INFORMATION TO USERS

This dissertation copy was prepared from a negative microfilm created and inspected by the school granting the degree. We are using this film without further inspection or change. If there are any questions about the content, please write directly to the school. The quality of this reproduction is heavily dependent upon the quality of the original material.

The following explanation of techniques is provided to help clarify notations which may appear on this reproduction.

1. Manuscripts may not always be complete. When it is not possible to obtain missing pages, a note appears to indicate this.

2. When copyrighted materials are removed from the manuscript, a note appears to indicate this.

3. Oversize materials (maps, drawings and charts are photographed by sectioning the original, beginning at the upper left hand corner and continuing from left to right in equal sections with small overlaps.

UMI®
ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600
COMPUTER VISION AND FRACTAL DIMENSION

NAVDEEP SINGH CHADHA
Computer Science Department

APPROVED:

Bf (V. Kreinovich)
Dr. Vladik Kreinovich. Chair

Ch.
Dr. Chitta Baral

Dr. Carroll W. Johnson

Associate Vice President for Research
and Graduate Studies
To My Parents
COMPUTER VISION AND FRACTAL DIMENSION

by

NAVDEEP SINGH CHADHA, B.TECH.E.E.

THESIS
Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

Computer Science Department
THE UNIVERSITY OF TEXAS AT EL PASO
May 1994
Acknowledgements

I want to thank all my teachers who have helped me in my journey through school. I specially want to thank Dr. Vladik Kreinovich and Dr. W. C. Johnson for guiding me continuously through my work and giving me their valuable time and support whenever I needed it. I also want to thank Dr. Chitta Baral for being on my graduate committee and providing me with valuable suggestions from time to time. Special thanks goes to Prof. Raymond Bell and my friends for continuously giving me moral support. I want to thank Dr. D. Williams of the Electrical Engineering Department for letting me use their equipment. I am also thankful to Institute for Manufacturing and Material Management for providing the funding for this project and to Alcatel Co. for proving us with the printed circuit boards and all the necessary help whenever needed.

I am grateful to my parents for their everlasting encouragement, support and guidance throughout my educational career, without which it would have been impossible for me to come this far.

This thesis was submitted to my Supervising committee on December 10, 1993.
Abstract

In this thesis, we consider a new approach to computer vision problems that is based on fractal theory. As a case study we take printed circuit boards, a vital part of modern electronic technology. Due to the progress in the technology the number of components being soldered on them has increased drastically. It becomes more and more difficult for human inspectors to check the quality of the solder joint, and hence there is a need for an automated solder joint inspection system. The ability of this system should not only be to just detect the defective solder joint, but to also classify the defects so that it can be used as a feedback for the manufacturing system. At present there are systems that do it for us, but most of these systems are still not fast enough for mass production. So it is necessary to improve these systems. In the present thesis in addition to the previous numerical characteristics used for testing solder joint we have suggest another characteristic related to fractal dimension. This characteristic clearly distinguishes a solder joint with a hole in it from a good solder joint. We prove mathematically that this is (in some reasonable sense) the best possible characteristics and develop algorithm for computing its value. The inclusion of this feature resulted in $\approx 4.5\%$ reduction of the computation time for a printed circuit board. For mass manufacturing, with thousands of solders, this is a considerable time saving.
# Contents

Acknowledgements .......................................................... iv
Abstract ........................................................................... v
1 INTRODUCTION ......................................................... 1
  1.1 Computer vision .................................................. 1
  1.2 Printed circuit boards inspection: an area where computer.
      vision is a must .................................................... 2
  1.3 Formulation of the problem ...................................... 3
2 THE EXISTING COMPUTER VISION TECHNOLOGY ....... 6
  2.1 Classification of feature ......................................... 7
  2.2 Circular sub-area feature set ................................. 9
  2.3 Moment of inertia feature set ................................ 9
  2.4 Surface area feature ............................................ 14
  2.5 Gaussian curvature related features ....................... 16
3 Fractal Dimension ....................................................... 20
  3.1 Fractal dimension ................................................ 20
  3.2 Fractal dimension for solder joints ......................... 23
  3.3 Definition and the main results ............................ 26
4 ALGORITHM FOR THE COMPUTATION OF THE FRACTAL
   DIMENSION FOR SOLDER JOINTS ......................... 28
List of Tables

