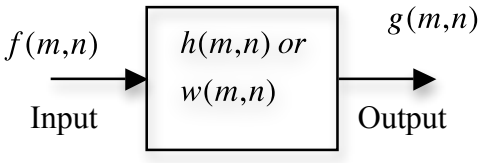


Figure2. 2 Overview of spatial filtering

Convolution:

$$g(m,n) = f(m,n) * h(m,n)$$
$$= \sum_{i=-I}^{I} \bullet \sum_{j=-J}^{J} h(i,j) \bullet f(m-i, n-j)$$

Correlation: $w(i,j) = h(-i,-j)$

$$g(m,n) = f(m,n) \circ h(m,n)$$
$$= \sum_{i=-I}^{I} \bullet \sum_{j=-J}^{J} w(i,j) \bullet f(m+i, n+j)$$

2.3 Morphological processing [16 - 18]

Morphology usually denotes a branch of biology that deals with the form, structure and configuration of animals and plants. Mathematical morphology is set theory-based methods of image analysis and plays an important role in many digital image processing algorithms and applications. In image processing field, mathematical morphology can be used as a tool for extracting image components that are useful in the representation and description of object shape, such as skeletons and boundaries.

Binary (logical) image has only two values for each pixel. Our project interprets the logical value) as black and 1 as white.

2.3.1 Sets theory

Objects in an image are defined as sets. So we introduce some basic operation firstly. In binary image, the sets are elements of the 2-D integer space Z^2.

A is a set composed of *ordered pairs*, if $a = (a_1, a_2)$ is an element of the set A. That is denoted:

$$a \in A$$

If a is not *a* element of set A, this can be denoted:

$$a \notin A$$

Null set or *empty* set is denoted as: $\varnothing$

If every element in set A is also an element in set B, we say set A is a *subset* of set B, denoted as:

$$A \subseteq B, \quad or \quad A \subset B$$

The *union* of two sets A and B is:

$$A \cup B = \{a \mid a \in A \text{ or } a \in B\}$$

The output pixel of union in binary image is white if either of the corresponding input pixels is white.

The *intersection* of two sets A and B is:

$$A \cap B = \{a \mid a \in A \text{ and } a \in B\}$$

The output pixel of intersection in binary image is white if both of the corresponding input pixels are white.

If every element in set A is not an element in set B, we say these two sets are *disjoint* or *mutually exclusive*, denoted as:

$$A \cap B = \varnothing$$

The *complement* of a set A is a set of elements that are not in A:

$$A^C = \{w \mid w \notin A\}$$

The *difference* of two sets A and B is:

$$A - B = \{w \mid w \in A, w \notin B\}$$

Let B denote a set in a 2-D integer space Z^2 (binary image). The *reflection* of a set B is given by

$$\hat{B} = \{w \mid w = -b, for\ b \in B\}$$

If there is a point with coordinate (x, y) in the object B in a binary image, will be replaced by (-x, -y) in $\hat{B}$ (Rotating 180^o around the origin).

The translation of B by the distance $z = (z_1, z_2)$ is:

$$(B)_z = \{c \mid c = b + z, for\ b \in B\}$$

2.3.2 Basic morphological operation

There is an important term in morphology—*structuring element* (SE): small sets or sub images used to probe an image under study for properties of interest. A structuring element is a matrix consisting of only 0's and 1's that can be any arbitrary shape and size. There is an origin for each structuring element. For different applications, SE should be designed in different shapes and size.

There are two kinds of basic morphological operation: erosion and dilation.

A and B are sets in Z^2, the erosion image A by structuring element B is defined as:

$$A \ominus B = \{z \mid (B)_z \subseteq A\} \text{ or}$$

$$= \{z \mid (B)_z \cap A^c = \varnothing\}$$

This operation can used to shin and shrink objects in binary image.

Dilation operation can be used to make objects thick in binary image, defined as:

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\} \quad \text{or}$$
$$= \{z \mid [(\hat{B})_z \cap A \subseteq A]\}$$

The dilation operation is always used to bridge gaps in image processing.

Now, we give an example for erosion and dilation operation. There is a logical image A as shown as figure 2.3(a). Each square denotes a pixel, and values in squares with dots are logical 1. Figure 2.3(b) shows a structuring element B. The origin is marked as red dot. The result of erosion of A by B is shown in figure 2.3(c). The figure 2.3(d) is the result of dilation A by B.

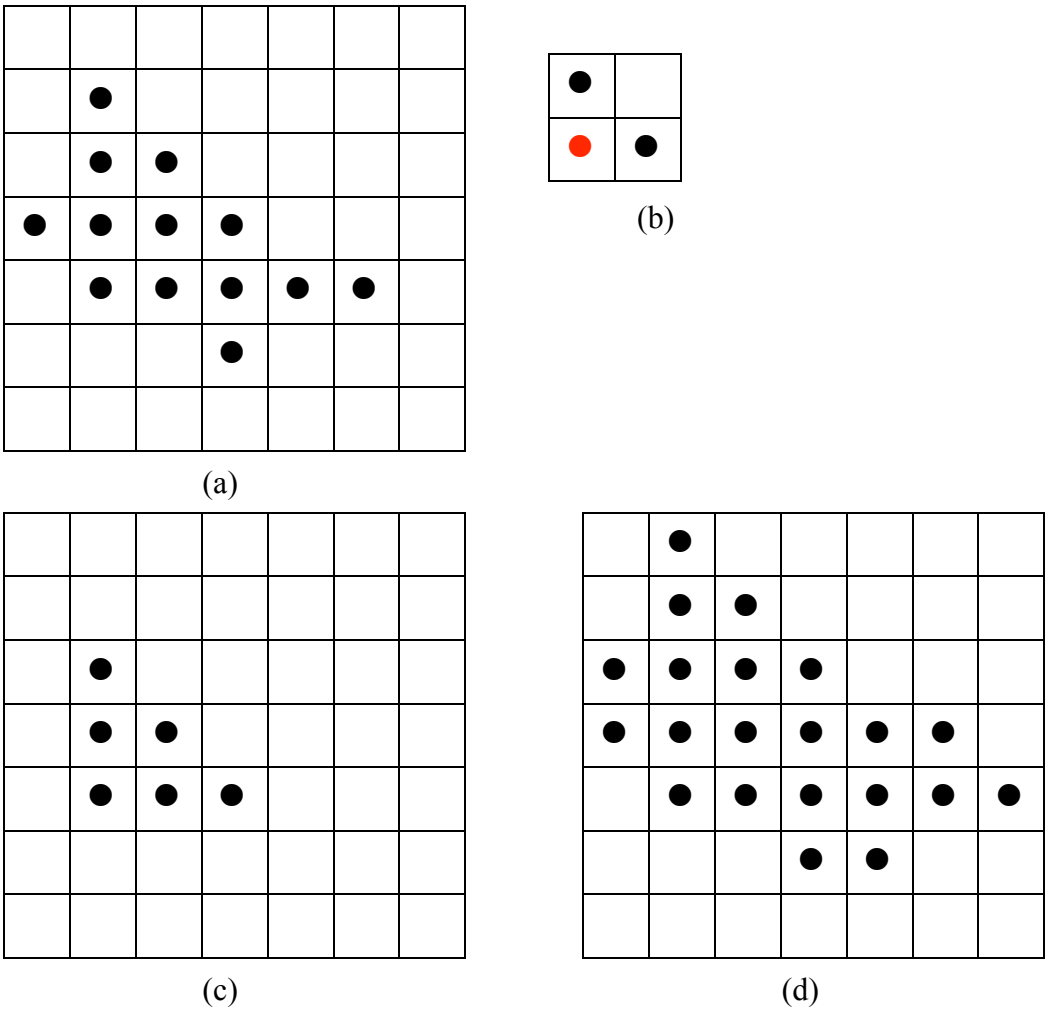


Figure 2.3 Examples for erosion and dilation

The other two kinds of operation is *opening* and *closing*.

The opening operation is erosion firstly and then dilation of the result:

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

The closing of image A by structuring element B is defined as:

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

2.4 Linear regression [19 - 21]

Linear regression is a concept in statistics. In many applications, we want to find the relationship between a single independent variable Y and dependent variables x. The independent variable is also called response variable. The analysis of the relationship between Y and x requires a statistical model, which is called regression model. If the relationship is linear, and there is only one independent variable, the regression model is called simple regression model, which can be written as:

$$Y = \alpha + \beta x + e$$

Where e is the random error. The mean random error is 0.

The responses Y_i corresponding to the dependent variable values x_i, $i = 1, 2, ..., n$ can be observed, and the parameters α and β can be estimated in the simple regression model.

2.4.1 Least squares estimators of the regression parameters

Find A and B are the estimator of α and β, respectively. The sum of squares of error e is denoted SSE, defined as:

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - A - Bx_i)^2$$

In this method, we want to minimize the SSE to estimate the parameters. Firstly, we compute the differential with respect to A and B. Then, setting the partial derivative equals to zero. At last, we can get:

$$B = \frac{\sum_{i=1}^{n} x_i Y_i - \bar{x} \sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

$$A = \bar{Y} - B\bar{x}$$

Where, $\bar{Y}$ and $\bar{x}$ are the mean value of response and dependent variables, respectively.

$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$, and $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. When we estimate the parameters, we can draw the straight line $A + Bx$ as the estimated regression line.

Chapter 3. The proposed algorithm for detection cracks.

In this chapter, we will present the algorithms we designed for this project. We get the skeleton of cracks on a binary image now, and we want to represent the cracks by straight white lines. This can be finished in several steps: (See figure 3.1)

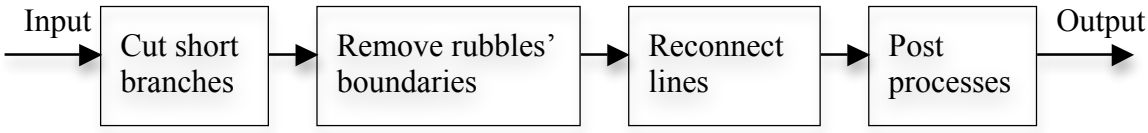


Figure3. 1 Whole processes in this project

In previous work, we extract the cracks from the rocks' pictures meanwhile the boundaries of "big noise" are kept. The geologist wants the rock boundaries of grasses, but not the rubbles' boundaries. So we have to remove the rubbles' boundaries, and the method is similar with removing "big noise". We will give the implementations of "removing rubbles' boundaries" and "post process" in chapter 4. In this chapter, we present why and how to design algorithms to cut the short branches, to decide which lines that probably belongs to the same cracks, and to connect those lines, which hopefully represents the same cracks.

3.1 Cutting short branches

From prior work, we have got the skeleton of the rock cracks. Zooming in the prior result (See figure 1.3), we can find that there are lots of small branches on the crack lines (as shown as figure 3.2). The branches will impact the measurement of the cracks, so we want to cut the short branches from the main cracks and remove them.

Definition 3.1 "Object": In this project, an object refers to an 8-connected component in the image.

Figure3.2 a part of main crack in the example image

From figure 3.2, we define two kinds of objects that should be cut. The first one is crossing line, which is formed by short branches connected to the rock crack line, as shown as figure 3.3 (a) and (b). Another kind is curve line, which will impact the measurement of cracks' direction as shown as figure 3.3 (c).

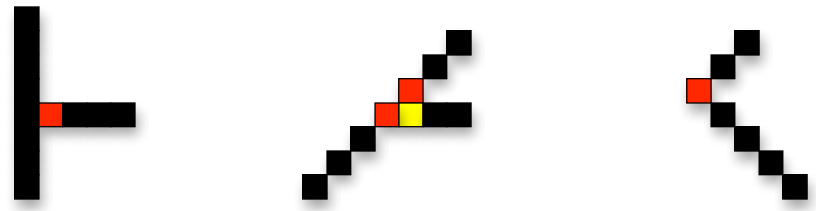


Figure3. 3 Lines should be separated.

3.1.1 Crossing lines

In figure 3.2, we can see that the branches, which are the information we don't want, are consisted of just a few pixels, but they connect to the main cracks. The short branches and a main crack form one object in this picture. So we can remove the short branches from the main crack in two steps.

(1) Separating short braches from main cracks.
(2) Removing the too short lines from the image.

To implement the first step, we should separate all the lines, if the line is curve line or crossing with other lines.

If the short branch connects to the main cracks, they are always in the way as shown as figure 3.3 (a) and (b). If red pixels are removed, the branch is separated from the main cracks. And we can find that the red pixels have several features:

(1) They have more than three neighbors.

(2) They have both diagonal neighbors and 4-adjacency neighbors.

Now, we use spatial correlation process to test the neighborhood for each pixel.

Firstly, define three kinds of mask as shown as follow:

1	1	1
1	1	1
1	1	1

(a)

0	1	0
1	0	1
0	1	0

(b)

1	0	1
0	0	0
1	0	1

(c)

Figure3. 4 Filtering masks

We denote the characteristic response of a mask for correlation as R. The correlation process as I discussed before. So if using the first filter mask in figure 3.4, R is the number of neighbors. When we use the second filter mask, and R is not equal with 0, that means there is 4-adjacency neighbor exist. When the filter mask as shown as figure 3.4 (c), and R is not 0, that means there is diagonal neighbor exist.

If the pixel has more than three neighbors, and has both 4-adjacency and diagonal neighbors, assigning the value “0” to it. In this way, the red pixel can be moved away and all crossing lines can be separated from each other. Then we define the number of pixel of each object as “area”. If the “area” is too small, assigning the value “0” to all the pixels in this object. So the small braches has been moved away.

For the object as figure 3.3(b), it is reasonable to remove the yellow pixel, but we remove the two red pixels. Because it is easy to implement and we can also get the result as we expect, if removing the two red pixels.

Definition 3.1: “Area”: The number of pixels in the object.

Algorithm 3.1 Removing short lines

(1) Define masks, $h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $h_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and $h_3 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

(2) Calculate R_1, R_2 and R_3 as the response of mask h_1, h_2 and h_3 for correlation respectively.

(3) If, $R_1 > 3$ $R_2 \neq 0$ and $R_3 \neq 0$, assign value "0" to that pixel.

(4) Label all the objects in the image and calculate the area for each object in the image.

(5) If $area < \tau_1$ (τ_1 is a threshold value for the area we defined before), assign "0" to all the pixels in this object to move it away.

3.1.2 Curve

Definition 3.3: "Opposite Position": A pixel p (as shown as the blue pixel in figure 3.5) at coordinate (x,y) has two neighbors whose coordinates are given by

$$(x,y-1),(x,y+1) \text{ or } (x-1,y),(x+1,y)$$

$$\text{or } (x+1,y-1),(x-1,y+1) \text{ or } (x-1,y-1),(x+1,y+1)$$

We call the two neighbors are at opposite position.

Figure3.5 Opposite Position

The geologists want to measure the length and direction of rock cracks. So we have to make each line straight. If there is a line as shown in figure 3.3(c), it is hard to measure the direction. One alternative is to cut it and make it become two straight lines, that means removing the red pixel in figure 3.3(c). We can find that the difference between "MajorAxisLength" and "MinorAxisLength" is smaller in the object as shown in figure 3.3(c) than the difference in the straight-line shaped object. In this kind of object, the red pixel has two neighbors, but they are not at the opposition position. Using this information, we can find the red pixel, and assign the

value “0” to the red pixels to cut this kind of line to make the curve becoming two straight lines.

We can also use the spatial filtering to find whether a pixel has two neighbors at opposite position. Define four masks as shown as below

0	1	0
0	0	0
0	1	0

(a)

0	0	0
1	0	1
0	0	0

(b)

1	0	0
0	0	0
0	0	1

(c)

0	0	1
0	0	0
1	0	0

(d)

Figure3.6 Filtering mask

Firstly, we can use the filtering mask figure 3.4(c). If the response equals 3, there are two neighbors. Then using the masks in figure 3.6, if all the four responses are different from zero, it means there are no neighbors in the opposite position.

Algorithm 3.2 Cutting curved lines

(1) Label the image; calculate area and convex area for all the objects.

(2) Define masks: $h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $h_4 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, $h_5 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$,

$h_6 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $h_7 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.

(3) Calculate R_1, R_4, R_5, R_6 and R_7 as the response of mask h_1, h_4, h_5, h_6 and h_7 for correlation respectively.

(4) If $ConvexArea - Area > \tau_2$ (τ_2 is a threshold value we defined before).

(5) At the same time, if $R_1 = 3$, $R_4 \neq 0$, $R_5 \neq 0$, $R_6 \neq 0$, and $R_7 \neq 0$, assign value “0” to that pixel.

After removing short branches and cutting curved lines, the result (has been zoomed in) is as shown as figure 3.7

Figure3.7 After removing short branches

3.2 Reconnecting cracks

After removing short branches and cutting curved lines, we can find that there are lots of short, straight lines on the image (see figure 3.8). Actually, these lines represent one crack. So we have to reconnect these lines.

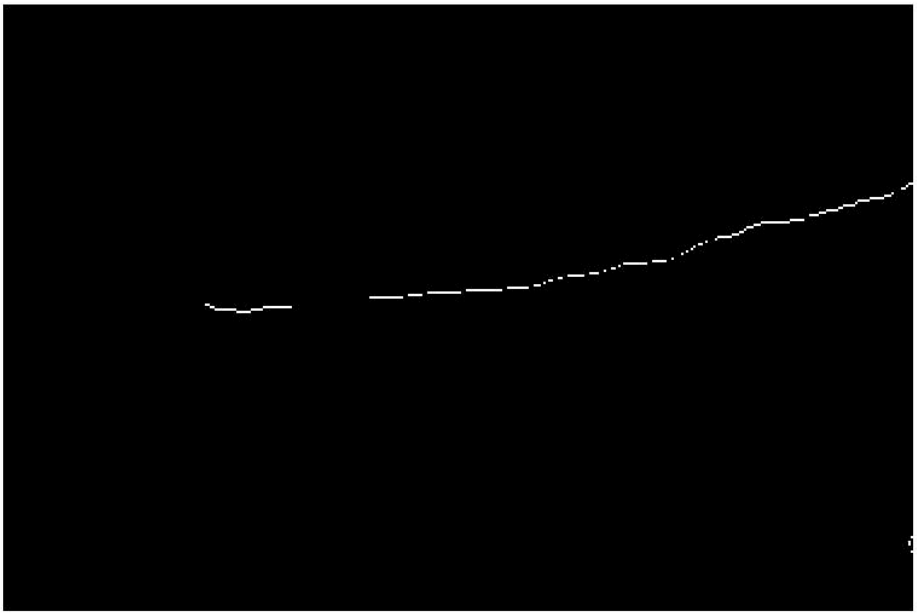


Figure3.8 Zooming in the image after cutting

Before reconnecting, we have to group the lines, which represent the same cracks. For two lines n and m, with centre points (x_n, y_n) and (x_m, y_m), their orientations are θ_n and θ_m, respectively. Orientation θ is an angle in degrees ranging from -90^o to 90^0. There are three conditions can be used to group the lines.

(1) Orientation.

$|\theta_n - \theta_m| < \tau_3$ (τ_3 is a threshold value we defined before)

If the lines represent one cracks, their orientation must be very close.

(2) Distance

$\sqrt{(x_n - x_m)^2 + (y_n - y_m)^2} < \tau_4$ (τ_4 is a threshold value as we defined before)

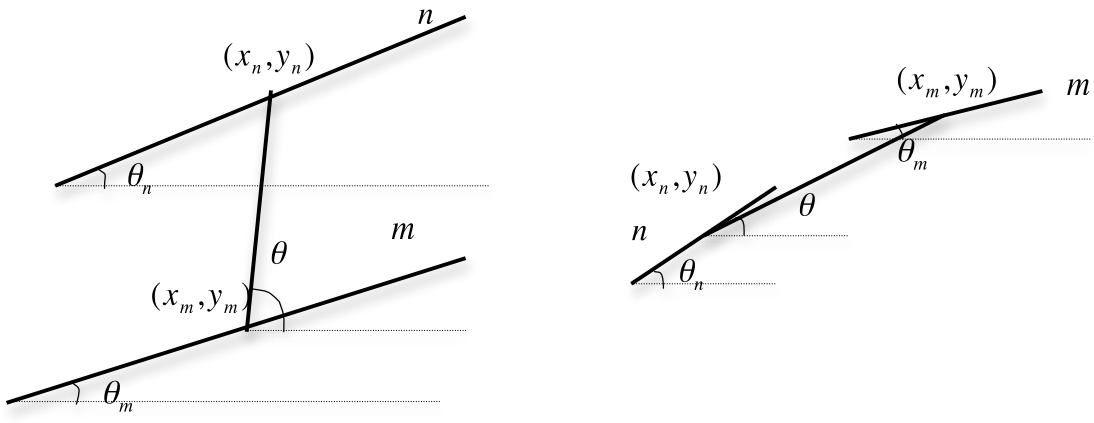
If the lines represent one crack, their position must be very close. That means the distance between them is a very small value.

(3) Unparallel condition

$|\theta - \theta_n| < \tau_5$ and $|\theta - \theta_m| < \tau_5$ (τ_5 is a threshold value we defined before)

Where, $\theta = \arctan k$, $k = -\dfrac{y_n - y_m}{x_n - x_m}$.

If both of the two lines' orientation and position are very close, they may be parallel lines. We have to make sure that they are not parallel. θ is the orientation of the line, which connects the centre points of lines n and m. If the two lines represent one crack, the difference between θ and their orientation must be very small. (See figure 3.9)



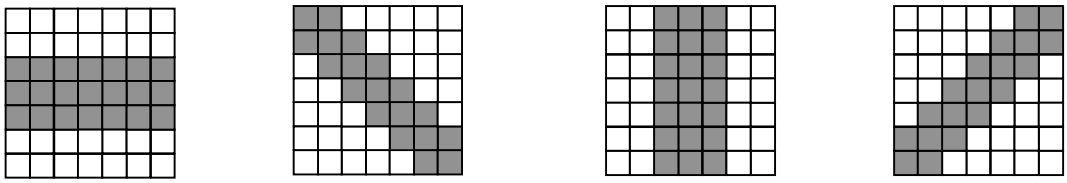
(a) (b)

Figure3.9 Two parallel lines and unparallel lines

Now, after grouping the lines, we have to reconnect them together. There are two ways can be used. The first one is morphological dilation operation with specifically structure element. Another way is using linear regression to draw a new straight line to represent the crack.