5.1 First 17 features from the first four feature sets calculated on five good solder joints under ideal lighting conditions. ............................ 39
5.2 Next 16 features from the first four feature sets calculated on five good solder joints under ideal lighting conditions. ......................... 40
5.3 First 17 features from the first four feature sets calculated on five good solder joints under ideal conditions. .............................. 41
5.4 Next 16 features from the first four feature sets calculated on five good solder joints under ideal conditions. .............................. 42
5.5 First 17 features from the first four feature sets calculated on five good solder joints under industrial conditions. ......................... 43
5.6 Next 16 features from the first four feature sets calculated on five good solder joints under industrial conditions. ......................... 44
5.7 First 17 features from the first four feature sets calculated on five good solder joints under industrial conditions. ......................... 45
5.8 Next 16 features from the first four feature sets calculated on five good solder joints under industrial conditions. ......................... 46
5.9 First 17 features from the first four feature sets calculated on five bad solder joints under ideal conditions. .............................. 47
5.10 Next 16 features from the first four feature sets calculated on five bad solder joints under ideal conditions. .................. 48
5.11 First 17 features from the first four feature sets calculated on five bad solder joints under industrial conditions. .......... 49
5.12 Next 16 features from the first four feature sets calculated on five bad solder joints under industrial conditions. .......... 50
5.13 The value of α that was calculated for good solder joints. ...... 51
5.14 The value of α that was calculated for solder joints with a hole in them................................................................. 51
Chapter 1

INTRODUCTION

1.1 Computer vision

Computer vision is an area of Computer Science and Computer applications in which computers are used to process visual images of real-world objects. Computer vision systems (they are also called machine vision systems) normally consist of three main parts:

- camera (that transforms light into electric signals).

- "vision box", a processor controlled device that transforms analog signals from the camera into digital signals for the computer.

- a computer itself, that processes the images.

Starting from mid 80's, computers and processors have become so powerful and cheap that the resulting machine vision systems are also relatively cheap, sufficiently reliable, and very powerful. They are therefore now actively used in manufacturing as well, in particular, in mass production. One of their main application is in
inspection and related process control. Inspection problems can be divided into two big classes:

- problems, in which the desired shape is known, and has to be followed precisely;
- problems in which reasonable deviations from the ideal shape are permissible.

Problems of first type are relatively easier to formalize, and easier to solve. There are many working systems [25] [10] [26] that solve them.

With problems of second type, the situation is more complicated, and many open problems remain. In this thesis, we will consider these problems.

As a case study, we will consider solder joint inspection. The ideal shape of a solder is known, but even if there is a slight deviation from that shape, it is still all right. Such problems are usually difficult to formalize and thus, difficult to automate. Hence, in many cases the existing technologies rely on the ability of a human inspector. Manual inspector is a weak point of a technological chain, he slows down the whole process, and the subjective character of his judgment causes errors. The area in which the necessity to automate visual inspection processes is most urgent is, solder joints inspection, because as technology advances components and related joints increase rapidly.

In this introduction we will describe the necessity of an automated (computer vision) system. We will describe briefly the existing systems, describe their drawbacks, and formulate the computer vision problem that we will be solving.

1.2 Printed circuit boards inspection: an area where computer vision is a must

Human inspectors often err. There are three possible types of errors:
• An inspector can mistakenly accept a faulty solder joint as a good one. This error can lead to the failure of the system, in which it will be used. This failure can sometimes lead to catastrophic consequences, and even if it simply causes a system to stop, it is usually extremely expensive to figure out what exactly is wrong with the system.

• An inspector can mistakenly classify good as defective. Then we waste time and money spent on producing them and thus we increase the total production cost.

• An inspector can correctly classify a solder joint as defective, but err in the nature of this defect. In this case the resulting correction of the technological process will lead us in the wrong direction. This will not only fail to solve the problem, but it will even add new defects to the existing ones.

A trained inspector can avoid these errors for some time, but after a while he gets tired, and his error rate increases. There is a need to develop an automated computer vision-based system which will overcome these drawbacks. The goals of this system are:

• to replace the human operators and automate the visual inspection system.

• to analyze the results obtained from this inspection system and utilize them to automatically control the manufacturing process.

1.3 Formulation of the problem

Crudely speaking, one can divide all possible applications of printed circuit boards into two main groups:
• advanced systems in which reliability is so critical that it justifies the increase in the expenses of the manufacturing system. Some of these applications include military applications and space exploration.

• mass production systems, such as TV industry in which the production cost is a major concern.

There already exists some systems which give us information reliable enough to determine whether the joint is good or bad and if is bad then what exactly is wrong. These methods include X-ray [25] [10] [26] or infra-red screening [9], laser scanning [17], etc. These systems are expensive, require hazardous environment and are thus unusable for mass production.

So the main remaining problem is to design a computer vision system for solder joint inspection in mass production.

As we have just mentioned, we cannot use fancy methods like X-rays, infra-red, etc for mass manufacturing. So we can use only a visual image. Such computer vision systems exist. The best of these systems is described in [30]. This system makes fewer errors than a human inspector.

However, for mass manufacturing with thousands of boards, and for a board that contains hundreds of solder joint, even 1% error is too much.

So, the goal of this thesis is to find a method to improve the existing vision systems. Right now one of the major concerns in these automated vision systems is the computation time. As compared to the human operators these systems still take a lot of time, which in turn slows down the manufacturing process. The main purpose of this thesis is to reduce this computation time.