3.2.1 Reconnection by dilation operation

The lines, which represent one crack, are very close to each other (See figure 3.8). So we can use morphology dilation operation (See section 2.3) to reconnect them. Firstly we define four kinds of structure elements (SE) to be used in four different situations (see figure 3.10). (Gray squares denotes pixels with logical value 1, and blank square denotes pixels with logical value 0)



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

Figure3.10 Four kinds of structure elements

If the orientation of the line in the range (-30^0, 30^0), we can use the first structure element in the figure 3.10. If the orientation in the range (-60^0, -30^0), we can use the SE as shown as figure 3.10(b). If the orientation in the range (-90^0, -60^0) and

(60^0, 90^0), the structure element as shown as figure 3.10(c) should be used. And we can use the SE as shown as figure 3.10 (d), if the orientations in the range (30^0, 60^0).

Algorithm 3.3 Reconnection by morphological dilation

(1) Label the image; calculate orientation and center point for all the objects.

(2) Defined four structuring elements, as shown in figure 3.10.

(3) Group the objects which should be reconnected together by the conditions we discussed before.

(4) If $-30^0 \le \theta \le 30^0$, apply the dilation operation on the group of objects by SE as figure 3.10(a).

If $-60^0 \le \theta \le -30^0$, apply the dilation on the group of objects operation by SE as figure 3.10(b).

If $-90^0 \le \theta \le -60^0$ or $60^0 \le \theta \le 90^0$, apply the dilation operation on the group of objects by SE as figure 3.10(c)

If $60^0 \le \theta \le 90^0$, apply the dilation operation on the group of objects by SE as figure 3.10(d).

(5) Obtain the skeleton the image at last.

Applying dilation operation on the lines with different orientation by different structure elements, we can reconnect the cracks. After morphology operation, the result from figure 3.8 is as shown as below (figure 3.11).

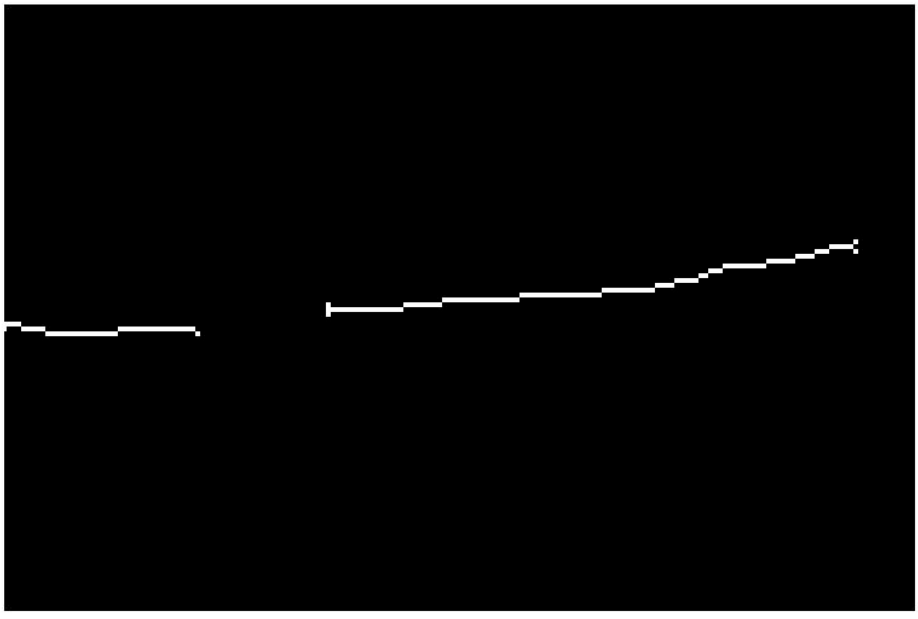


Figure3.11 Result from morphology process to reconnecting

From figure 3.8 and figure 3.11, we can find that if the distance of two lines is long, the cracks cannot be connected, so we have another way to connect them –linear regression.

3.2.2 Reconnection by linear regression

Pixel is a term to denote the elements of a digital image. Each pixel has its own address, and the address of a pixel corresponds to its coordinates. In this case, the horizontal coordinate of pixels on the lines, which should be connected, can be considered as dependent variable x, and the vertical coordinate can be consider as observed response Y. We want to generate a new straight line to represent the cracks. The straight line is a simple linear regression model:

$$Y = \alpha + \beta x$$

Where, a and b are parameters can be estimated using least squares as we discussed in section 2.4

Then we can draw an estimated regression line to represent the crack.

In this project, we connect the lines using linear regression by the algorithm 3.4:

Algorithm 3.4 Reconnection by linear regression

(1) Label the image; there are N objects in the image. Group the lines that should the connected together.

(2) For each group of lines, generate a new image with the same size as the original image. In each new image $f_n(i,j)$, for $i = 1,2...,I$, $j = 1,2...,J$ and $n = 1,2...,N$, there is only one group of lines (white lines on black background).

(3) For each group of lines, if $f_n(i,j) \neq 0$ (the pixel is white), assign i to an array x, and assign j to an array y.

(4) Suppose that there is linear relationship between x and y, $y = \alpha + \beta x$. Using least squares estimator to calculate the parameters α and β.

(5) For each group of lines, generating a new image $g_n(i,j)$, for $i = 1,2...,I$ and $j = 1,2...,J$. In this image $g_n(i,j)$, all pixels' value is 0. It is an image with all pixels black.

(6) Find maximal and minimal value in the x array $x_{\min}$: and $x_{\max}$. For each horizontal coordinate i from $x_{\min}$ to $x_{\max}$, assign value 1 to the pixel, whose coordinate is (i,j), when $j = \alpha + \beta x$.

(7) Add up all $g_n(i,j)$. $g(i,j) = \sum_{n=1}^{N} g_n(i,j)$. $g(i,j)$ will be the result as we expected.

In this way to reconnect the lines, there are two problems. First, the estimated line may be outside the image border, and there is $j = \alpha + \beta i > J$ or $j = \alpha + \beta i < 0$. This situation is not allowed. So we have to limit $y_{\min} \le j \le y_{\max}$ before assigning 1 to the pixel with coordinate (i,j).

And in digital image, the real continuous sense has been converted into digital form (discrete). The line is represented by discrete pixels in the digital image. If $|y_{\max} - y_{\min}| >> |x_{\max} - x_{\min}|$, the estimated regression line will be represented as several single points in the image $g_n(i,j)$. So in this situation, we have to make a small change. For each vertical coordinate j from $y_{\min}$ to $y_{\max}$, when the horizontal coordinate $i = \frac{j - \alpha}{\beta}$, assigning value "1" to the pixel, with coordinate (i,j).

So the steps (6) and (7) should be changed as:

(6) Find maximal and minimal value in the array x, and y, respectively. In the image $g_n(i,j)$, if $|y_{\max} - y_{\min}| \geq |x_{\max} - x_{\min}|$, for each horizontal coordinate i from $x_{\min}$ to $x_{\max}$, calculating $j = \alpha + \beta i$. Then if $j = \alpha + \beta i \leq J$ and $j = \alpha + \beta i \geq 0$, assigning value "1" to the pixel, whose coordinate is (i,j)

(7) If $|x_{\max} - x_{\min}| < |y_{\max} - y_{\min}|$, for each vertical coordinate j from $y_{\min}$ to $y_{\max}$, calculating $i = \frac{j-\alpha}{\beta}$. Then if $i = \frac{j-\alpha}{\beta} \leq I$ and $i = \frac{j-\alpha}{\beta} \geq 0$, assigning value "1" to the pixel, whose coordinate is (i,j).

(8) Add up all $g_n(i,j)$. $g(i,j) = \sum_{n=1}^{N} g_n(i,j)$. $g(i,j)$ will be the result as we expected.

Using linear regression to generate a new line to represent the group of lines as shown in figure 3.8, the result is as shown in figure 3.12.

Figure3.12 The result from connecting lines by linear regression

Chapter 4. Implementation

4.1 Matlab

Matlab stands for matrix laboratory. It is originally developed to provide easy access to matrix software. Now, MATLAB is a high-performance language for technical computing in education and industry. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. [24]

There are many toolboxes included in Matlab. Toolboxes can extend the capability of the Matlab environment to solve particular classes of problems. The image processing toolbox supports a lot of image processing operation.

GUIDE is the MATLAB graphical user interface development environment, providing a set of tools for developing graphical user interfaces (GUIs). These tools greatly simplify the process of designing and building GUIs [24].

Our project is implemented in Matlab and by using image processing toolbox.

4.2 Graphical user interface

In this project, we develop a graphical user interface (GUI) to make the interaction between people and program user friendly.

With the help of the GUIDE Layout Editor, we can design a GUI by clicking and dragging GUI components – such as push button, text field – into the layout area. It is also very easy to modify the components in GUIDE Layout Editor. The GUI layout is stored as a FIG-file in Matlab.

An M-file will be generated automatically after saving the GUI layout. The M-file contains code to initialize the GUI and a framework for GUI callbacks. A callback is a function that we write and associate with a specific GUI component. We can add code to callbacks to control the operation, when users interact with GUI, in the M-file. When an event occurs for a component, the component callback will be invoked, which is triggered by that event.

The GUI, designed for this project is as shown in figure 4.1

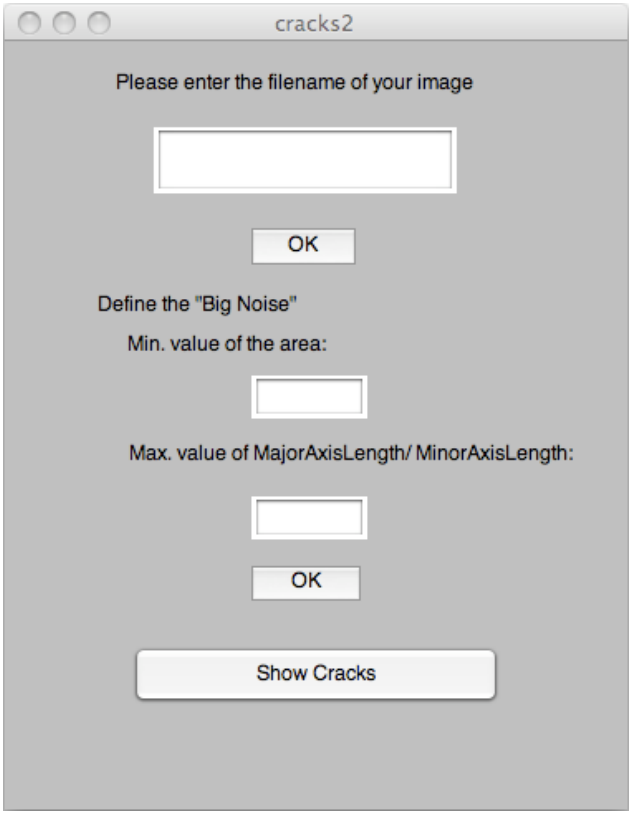


Figure4.1 Graphical user interfaces

There are six callbacks in this GUI: three for editable text box – filename, area and shape; the other three for push button – OK1, OK2 and result.

In the first editable text box, the user should enter the filename and the standard file extension, which is used for specifying the file's format, of the input image. And if the input image is not in a directory on the Matlab path, or in the current directory, the user has to specify the full pathname.

When the user push the first "OK" button, the binary image without preprocessing will be displayed.

As we discussed before, the "big noise" is distinguished from cracks by the size and shape. (See section 1.4) We should choose two different threshold values to define the "big noise". And for different images, we should choose different threshold values. Thus, the user is asked to enter the threshold values through observing the binary image, which is generated at the last step. When you push the second "OK" button, a binary image with "big noise" in it will be shown. If the result is not satisfying, different threshold values can be tried.

Then by pushing the “Show Cracks” button you get two images. So there is a binary image with rock cracks showing as white lines on black background. Another is the input color image with red lines super imposed to represent the cracks.

The whole process can be presented as figure 4.2.

The rest of this chapter will present the implementation details for each GUI push button component.

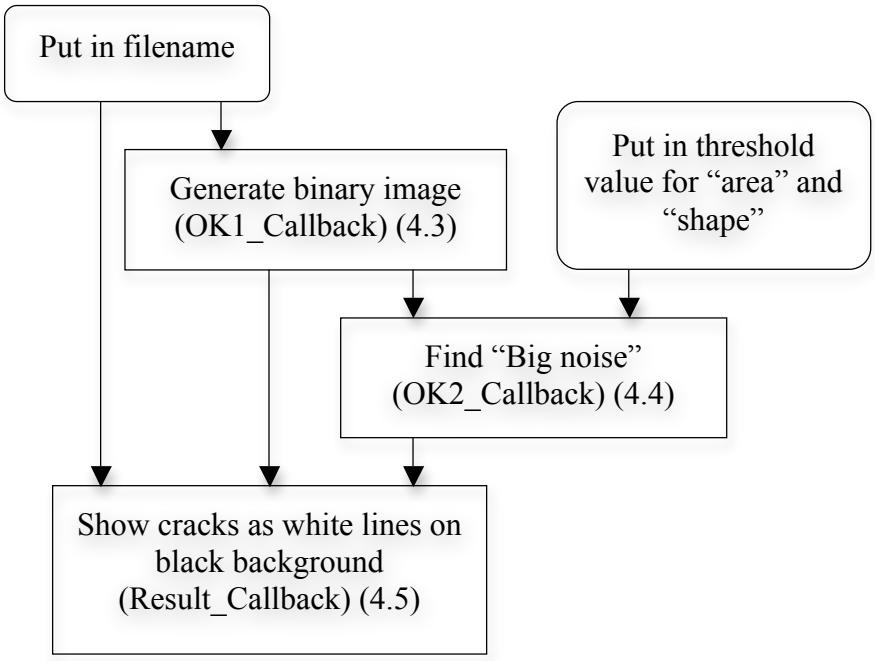


Figure4.2 Workflow of this project

4.3 Generate binary image

In this part, we obtain the image’s filename as a string value from edit text, and read the image to handle variable. We generate two binary images I and I_1: I is an image without preprocessing; I_1 is a preprocessed image. This part is implemented by function OK1_Callback. The workflow of this function is: (also see figure 4.2)

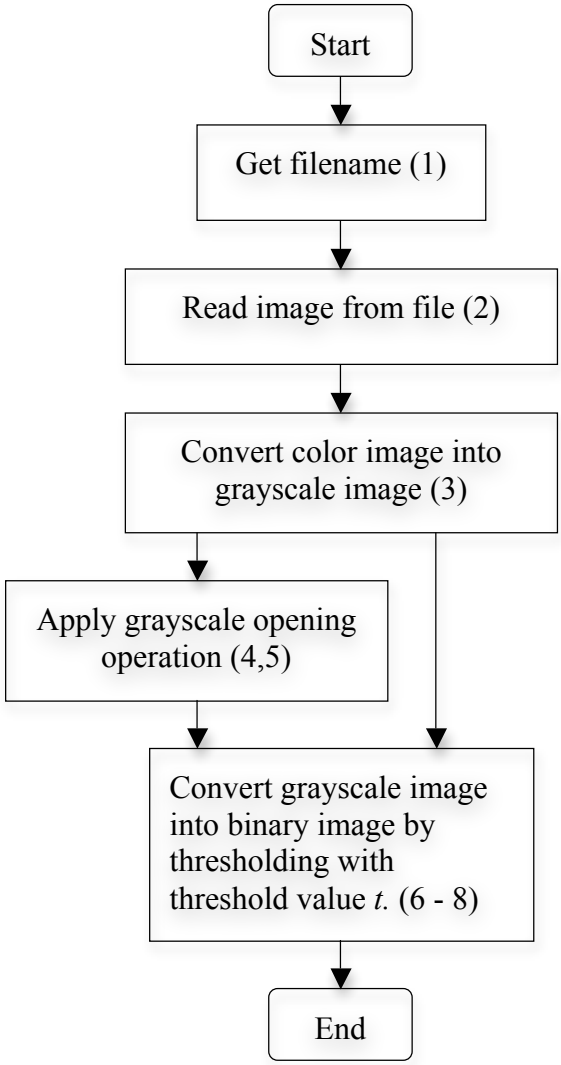


Figure4.3 Flow chart of OK1_Callback

(1) Get the content of edit text box – "filename" as string value.

(2) Read the image from the file specified by the string "filename" to handle objects I and I_1 (Input image is color image, so I and I_1 are M-by-N-by-3 arrays). This step is implemented by Matlab command "*imread*".

(3) Convert I and I_1 into grayscale image, so I and I_1 becomes an M-by-N array. This can be implemented by Matlab command "*rgb2gray*".

(4) Create a flat disk-shaped structuring element *se* with radius 11 by Matlab command "*strel*".

(5) Perform morphological opening on the grayscale image I_1 with the structuring element *se*. Return the result to I_1. The command "*imopen*" can implement this operation.

(6) Return the intensity histogram of image I to h by command "*imhist*".

(7) Compute a probable threshold value *t* by a function "*minhist*". The function "*minhist*" is made to find the intensity value *t* when the histogram has its minima.
(8) Using threshold value *t* and command "im2bw", convert grayscale *I* and I_1 to binary image by thresholding. Return the result to *I* and I_1 respectively.
(9) Save the handles objects as GUI data.

There are three output of this callback: I, I_1, and I_2. I, the binary image without preprocessing, is the input for the OK2_Callback. I_1, the preprocessed binary image, is one of the inputs for the Result_Callback, I_2, the input color image without any processing, is the other input for the function Result_Callback.

4.4 Find "big noise"

The plants, covering parts the rocks, and rubbles are defined as "big noise" in this project as we discussed in section 1.4. We find the "big noise" in the callback OK2. Input of this callback is the image *I*, one of the output of OK1_Callbak. The process in the function OK2 is: (see figure 4.4)

(1) Get the user typed area threshold value *a* and shape threshold value *s* used to defined the "big noise".
(2) Covert *a* and *s* from string data to double data by the Matlab command "*double*"
(3) Label the all connected components in the input image *I* as objects. There are N objects in the image I. This can be implemented by the command "*bwlabel*".

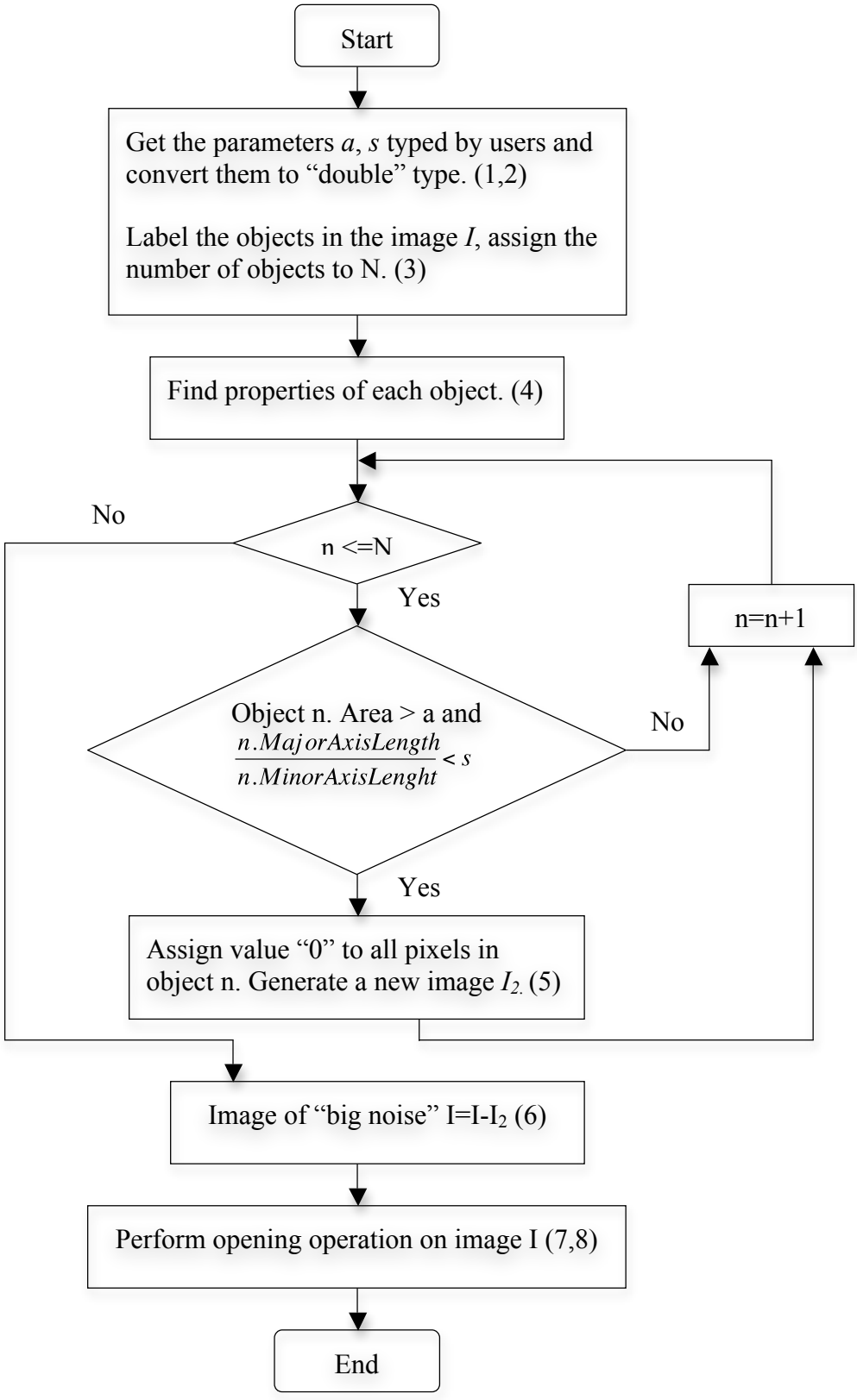


Figure4.4 Flow chart of OK2_Callback

(4) Measure properties of every objects in the image *I* by the command "*regionprops*"

(5) For each objects, if the $area > a$ and $MajorAxisLength/MinorAxisLength < s$, assign value "0" to all the pixels in this object. In this way, generate a new image I_2.

(6) The "big noise" $I = I - I_2$.

(7) Create a flat square-shaped structuring element *se* with radius 11 by Matlab command "*strel*".

(8) Perform morphological opening on the image *I* with the structuring element *se*. Return the result to *I*. The command "*imopen*" can implement this operation.

(9) Save *I* as GUI data.

In this way, we obtain the image with only "big noise" on it as another input for the callback result.