The structure of this thesis is as follows. In Chapter 2, we describe the computer vision technology that is used in the existing systems. In Chapter 3, we describe our idea. In Chapter 4, we present corresponding algorithm. In Chapter 5, we
describe related experiments and their results.
Chapter 2

THE EXISTING COMPUTER VISION TECHNOLOGY

A picture coming from a vision box consists of \( \approx 10^6 \) pixels. We are talking about a mass manufacturing system, where in a few minutes we have to examine the entire board with hundreds of joints. So, to analyze a joint, we have only a few seconds. Even with the most powerful PC's we can make only a few computations with each pixel intensity. Usually, the image processing is done as follows:

- First, we process the image several times to generate several characteristics of the image (called "features"). This part takes practically all the time.

- Second, we process these features to decide whether a solder joint is good or bad.

Features are selected very carefully, since adding irrelevant or redundant features can increase the processing time of the system.

In general, a set of features must satisfy the following requirements:

**Discrimination**: features should be different for objects belonging to different classes.
Reliability: features should be similar for all the objects in the same class.

Independence: various features should be uncorrelated with each other.

Small number: a small number of features will make the classification process simple and fast.

2.1 Classification of feature

An ideal solder joint has a smooth reflecting surface that covers the center with a necessary solder layer, and quickly decreases as we go away from the center. Many things can go wrong with a solder joint. We may have not enough solder in the center (this is called a hole). We may also instead of a more or less homogeneous and smooth distribution of solder, have spots where solder layer is thicker and spots where it is not.

So, to determine the quality of a solder joint, we can use the following criteria:

- First, we check that in different areas (center, circle around the center, and an area far from center) there is exactly the right amount of solder, and that this amount is distributed more or less uniformly. To check that, we can use the total intensity in these areas, and the mean square deviations of intensity. These characteristics are the easiest to compute. They form the first set of features.

- Instead of dividing the image into subareas, we can use global criteria that show whether all the solder is located in the center or there is lots of unnecessary extra solder outside the center. In mechanics, the degree with which the masses are located near the center is described by the moment of inertia: the smaller it is, the closer are the masses. So, this same formula can be (and
is) used for images as well.

- When a surface is not smooth, its total surface area increases. So, the total area can serve as a feature. This feature was proposed in [30]

- When there is too much solder, the surface becomes almost planar, so its curvature is close to 0. When the size of the solder area is too small, the curvature becomes large. So, the curvature can also serve as a feature. This idea was also proposed in [30]

In the following sections, we will describe these features in detail. But before we do that, we need to make one remark.

The intensity of a pixel depends not only on the corresponding point on a surface, but also on the intensity of the illumination and on the amplification of the vision board. These two parameters can change drastically (especially in real life manufacturing environment). So, to avoid out possible dependency on them, we will normalize pixel intensities by dividing them by the biggest.

Let's describe this normalization in mathematical terms. Let's denote the number of horizontal pixels coordinates by \( n_x \), and the number of vertical pixels coordinates by \( n_y \). x-coordinates will be thus numbered \( 0, 1, ..., n_x - 1 \) and y-coordinates \( 0, 1, ..., n_y - 1 \). The intensity in a point with coordinates \( i \) and \( j \) will be denoted by \( f(i, j) \). In these terms, normalization means that we compute

\[
f_{\text{max}} = \max_{i,j} f(i, j)
\]  

and then

\[z(i, j) = \frac{f(i, j)}{f_{\text{max}}}
\]  

(2.1)
2.2 Circular sub-area feature set

First of all, we can compute the average intensity

\[ a_0 = \mu = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} z(i, j) \]  

(2.3)

and the variance of intensity

\[ a_1 = \sigma^2 = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} (z(i, j) - \mu)^2 \]  

(2.4)

These two features do not change if we shift or rotate the image.

In addition, as we mentioned, it is reasonable to divide the image into three sub-areas: outer-ring, middle-ring, and lead area. For each of them, we can compute the corresponding average intensity and variation

\[ a_2 = \mu_1 \]  

(2.5)

\[ a_3 = \sigma^2_1 \]  

(2.6)

\[ a_4 = \mu_2 \]  

(2.7)

\[ a_5 = \sigma^2_2 \]  

(2.8)

\[ a_6 = \mu_3 \]  

(2.9)

\[ a_7 = \sigma^2_3 \]  

(2.10)

In these formulas, the subscript 1, 2, 3 of \( \mu \) and \( \sigma \) stand for outer-ring, middle-ring, and lead area respectively. So, we have 8 features \( a_0, ..., a_7 \).

2.3 Moment of inertia feature set

To compute the moment of inertia, we must be able to use distances. To simplify computations, let us (following [30]) measure distances not in inches or centimeters.
but in artificial units, in which the size of the area that we are analyzing is exactly 1. So, both $x$ and $y$ will go from $[-0.5, +0.5]$ to $[-0.5, +0.5]$ (so that the center is exactly at the point $(0,0)$).