4.5 Show cracks as white lines on black background.

When the user presses the last button "Show cracks", he/she can get the result of this project. This is implemented by the function Result_Callback.

There are two input images for this function: the one is I_1 – preprocessed binary image; another is *I* – the "big noise" image. The process in this function is as shown as figure 4.5

(1) Subtract the "big noise" I_1 from preprocessed image *I*. We can get the image *I* without "big noise", but the boundaries of "big noise" are kept on the image.

(2) Get the skeleton the cracks by the Matlab command "*bwmorph*".

(3) Cut the short branches away from the rocks cracks as we discussed in section 3.1.1. The Matlab function "*cut_shortbrach*" listed in appendix A can be used to implement this.

(4) Remove the boundaries of rubbles way from this image. It is implemented by the function "*remove*" listed in appendix A.

(5) Cut the curved lines as we presented in section 3.1.2. The Matlab function "*cut_curveline*" listed in appendix A can be used to implement this.

(6) Reconnect the lines, which represent one crack, by linear regression. We use the function "*linearegression*" to implement this. And the theory of this function as we discussed in section 3.2.2.

(7) Perform some post process by the function "*post*" on the image. The function "*post*" is listed in appendix A.

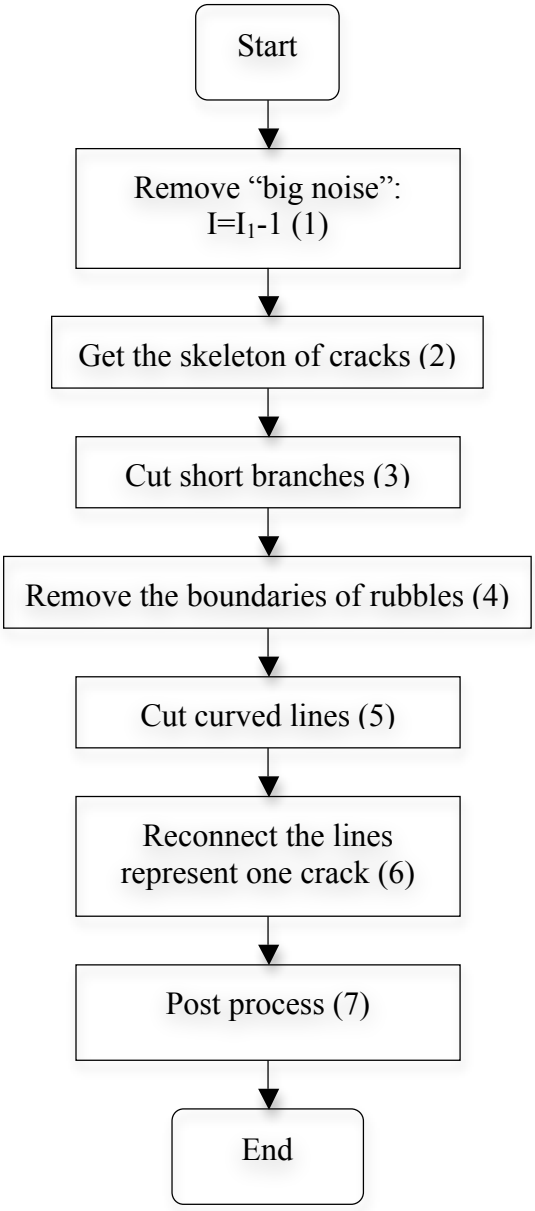


Figure4.5 Flow chart of Result_Callback

There are several functions, which are used in this call back, are developed by us. We have presented the method before. Now, we give the implementation detail of this function.

4.5.1 Cut short branch

In this part, we use algorithm 3.1 – removing short lines to separate the crossing lines and remove short lines. The flow chart of this algorithm is shown as figure 4.5.

In this function, we deal with the image as a two-dimensional matrix. Firstly, after filtering operation, check the value of each element in the matrix and assign value “0” to the “crossing point”. Finally, the very small objects are remove. The Matlab

commands, "*size*", "*imfilter*", "*bwlabel*", "*regionprops*" are used in thisis listed in appendix A program, and the function.

4.5.2 Remove rubbles' boundaries

When we remove the "big noise", the boundaries of them are kept, because the geologists want some of them. But the boundaries of small rubbles, which are not welcomed, should be removed. The rubble is a kind of "big noise". Their boundary shape is typically approximately like a circle – not long and thin, and "area" is not large. Thus, we can distinguish the rubbles' boundaries from cracks by "shape":

For each object, if $$\frac{MajorAxisLength}{MinorAxisLength} < 7$$

Remove this object. We implement this in Matlab as the program – moving "big noise", but not so many conditions. (See figure 4.3).

4.5.3 Cut curve line

The algorithm 3.2 describes how to cut curved lines. In this algorithm, the key point is to find the corner pixel, and assign value "0" to that pixel. The method to implement the algorithm 3.2 is very familiar with the implementation of algorithm 3.1, whose program flow chart is shown in figure 4.6. The differences are:

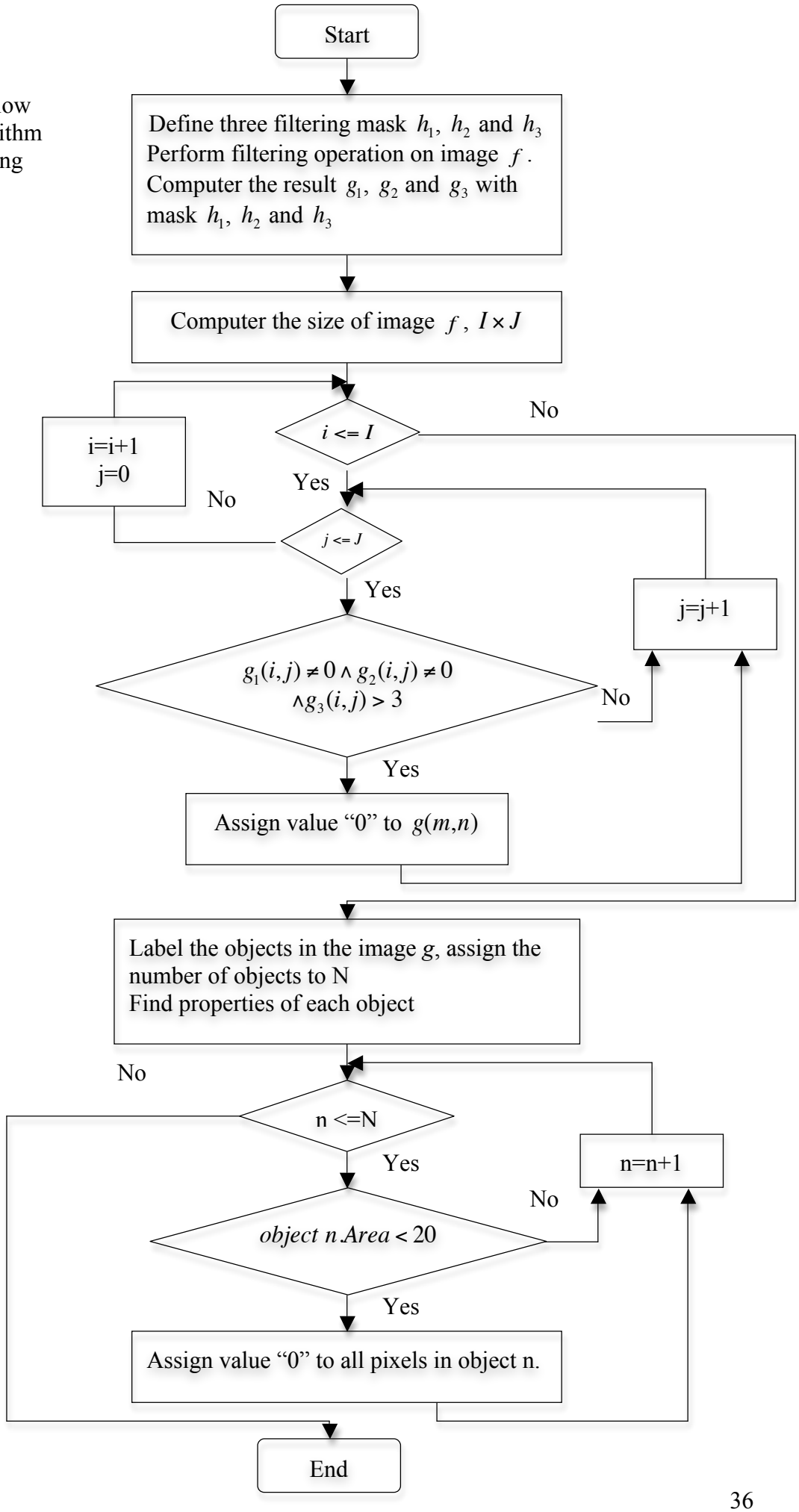


Figure 4.5 Flow chat of algorithm 3.2 – removing short lines.

(1) Define different filtering masks, which have been discussed in section 3.1.2. The filtering masks used to find the corner pixel on curved line was presented in algorithm 3.2.

(2) In this algorithm, we do not need to remove any of the objects. The corner points are found, and we assign value "0" to that point, after that program is ended.

4.5.4 Connect lines by linear regression

We implement the algorithm 3.4 by Matlab. This algorithm is used to connect the lines, which represent a same crack. The implementation of this algorithm is presented by flow chart in Figure 4.7 (Figure 4.7 is too large to be presented in one page)

(1) For the input binary image f, label all the objects in the image and find the properties of each object by Matlab command "*bwlabel*" and "*regionprops*" respectively. There are N objects in the input image.

(2) f is an $I-by-J$ matrix. Generate a matrix g with size $I \times J$, and all the pixels' values are "0" in matrix g.

(3) For each object in the image f, find other objects, which should be connected with it, as the conditions we defined in section 3.2. Assign value "N+1" to all pixels in these objects in the image g.

(4) At that time, the pixels in the image g can be consider as single points, which present a linear relationship.

(5) Store the row and column number of pixels, whose values are not 0 in array x and y, respectively.

(6) Consider x as observation predictor variables, and y as observation response. Estimate the parameters b_1 and b_2 by linear regression. This can be done by Matlab command "*regress*".

(7) Generate another matrix g_2 with size $I \times J$, and let all the pixels' values in matrix g_2 be 0.

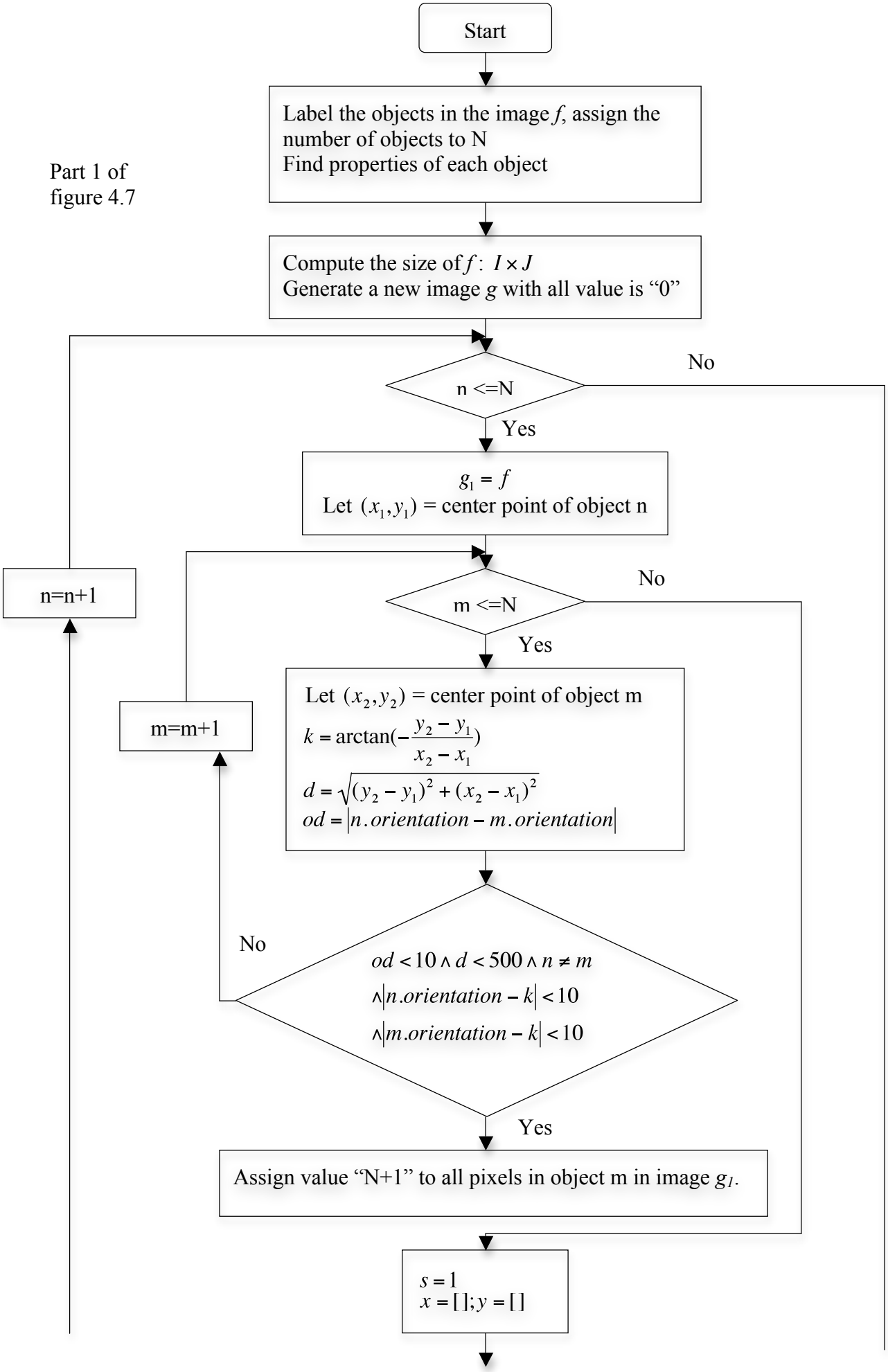
(8) If $|x_{\max} - x_{\min}| > |y_{\max} - y_{\min}|$, when the value of row number i is at the range $[x_{\min}, x_{\max}]$, calculate $j = around(b_1 + b_2 * i)$, if the value of j is in the range $[y_{\min}, y_{\max}]$, make $g_2(i,j) = 0$.

(9) Else, when the when the value of row number j is at the range $[y_{\min}, y_{\max}]$, calculate $i = around((j - b_1)/b_2)$, if the value of j is in the range $[x_{\min}, x_{\max}]$, make $g_2(i,j) = 0$.

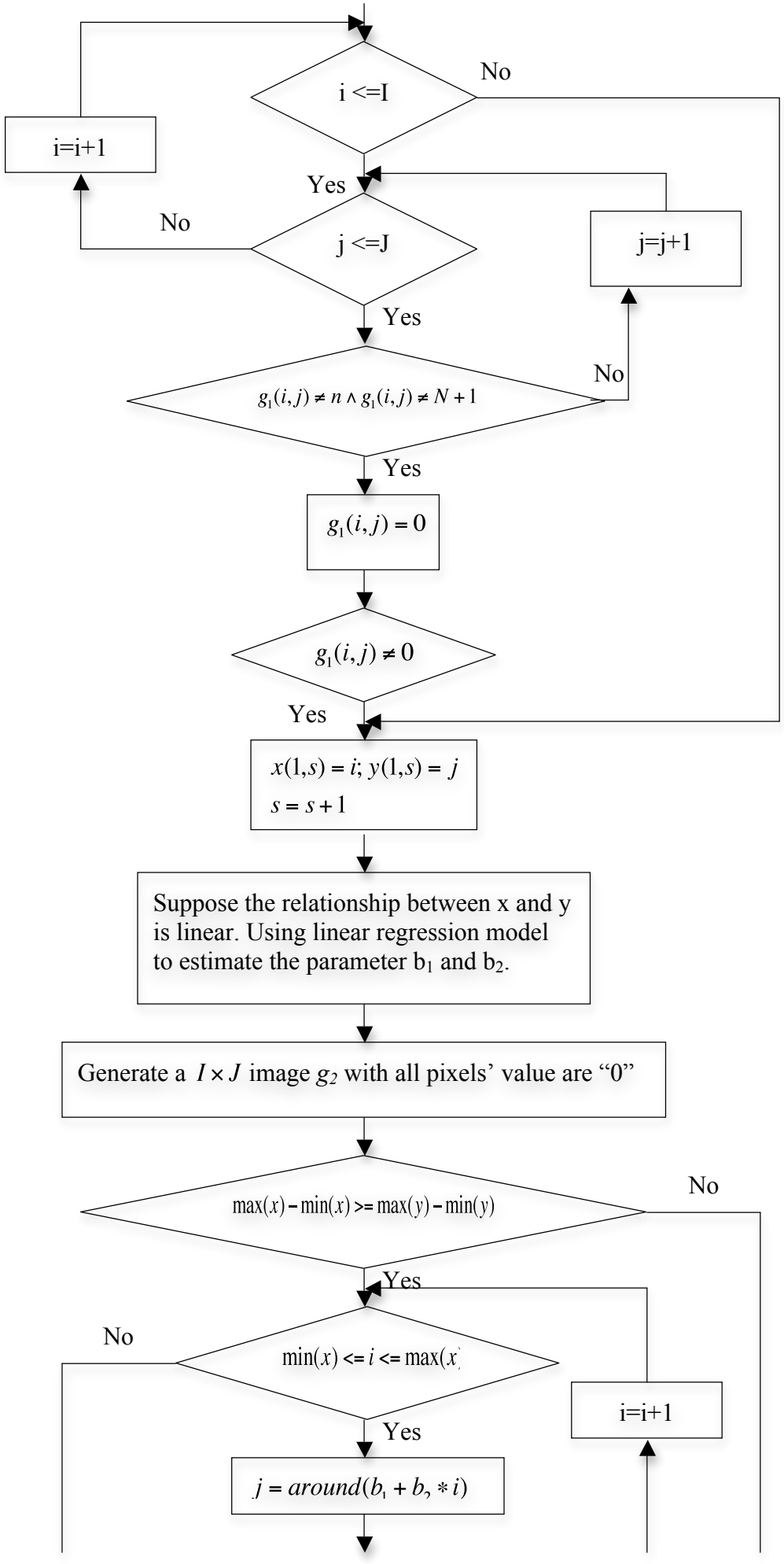
In this way, we draw a straight line on the image g_2. This straight line represents the object n, and other lines, which should be connected with n.

(10) Make $g = g + g_2$. For each object we get a new line, and add them together at last.

Part 1 of figure 4.7



Part 2 of figure 4.7



Part 3 of figure 4.7

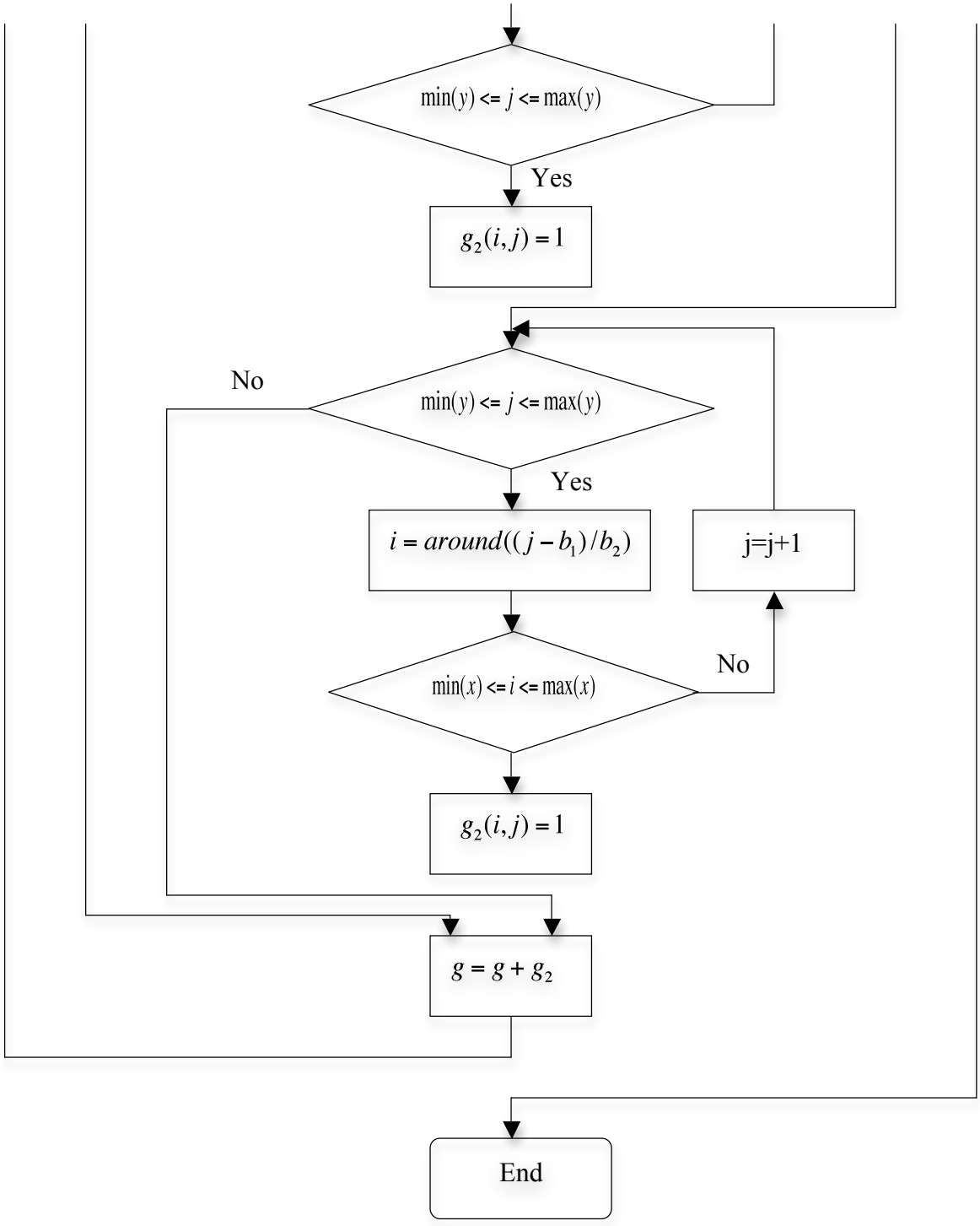


Figure4.6 Flow chat of algorithm 3.4 – reconnect lines by linear regression

4.5.5 Post process

At last, we perform some post processing. Firstly we perform the algorithm 3.1 cut short braches and algorithm 3.4 reconnect lines by linear regression again, but we choose different threshold values for this two algorithm. Then:

(1) Remove some really small objects. The cracks on the image are long object, if the object is really small, that means it is not a crack as we expected.