In these units, a pixel $(i, j)$ has coordinates
\[ x(i) = \frac{1 + 2i - n_x}{2n_x} \quad (2.11) \]
\[ y(i) = \frac{1 + 2j - n_y}{2n_y} \quad (2.12) \]

The centralized inertia tensor of a mass distribution relative to a given coordinate system is given by \[ [I] = \int \int \rho(r)(r^T r[1] - rr^T)dx dy dz \quad (2.13) \]
where
- $r = (x, y, z)$ is the 3-D coordinate vector relative to the CS coordinate system,
- $[1] = 3 \times 3$ identity matrix,
- superscript $T$ denotes transposition,
- boldface denotes vectors,
- brackets denote matrices,
- $\rho(r)$ denotes density function (i.e. the mass distribution)

and center of mass vector for a mass distribution $\rho(x)$ is given by
\[ r_{cm} = \int \int \int r \rho(r) dx dy dz \quad (2.14) \]

Each pixel in normalized image representation can be viewed as a block with density $2z(i, j)$ and sizes $\Delta x = 1/n_x$ and $\Delta y = 1/n_y$. So, (2.14) turns into
\[ x_{cm} = \frac{1}{2\mu} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} m(i, j)x(i) \quad (2.15) \]
\[ y_{cm} = \frac{1}{2\mu} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} m(i, j)y(i), \quad (2.16) \]

where \( m(i, j) = 2\Delta x\Delta yz(i, j) \)

The inertia tensor for a uniform density rectangular block of dimensions \( a \times b \times c \) and mass \( M \) is well-known:

\[
[I_{\text{block}}(a, b, c, M)] = \begin{pmatrix}
\frac{M}{12} & 0 & 0 \\
0 & c^2 + a^2 & 0 \\
0 & 0 & a^2 + b^2
\end{pmatrix}. \quad (2.17)
\]

The total inertia tensor can be calculated by summing the appropriate inertia tensors of the individual pixels blocks:

\[
[I_{\text{tot}}] = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} [I_{\text{block}}(\Delta x, \Delta y, 2z(i, j), m(i, j))] + m(i, j)(v(i, j)^T v(i, j)[1] - v(i, j)v(i, j)^T)
\]

where \( v(i, j) = (x_{cm} - x(i), y_{cm} - y(j), 0) \)

Therefore

\[
m_0 = I_{xx} = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} (i_{xx}(i, j) + m(i, j)(y(i) - y_{cm})^2) \quad (2.19)
\]

\[
m_1 = I_{yy} = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} (i_{yy}(i, j) + m(i, j)(x(i) - x_{cm})^2) \quad (2.20)
\]

\[
m_2 = I_{zz} = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} (m(i, j)((x(j) - x_{cm})^2 + (y(j) - y_{cm})^2)) \quad (2.21)
\]

\[
m_3 = [I_{xy}] = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} (m(i, j)(x(i) - x_{cm})(y(j) - y_{cm})) \quad (2.22)
\]

where \( i_{xx}(i, j), i_{yy}(i, j), i_{zz}(i, j) \) are the moment of the block on pixel \((i, j)\). They are given by the following formulas:

\[
i_{xx}(i, j) = \frac{m(i, j)}{12} \left( \left( \frac{1}{n_y} \right)^2 + (2z(i, j))^2 \right) \quad (2.23)
\]
\( i_{yy}(i, j) = \frac{m(i, j)}{12} \left( (2z(i, j))^2 + \left( \frac{1}{n_x} \right)^2 \right) \) \hspace{1cm} (2.24)

Also,

\( m(i, j) = \left( \frac{1}{n_x} \right) \left( \frac{1}{n_y} \right) (2z(i, j)) \) \hspace{1cm} (2.25)

Since \( i_{zz} \) is relatively small, it is neglected. Other features based upon the combinations of these moments are introduced in [3]:

\[ m_4 = |m_3| \] \hspace{1cm} (2.26)

\[ m_5 = \frac{m_0 + m_1 + m_2}{3} \] \hspace{1cm} (2.27)

\[ m_6 = \frac{m_0 + m_1}{2m_2} \] \hspace{1cm} (2.28)

\[ m_7 = m_0m_1 + m_1m_2 + m_2m_0 \] \hspace{1cm} (2.29)

\[ m_8 = \sqrt{\frac{m_0^2 + m_1^2}{2}} \] \hspace{1cm} (2.30)

\[ m_9 = \sqrt{\frac{m_0^2 + m_1^2 + m_2^2}{3}} \] \hspace{1cm} (2.31)

\[ m_{10} = \frac{m_0m_1}{m_2} \] \hspace{1cm} (2.32)

\[ m_{11} = \left| \frac{m_3}{m_2} \right| \] \hspace{1cm} (2.33)

\[ m_{12} = \frac{m_0 - m_1}{m_2} \] \hspace{1cm} (2.34)
\[ m_{13} = \frac{m_0 - m_1}{m_3} \quad (2.35) \]
\[ m_{14} = \left| \frac{m_3}{m_0 - m_1} \right| \quad (2.36) \]
\[ m_{15} = | m_0 - m_1 | \quad (2.37) \]
\[ m_{16} = \frac{(m_0^2 + m_1^2)}{2m_0m_1} \quad (2.38) \]
\[ m_{17} = \frac{2m_2}{\sqrt{m_0^2 + m_1^2}} \quad (2.39) \]
\[ m_{18} = | (m_0 - m_1)(m_1 - m_2)(m_2 - m_0) | \quad (2.40) \]
\[ m_{19} = \left| \frac{m_0^2 - m_1^2}{m_3^2} \right| \quad (2.41) \]

The two-dimensional moment features [13] are given by:

\[ m_{20} = I_x = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} \frac{1}{n_xn_y}z(i,j)(y(j) - y_{cm})^2 \quad (2.42) \]
\[ m_{21} = I_y = \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} \frac{1}{n_xn_y}z(i,j)(x(j) - x_{cm})^2 \quad (2.43) \]

The maximum and minimum moments along the principal axes are computed as follows [13]:

\[ m_{22} = \frac{I_x + I_y + \sqrt{(I_x - I_y)^2 + 4I_{xy}^2}}{2} \quad (2.44) \]
\[ m_{23} = \frac{I_x + I_y - \sqrt{(I_x - I_y)^2 + 4I_{xy}^2}}{2} \quad (2.45) \]
2.4 Surface area feature

The surface consists of a quadrilateral, formed by four adjacent pixels. The quadrilateral can be divided into two triangles. The sum of the areas of the two triangles is the area of the quadrilateral. The sum of the areas of all quadrilateral is area of the image surface. For an \( n \times n \) constant intensity image, the surface area is \((n - 1)^2\). To normalize the feature we divide the computed surface area by the constant intensity surface area to normalize this feature.

For four adjacent pixels in the image: \( f(i, j), f(i+1, j), f(i, j+1), f(i+1, j+1) \), the four 3-D points \( \{p_1, p_2, p_3, p_4\} \) are given by the vectors:

\[
\begin{align*}
p_1 &= [i \ j \ f(i, j)]^T \\
p_2 &= [i + 1 \ j \ f(i + 1, j)]^T \\
p_3 &= [i \ j + 1 \ f(i, j + 1)]^T \\
p_4 &= [i + 1 \ j + 1 \ f(i + 1, j + 1)]^T
\end{align*}
\]

Now, consider the two triangles of the quadrilateral separately. Let \( \vec{p}_1, \vec{p}_2, \vec{p}_3 \) determine the first triangle, then, two difference vectors are given by:

\[
\begin{align*}
\vec{d}_1 &= \vec{p}_2 - \vec{p}_1 \\
\vec{d}_2 &= \vec{p}_3 - \vec{p}_1
\end{align*}
\]  

(2.46) 

(2.47)

The unit normal to the triangle, which is used to define the direction, is computed as follows:

\[
\vec{n}_z = \frac{\vec{d}_1 \times \vec{d}_2}{||\vec{d}_1 \times \vec{d}_2||}
\]  

(2.48)

We define \( \vec{d}_1 \) to define the \( x \) direction:

\[
\vec{n}_x = \frac{\vec{d}_1}{||\vec{d}_1||}
\]  

(2.49)

The \( y \) direction is then determined by \( \vec{n}_x \) and \( \vec{n}_y \) as follows:

\[
\vec{n}_y = \vec{n}_z \times \vec{n}_x
\]  

(2.50)
Let $A_{123}$ denote the area of this triangle. The length of the base $\vec{p}_1\vec{p}_2$ of the triangle formed by the points $\vec{p}_1, \vec{p}_2, \vec{p}_3$ is equal to $|\vec{d}_1|$ i.e. to the magnitude of $\vec{d}_1$ in $x$ direction, i.e. $(\vec{n}_x^T \cdot \vec{d}_1)$. The distance between the point $\vec{p}_3$ and the line $\vec{p}_1\vec{p}_2$ is given by the magnitude of $\vec{d}_2$ in the $y$ direction, which is $(\vec{n}_y^T \cdot \vec{d}_2)$. This distance is the height of this triangle.

Thus the area of this triangle can be computed [30] as $\frac{1}{2}$ base $\times$ height, i.e. by computing two vector inner products:

$$A_{123} = \frac{1}{2} (\vec{n}_x^T \cdot \vec{d}_1)(\vec{n}_y^T \cdot \vec{d}_2) \quad (2.51)$$

Similarly, we can compute the area $A_{234}$ the second triangle with sides represented by the vectors $\vec{p}_4, \vec{p}_3, \vec{p}_2$. The area associated with the four adjacent pixels with intensities $f(i, j)$, $f(i + 1, j)$, $f(i, j + 1)$, $f(i + 1, j + 1)$ is given by

$$a(i, j) = A_{123} + A_{234} \quad (2.52)$$

and the total normalized surface area by [30]

$$A = \frac{1}{(n_x - 1)(n_y - 1)} \sum_{i=0}^{n_x-2} \sum_{j=0}^{n_y-2} a(i, j) \quad (2.53)$$
A gray level surface is flat for \( A = 1 \) and increases as the surface contains more and more undulations.