(2) Make the lines bold (i.e. more than one pixel width) and make white lines become red ones. The wider red lines are plotted on the top of the color images for further analysis.

Removing small objects is very easy to implement:

(1) Label all the objects in the image, and find their properties.
(2) For each object, if the "area" is smaller than 70 pixels, assign value "0" to all the pixels in this object.

The implementation of the second step of post process is:

(1) Perform morphological dilation on the input binary image by the disk-shaped structuring element with radius 5.
(2) Convert the processed image with cracks in "*uint8*" type.
(3) In the uint8 type image, if the pixel's value is not "0", assign value "255" to this pixel.
(4) Generate two matrixes with same size as the input binary image by the command "*zeros*". All the pixels' values are "0".
(5) Concatenate these three matrixes together by the Matlab command "*cat*". In this way, we get a color image with cracks showing as red lines on black background.
(6) Add this color image to the input color rock photo.
(7) Display the result.

Chapter 5. Results and conclusion

5.1 Some results

Now, we deal with the image as shown as figure 1.2, which is one of the digital photos taken by the geologist.

We enter its file name “Ritland1.jpg”, and threshold value for definition the “big noise”: “area” is 500, “ $MajorAxisLength/MinorAxisLength$” is 3.5.

The result is as shown as figure 5.1.

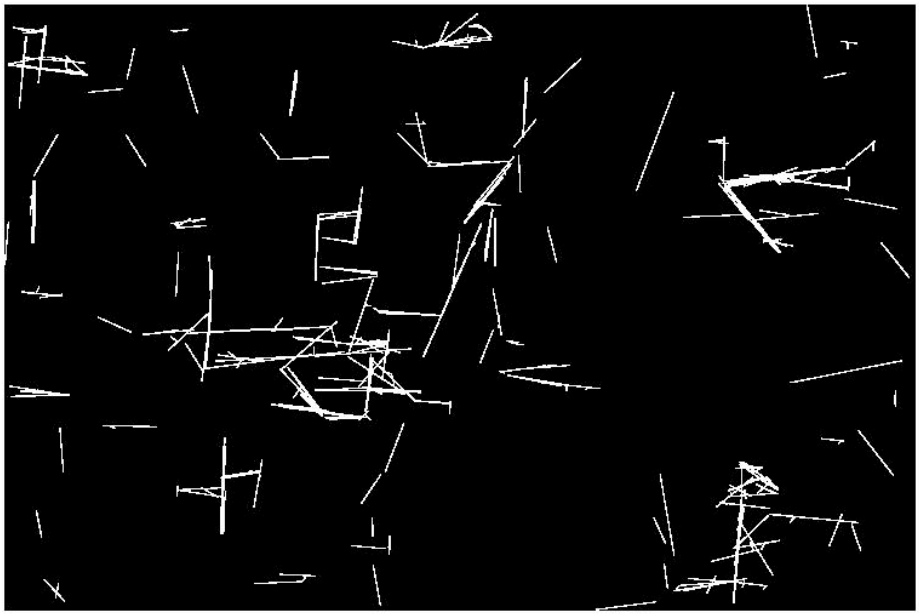


Figure5.1 Cracks shown as white lines

Comparing the result with the original input color photo: (See figure 5.2). The red lines represent the result lines. From these two images, we can find that the result with lines showing the rock cracks approximately, but not perfectly. The processes of this kind of image need to be improved in future.

We use black ellipse to mark two regions in figure 5.2. In the region 1, we can find that there are many overlapped lines. In section 3.2, we have discussed that if two lines are parallel lines, they would not be reconnected. But from region 1 in figure 5.2, the two parallel lines should be reconnected, if the distance between them is really small. In region 2, we can find that there are some slight cracks disappeared. That is not so good; our processing needed to be improved in future.

Matlab is a very powerful tool for academic purpose, but its efficient is very low. It takes some time to run the program and get the result.

The geologist took many photos for the Ritland crater, we deal with them in the same way, and the results will be given in the appendix B.

Figure5.2 Compression the result with input image

5.2 Conclusion

This project is dealing with the pictures photographed by geologist from Ritland meteor crater. We extract the rock cracks from the crater and show them as white lines on a binary image with black background. The project is implemented in Matlab. Statistical technology – linear regression and image filtering processing – spatial and morphological filtering are used in this project.

All the processes in the project are considered in two stages: what kind of information should be processed, and how to process them. The analysis of the different features of information is very important in this project. Through this analysis, we can classify different information:

- Analyze the neighborhood features of each pixel to decide how to separate short branches from cracks.

- Analyse the properties of each object in the image to find which ones should be removed.
- Analyse the properties of each object to find which ones should be connected together.

When finding out the information, which needs to be processed, we can give mathematical presentation of the features, and deal with the information.

Chapter 6. Further research

Although our work solves some problems, more problems follow. In this chapter, we give some topics for further study.

6.1 Estimators of the regression parameters

In this project, we draw a straight line $y = ax + b$ to represent a rock crack, so there are two parameters need to be estimated. We calculate these two parameters by Matlab command “regress”. This function calculates the parameters by least square estimators.

In this method, we minimize the SSE to estimate the parameters. SSE is the sum of the squared differences e between the estimated responses and actual response values. (See figure 6.1)

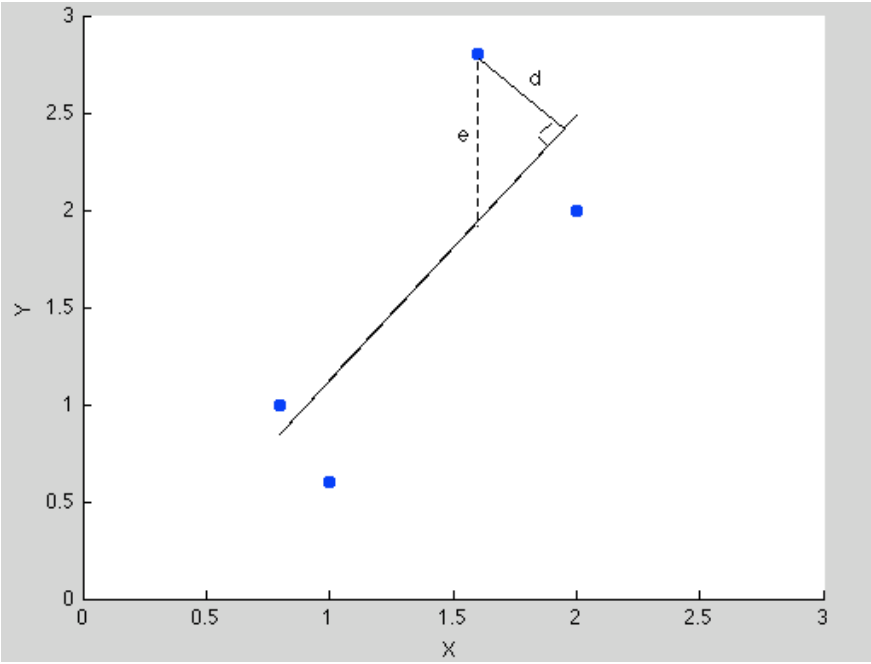


Figure6.1 Scatter diagram with regression lines

From figure 6.1, we can find that the least squares procedure produces a line that minimizes the sum of squares of vertical deviations from the points to the line. But in our project, we hope the line to represent crack, so the points are pixels on lines, and the error can be consider as the distance between the points and estimated line. If we can minimize the sum of squares of distance d from the point to the line, the result will be better. We can develop another function to calculate the regression line in further study.

6.2 Error analysis

From the result, we can find that the lines, we used to represent cracks are not perfect, and one crack represented by several lines. We can design some algorithm to measure the differences between real cracks and lines, which generated by us. Then we can find a line, which represent the crack best.

6.3 Crack properties analysis

After generating binary images showing the cracks as white lines on the black background, we hope that the cracks' lengths and directions can be measured and quantified automatically. This is a part of further works.

6.4 Color factor

From the result, we can find that, not all the rubbles' boundaries have been removed. We have to improve our algorithm. The color factor has been ignored at first, but maybe this factor can be used to analysis, and improve our algorithm.

Bibliography

[1] The Research Council of Norway. “Norsk meteorittkrater under lupen” *(*in Norwegian). Retrieved 8 June 2009.ument.

[2] http://folk.uio.no/sveinkro/

[3] http://www.geo.uio.no/ritland/

[4] Fridtjof Riis, Henning Dypvik, Svein Olav Krøgli (August 2008). “The Ritland crater – An early Cambrian impact structure in West Norway” *The 33rd International Geological congress*, Oslo 2008.

[5] http://en.wikipedia.org/wiki/Ritland_crater

[6] Qin Li, Lei Zhang, Jane You, and David Zhang, “Dark Line Detection with Line Width Extraction”, *Image processing, 2008. The 15th IEEE International Conference*, pp. 621 -624, Oct.2008.

[7] Nitin Aggarwal, and William Clem Karl, “Line Detection in Images Through Regularized Hough Transform”, *IEEE Trans. Image Process*., vol.15, No. 3, pp 582 - 591, Mar. 2006.

[8] Richard O. Duda, and Peter E. Hart, “Use of the Hough transformation to detect lines and curves in pictures”, *Comm. ACM*,. vol. 15, pp. 11 – 15,Jan. 1972.

[9] J. C. Goswami, and A. K. Chan, *Fundamentals of wavelets: theory, algorithms, and applications,* John Wiley & Sons, Inc., 1999.

[10] Jun Li, *A wavelet approach to edge detection*, Aug. 2003.

[11] Yaxiong Huang, and Bugao Xu, “Automatic inspection of pavement cracking distress”, *Journal of Electron. Image*., vol 15, 013017, 2006.

[12] Siwaporn Sorncharean, and Suebskul Phiphobmongkol, “Crack detection on asphalt surface image using enhanced grid cell analysis”, *4th IEEE International Symposium on Electronic Design, Test & Application*., pp. 49 – 54, Mar. 2008.

[13] Peggy Subirats, Jean Dumoulin, Vincent Legeay, and Dominique Barba, “Automation of pavement surface crack detection using the continuous wavelet transform”, ”, *Image processing, 2006. IEEE International Conference*, pp. 3037 – 3040, Feb. 2006.

[14] http://en.wikipedia.org/wiki/Edge_detection

[15] http://en.wikipedia.org/wiki/Image_processing

[16] R. C. Gonzales, and R. E. Woods, *Digital Image Processing, Third Edition*. Pearson Education, Inc., 2008.

[17] Ivar Austovoll, *Computer vision, Machine vision, Lecture notes for MIK 170 image processing.* Department of Electrical and Computer Engineering, UiS. 2009.

[18] Frank Y. Shih, *Image Processing and Mathematical Morphology, Fundamentals and Applications*. Taylor & Francis Group, LLC. 2009.