## 2.5 Gaussian curvature related features

Let's consider two neighbouring points on the grey level surface: with position vectors \( S = (x, y, f(x, y)) \) and \( S + dS = (x + dx, y + dy, f(x + dx, y + dy)) \), correspond to the parameter values \( x, y \) and \( x + dx, y + dy \) respectively. Then

\[
dS = \frac{\partial S}{\partial x} dx + \frac{\partial S}{\partial y} dy
\]

or, using the notation \( S_x \) and \( S_y \) for partial derivatives:

\[
dS = S_x dx + S_y dy
\]

Since the two points are adjacent on a curve passing through them, the length \( ds \) of the element of the arc joining them is equal to their actual distance \( |dS| \). Thus

\[
ds^2 = (S_x dx + S_y dy)^2
\]

\[
ds^2 = S_x^2 dx^2 + 2S_x S_y dx dy + S_y^2 dy^2.
\]

If then we write \( E = S_x^2, F = S_x S_y, G = S_y^2 \) we have the formula

\[
ds^2 = Edx^2 + 2F dx dy + G dy^2
\]

The quantities denoted by \( E, F, G \) are called the \textit{fundamental magnitudes of the first order} and these can be calculated by substituting the values of \( S_x \) and \( S_y \). Thus

\[
E = 1 + f_x^2
\]

\[
F = f_x f_y
\]
Similarly, \( g = |S_x \times S_y|^2 = S_x^2 S_y^2 - (S_x \times S_y)^2 = EG - F^2 \). so

\[
g = EG - F^2 = 1 + f_x^2 + f_y^2
\]  

(2.62)

The second derivatives of \( S \) with respect to \( x \) and \( y \) are equal to

\[
S_{xx} = \frac{\partial S}{\partial x^2} = (0.0, f_{xx})
\]

\[
S_{xy} = \frac{\partial S}{\partial x \partial y} = (0.0, f_{xy})
\]

\[
S_{yy} = \frac{\partial S}{\partial y^2} = (0.0, f_{yy})
\]

The fundamental magnitudes of the second order are the parts of these vectors in the direction of the normal to the surface. They will be denoted by \( L, M, N \). Thus

\[
L = n \cdot S_{xx}
\]

\[
M = n \cdot S_{xy}
\]

\[
N = n \cdot S_{yy}
\]

, where \([4]\)

\[
n = \frac{S_x \times S_y}{|S_x \times S_y|} = \frac{S_x \times S_y}{\sqrt{g}}
\]

. We may express \( L, M, N \) in terms of scalar triple products of vectors. Normally,

\[
[S_x, S_y, S_{xx}] = (S_x \times S_y) \cdot S_{xx} = f_{xx}
\]  

(2.63)

On the other hand,

\[
(S_x \times S_y) \cdot S_{xx} = \sqrt{g} n \cdot S_{xx} = \sqrt{g} L
\]  

(2.64)

Similarly

\[
[S_x, S_y, S_{xy}] = S_x \times S_y \cdot S_{xy} = \sqrt{g} n \cdot S_{xy} = \sqrt{g} M = f_{xy}
\]  

(2.65)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
and

\[ [S_x, S_y, S_{yy}] = S_x \times S_y \cdot S_{yy} = \sqrt{g} \mathbf{n} \cdot S_{yy} = \sqrt{g} N = f_{yy}. \]  

(2.66)

Thus from these equations we have

\[ L = \frac{f_{xx}}{\sqrt{g}} \]  

(2.67)

\[ M = \frac{f_{xy}}{\sqrt{g}} \]  

(2.68)

\[ N = \frac{f_{yy}}{\sqrt{g}} \]  

(2.69)

The Gaussian curvature function of a smooth surface is defined as [30]:

\[ K = \frac{b}{g} = \frac{f_{xx} f_{yy} - f_{xy}^2}{(1 + f_x^2 + f_y^2)^2} \]  

(2.70)

where

\[ b = LN - M^2 \]  

(2.71)

The mean curvature function of a smooth surface is given by

\[ H = \frac{EN + GL - 2FM}{2g} \]  

(2.72)

It can be shown that

- (1) for \( K > 0 \) all points \((x, y)\) where the local neighbourhood of the surface has an ellipsoidal shape.

- (2) for \( K < 0 \) all points \((x, y)\) where the local neighbourhood of the surface has a saddle shape.

- (3) for \( K = 0 \) all points \((x, y)\) where the local neighbourhood of the surface has is flat or cylindrical.
We can also compute the average Gaussian curvature, and the average mean curvature. Thus compute the values $b(i,j), g(i,j), K(i,j)$ and $H(i,j)$ for every pixel $(i,j)$.

\[
c_0 = K_{avg} = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} K(i,j)
\]

\[
c_1 = H_{avg} = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} H(i,j)
\]

Some of the other features for measuring surface topology [7] [21] are also employed.

\[
c_2 = b_{avg} = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} b(i,j)
\]

\[
c_3 = g_{avg} = \frac{1}{n_x n_y} \sum_{i=0}^{n_x-1} \sum_{j=0}^{n_y-1} \sqrt{g(i,j)}
\]
Chapter 3

Fractal Dimension

The gray level surface of the solder joint is not smooth. By a smooth surface, we mean that at every point on this surface there exits a tangent plane. There are many shapes in nature that are not smooth. While studying these forms Mandelbrot developed a new notion of a fractal. The fractal is a general notion that includes not only non-smooth curves and surfaces, but also disconnected “dust”, and others so oddly shaped that there are no good terms for them. The name “fractal” comes from the word “fraction”. Crudely speaking fractals are sets of non-integer (fractional) dimension. Let us explain what it means.

A typical example of fractal curves would be the path followed be a small particle in water or other fluid. The particles follow a perfectly irregular motion. They go, stop, start again, mount, descend, mount again, without least tending toward immobility. This motion of the particles is called a Brownian motion.

3.1 Fractal dimension

To illustrate the notion of fractal dimension let us consider a realistic problem. Consider a segment of a road of length $L$. We have to place a number of McDonalds...
on this segment in such a way that there is one within \( \delta \) distance from any point on this segment of road. Since building McDonalds costs money, we would like to satisfy this condition with the smallest possible number of McDonalds. Let's first figure out where to place the first McDonalds. There must be a McDonald at a distance \( \leq \delta \) from the starting point. It makes no sense to build it at a distance \( < \delta \), because then that McDonald will also unnecessarily the part before the starting point. So, we want to design the smallest possible number of McDonalds. we must place the first one at a distance \( \delta \). This McDonald serves the area from 0 (= starting point) to the point \( 2\delta \).

To serve the point that are to the right of \( 2\delta \), we must have another McDonald. For the same reason of minimality, we must place it exactly at a distance \( \delta \) from this \( 2\delta \) point, i.e., at a point \( 3\delta \). This second McDonald serves an area from \( 2\delta (= 3\delta - \delta) \) to \( 4\delta (= 3\delta + \delta) \). Similarly, a third McDonald (if it is necessary) must be placed at a point \( 5\delta \), and serve an area between \( 4\delta \) and \( 6\delta \). etc.

So, each McDonald serve an area of size \( 2\delta \). The total length of the road segment is \( L \). So the number of McDonalds that cover the intended area is proportional to \( L/\delta \):

\[
N(\delta) \propto \frac{L}{\delta}
\]  
(3.1)

Now consider a square of size \( L \times L \). In this case, every McDonald serves the area that is a circle of radius \( \delta \). The area of this circle is \( \pi \delta^2 \). So, if we have \( N \) McDonalds, we will cover the area that is at most \( N \pi \delta^2 \). The total area of the square is \( A = L^2 \). So, to cover the entire square, we must take such an \( N \) that \( N \pi \delta^2 \geq A \); i.e., \( N \geq \frac{A}{\pi \delta^2} \). We can actually place \( N \approx \text{const.} \frac{1}{\delta^2} \) McDonalds and cover the entire square. So, in this case,

\[
N(\delta) \propto \frac{A}{\delta^2}
\]  
(3.2)
Figure 3.1: Number of McDonalds (represented by solid circles) needed on a square of $L \times L$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In a 3-D case, e.g., when we want to cover a cube of size $L$, each McDonald serves the body of sphere of radius $\delta$. Its volume is $\frac{4}{3}\pi\delta^3$. The total volume is $V = L^3$, so the smallest possible number of McDonalds must be $\geq \frac{V}{\frac{4}{3}\pi\delta^3}$. Like in 2-D case, we can actually place $\approx \text{const}\frac{V}{\delta^3}$ McDonalds and cover the area, so in this case,

$$N(\delta) \propto \frac{V}{\delta^3} \quad (3.3)$$

It has been proven that for smooth curves (i.e., 1-D sets) the formula (3.1) is true, for smooth surfaces (i.e., for 2-D sets) the formula (3.2) holds and for 3-D bodies, the formula (3.3) holds.

So, if for some set

$$N(\delta) \propto \frac{G}{\delta^D} \quad (3.4)$$

for some constants $G$ and $\delta$, and $D$ is an integer, then $D$ coincides with the dimension of that set. So, it is natural to propose the following generalization: if for some set, formula (3.4) holds for a non-integer $D$, this value $D$ is also called a dimension, of a fractal dimension.

For example, a trajectory of a Brownian motion is of dimension $D = 3/2$. Sets of fractal dimension are called fractals.

### 3.2 Fractal dimension for solder joints

Brownian motion is an example of a random process, in the sense that the initial trajectory determines only the probabilities of different possible further motions. Many other random processes result in a fractal.