[19] Sheldon M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists, Third Edition*. Elsevier Inc., 2004

[20] Murray R. Spiegel, John J. Schiller, and R. Alu Srinivasan, *Probability and Statistics, Third Edition*. The McGraw-Hill Companies Inc., 2009.

[21] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye, *Probability & Statistics for Engineers & Scientists, Eighth Edition*. Pearson Education, Inc., 2007.

[22] Patrick Marchand, and O. Thomas Holland, *Graphics and GUI with MATLAB, Third Edition*. CRC Press LLC., 2003

[23] R. C. Gonzales, R. E. Woods, and S. L. Eddins, *Digital Image Processing Using Matlab*. Prentice Hall, 2003.

[24] *Learning MATLAB® 7, Release 14,* The MathWork, Inc., 2005.

[25] *Image Processing Toolbox™ 6 User's Guide*, The MathWork, Inc., 2009.

[26] http://en.wikipedia.org/wiki/Mathematical_morphology

[27] http://en.wikipedia.org/wiki/Linear_regression

Appendix

Appendix A – Matlab function

A.1 Minhist

```
function a=minhist(h)
% Find the intensity value when histogram has minimal.
N=size(h);
a=[];
i=1;
for n=5:N-5
    if h(n)< h(n-1) && h(n)<h(n+1) && h(n)<h(n-2) &&
h(n)<h(n+2) && h(n)<h(n-3) && h(n)<h(n+3) && h(n)<h(n-4)
&& h(n)<h(n+4) && h(n)<h(n-5) && h(n)<h(n+5) && n<100
        a(i)=n;
        i=i+1;
    end
end
return;
```

A.2 Cut_shortbranches

```
function g=cut_shortbranch(f)
%CUT_SHORTBRANCH Cut crossing line and remove small
object in binary image.
%   g=cut_shortbranch(f) returns a binary image with same
size as f,
%   crossing lines have been cut and small objects have
been removed.
g=f;
h1=[0 1 0; 1 0 1; 0 1 0];
h2=[1 0 1; 0 0 0; 1 0 1];
h3=[1 1 1; 1 1 1; 1 1 1];
I1=imfilter(f, h1);
I2=imfilter(f, h2);
I8=uint8(f);
I3=imfilter(I8, h3);
[I,J]=size(f);
%Romove crossing point
for i=1:I
   for j=1:J
       if I1(i,j)~= 0 && I2(i,j)~=0 && I3(i,j)>3
              g(i,j)=0;
        end
    end
end
[g,N]=bwlabel(g);
M=regionprops(g, 'all');
%Remove small objects
for n=1:N
     if M(n,1).Area < 20
```

```
            for i=1:I
               for j=1:J
                   if g(i,j)==n
                       g(i,j)=0;
                   end
               end
            end
     end
end
return;
```

A.3 Remove

```
function g=remove(f)
%ROMOVE Reomve rubbles' boundary
%   g=remove(f) returns a binary image without boundary
of approximate
%   circle shape object. Input and output are binary
image.
[g,N]=bwlabel(f);
M=regionprops(g, 'all');
[I,J]=size(g);
for n=1:N
    if M(n,1).MajorAxisLength/M(n,1).MinorAxisLength < 7
        for i=1:I
            for j=1:J
                if g(i,j)==n
                    g(i,j)=0;
                end
            end
        end
    end
end
return;
```

A.4 Cut_curveline

```
function g=cut_curveline(f)
%CUT_CURVELINE Cut curved lines in a binary image
%   g=cut_curveline(f) returns a binary image without
curved lines.
g=f;
h3=[1 1 1; 1 0 1; 1 1 1];
h4=[0 1 0; 0 0 0; 0 1 0];
h5=[0 0 0; 1 0 1; 0 0 0];
h6=[1 0 0; 0 0 0; 0 0 1];
h7=[0 0 1; 0 0 0; 1 0 0];
I8=double(f);
I3=imfilter(I8, h3);
I4=imfilter(I8, h4);
I5=imfilter(I8, h5);
```

```
I6=imfilter(I8, h6);
I7=imfilter(I8, h7);
[I,J]=size(f);
[g,N]=bwlabel(g);
M=regionprops(g, 'all');
 for n=1:N
     if M(n,1).MajorAxisLength/M(n,1).MinorAxisLength <
10
         for i=1:I
             for j=1:J
                 if g(i,j)==n && I3(i,j)==2 && I4(i,j)~=
2 && I5(i,j)~=2 && I6(i,j)~=2 && I7(i,j)~=2
                     g(i,j)=0;
                 end
             end
         end
     end
 end
```

A.5 Linearegress

```
function g=linearegress(f)
%LINEAREGRESS Reconnect the lines which belong to the
same crack
[f,N]=bwlabel(f);
M=regionprops(f, 'all');
[I,J]=size(f);
g=zeros(I,J);
for n=1:N
    g1=f;
    x1=M(n,1).Centroid(1,1);
    y1=M(n,1).Centroid(1,2);
%Group the lines which belong to a same crack.
    for m=1:N
        x2=M(m,1).Centroid(1,1);
        y2=M(m,1).Centroid(1,2);
        k=-(y1-y2)/(x1-x2);
        k=atand(k);
        d=(y1-y2).^2+(x1-x2).^2;
        d=d.^0.5;
        od=abs(M(n,1).Orientation-M(m,1).Orientation);
        if od<20 && abs(M(n,1).Orientation-k)<10 &&
abs(M(m,1).Orientation-k)<10 && d<200 && n~=m
            for i=1:I
                for j=1:J
                    if g1(i,j)==m
                        g1(i,j)=N+1;
                    end
                end
            end
        end
    end
```

```
%Using least squares learn regression model to estimate
two paremters
    s=1;
    x=[];
    y=[];
    for i=1:I
        for j=1:J
            if g1(i,j)~=n && g1(i,j)~=N+1
                    g1(i,j)=0;
            end
            if g1(i,j)~=0
                            x(1,s)=i;
                            y(1,s)=j;
                            s=s+1;
            end
        end
    end
%Generate a new line ro represent the crack
    x=x';
    y=y';
    b=regress(y, [ones(size(x)) x]);
    b1=b(1);
    b2=b(2);
    g2=zeros(I,J);
    a=min(x);
    c=max(x);
    e=min(y);
    h=max(y);
    if (c-a)>=(h-e)
        for i=a:c
            j=ceil(b1+b2*i);
            if j>=e && j<=h
            g2(i,j)=1;
            end
        end
    else
        for j=e:h
            i=ceil((j-b1)/b2);
            if i>=a && i<=c
            g2(i,j)=1;
            end
        end
    end
%Add uo all reconnected lines.
    g=g+g2;
end
```

A.6 Post

```
function g=post(f)
%POST Remove really small objects.
[g,N]=bwlabel(f);
```

```
M=regionprops(g, 'all');
[I,J]=size(g);
for n=1:N
    if M(n,1).Area < 70
        for i=1:I
            for j=1:J
                if g(i,j)==n
                    g(i,j)=0;
                end
            end
        end
    end
end
return;
```

A.7 Post3

```
function g=post3(f1,f4)
%POST3 Make the white lines into red and plot the red
lines on top of
%original color image
[I,J]=size(f1);
for i=1:I
    for j=1:J
        if f1(i,j)~=0
            f1(i,j)=255;
        end
    end
end
f1=uint8(f1);
f2=zeros(I,J);
f3=zeros(I,J);
f2=uint8(f2);
f3=uint8(f3);
f1=cat(3,f1,f2,f3);
g=f1+f4;
return;
```

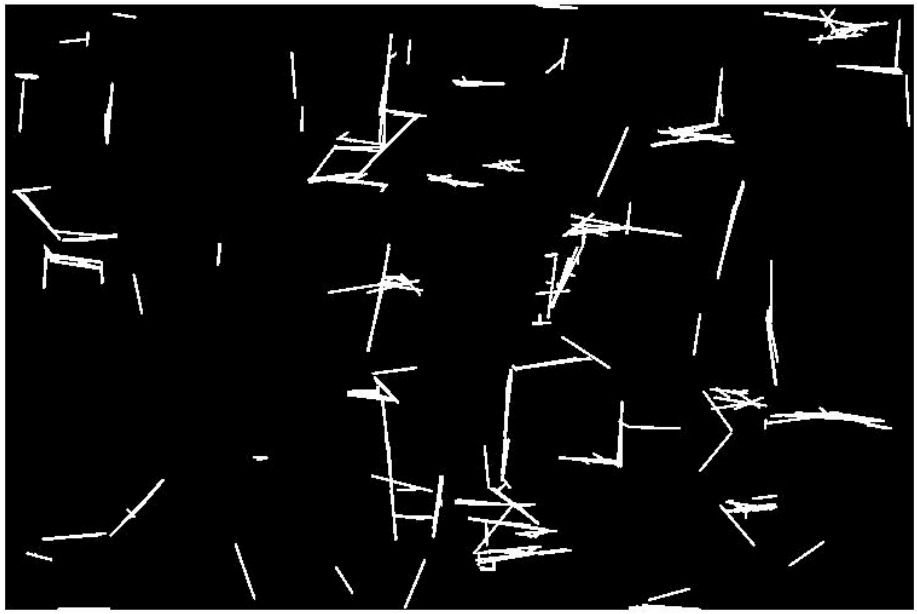
Appendix B – Some results

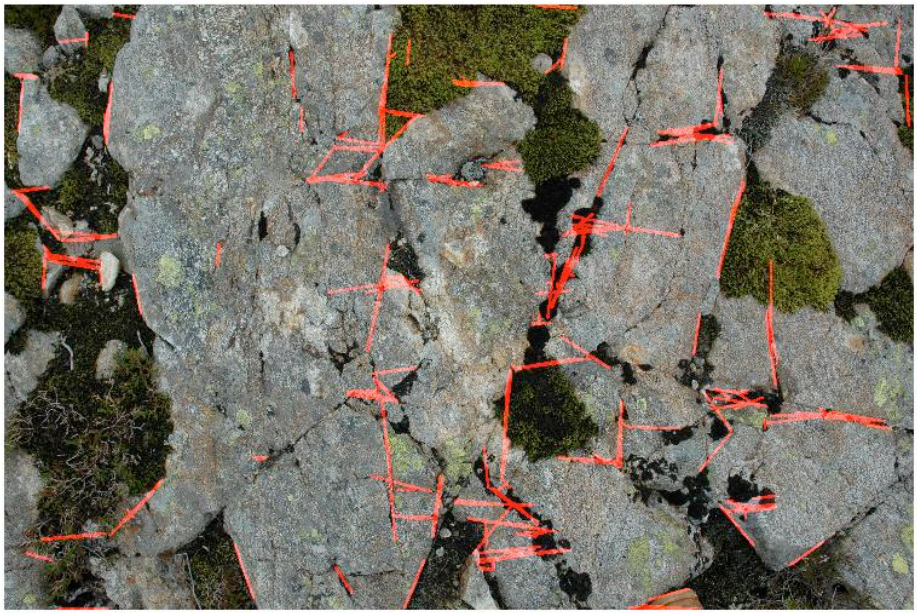
B.1

Filename: Ritland2.jpg

Area: 500

Shape: 5





B.2

Filename: DSC_8367.jpg

Area: 200

Shape: 7

B.3

Filename: DSC_8376.jpg

Area: 100

Shape: 3.5

B.4

Filename: DSC_8378.jpg

Area: 1000

Shape: 4.5