Since the surface of a solder joint looks like random, let’s try to describe it as such. One of the major ways to describe a random process is to describe its autocorrelation function, or to describe how different are the intensities in neighbouring
points. If we take points \( \bar{x} \) and \( \bar{x} + \bar{s} \) at a distance \( s = |\bar{s}| \), then the difference between the intensities is \( I(\bar{x} + \bar{s}) - I(\bar{x}) \). The mean square value of this difference is

\[
E[(I(\bar{x} + \bar{s}) - I(\bar{x}))^2].
\] (3.5)

Let’s assume (for simplicity) that this mean square value depends only on the distance \( s = |\bar{s}| \), i.e. that

\[
f(s) = E[(I(\bar{x} + \bar{s}) - I(\bar{x}))^2]
\] (3.6)

We need to find a function that best approximates \( f(s) \) (according to some optimality criterion). The intensity \( I(\bar{x}) \) depends on:

- solder joint
- illumination of the scene

If we change the illumination, then \( I(\bar{x}) \) will change to \( \bar{I}(\bar{x}) = kI(\bar{x}) \), where \( k \) is a constant. Hence from (3.6) we have \( \bar{f}(s) = k^2f(s) \). So, if \( f(s) \) is a good approximation for some case, then so is \( kf(s) \).

So although we spoke about choosing a function \( f(s) \), it is better to speak about choosing a family of function \( f \). It is reasonable to suggest that if a function \( f(z) \) belongs to this family, then this family must contain a function \( \bar{f}(z) = Cf(z) \) for every positive real number \( C \).

What family should we choose? The criterion to choose the family may be computational simplicity, average approximation error, or something else. In mathematical optimization problems, numeric criteria are most frequently used. where to every family we assign some value expressing its performance and choose a family for which this value is maximal. However, it is not necessary to restrict ourselves to such criteria only. For e.g., if we have several different families that give us the
same average approximate error $E$, we can choose between them the one that has the minimal computational complexity $C$. In this case, the actual criterion that we use to compare two families is not numeric, but more complicated: a family $\phi_1$ is better than the family $\phi_2$ if and only if either $E(\phi_1) < E(\phi_2)$, or $E(\phi_1) = E(\phi_2)$ and $C(\phi_1) < C(\phi_2)$. A criterion can be even more complicated. What a criterion must do is to allow us, for every pair of families, to tell whether the first family is better with respect to this criterion (we will denote it by $\phi_2 < \phi_1$) or the second is better ($\phi_1 < \phi_2$) or these families have the same qualities in the sense of this criterion ($\phi_1 \sim \phi_2$).

Moreover the criterion that we choose should be consistent. For e.g. if $\phi_1 < \phi_2$ and $\phi_2 < \phi_3$ then $\phi_1 < \phi_3$.

The criterion must be final in the sense that it must choose a unique optimal family, i.e. a family that is better with respect to this criterion then any other family. If a criterion does not choose any family at all, it is of no use. If several different families are "the best" according to this criterion, we still have a problem choosing one of them. Therefore, we need some additional criterion for that choice.

The criterion must be reasonably invariant. For e.g. in our case

$$f(|\bar{s}|) = E[(I(\bar{x} + \bar{s}) - I(\bar{x}))^2]$$

The function is dependent on $|\bar{s}|$. As mentioned earlier $|\bar{s}|$ is the distance between any two pixel points in an image. Once the image has been shot the value of $|\bar{s}|$ between any two pixel points is a constant number, that depends upon the resolution of the image. The intensity $I$ at $(\bar{x} + \bar{s})$ is going to be same even if we choose to represent the value of $|\bar{s}|$ in mm, cm or km. Suppose we have two functions $f(s)$ and $\tilde{f}(s)$, the results of applying $f(s)$ is better then $\tilde{f}(s)$ when $|\bar{s}|$ is in mm (or the family $\phi = \{Cf(s)\}$ was better than the family $\tilde{\phi} = \{C\tilde{f}(s)\}$).

Now since $f$ and $\tilde{f}$ do not depend upon the units of $|\bar{s}|$ it is reasonable to expect
that when we choose the units of $|\bar{s}|$ is cm the results of applying $f$ will still be better than applying $\tilde{f}$.

3.3 Definition and the main results

By one dimensional family we mean the set of functions $\{Cf(s)\}$, where $f(s)$ is a fixed function of one variable, $C$ runs over all the positive real numbers. The set of all one-dimensional families will be denoted by $S_1$.

A pair of relation $(<,\sim)$ is called consistent [32] if it satisfies the following conditions:

1. If $a < b$ and $b < c$ then $a < c$.
2. $a \sim a$.
3. If $a \sim b$ the $b \sim a$.
4. If $a \sim b$ and $b \sim c$ then $a \sim c$.
5. If $a < b$ and $b \sim c$ the $a < c$.
6. If $a \sim b$ and $b < c$ then $a < c$.
7. If $a < b$ then $b < a$ or $a \sim b$ is impossible.

Assume a set $A$ is given. Its elements will be called alternatives. By an optimality criterion we mean a consistent pair $(<,\sim)$ of relations on the set $A$ of all alternatives. If $b < a$, we say that $a$ is better than $b$: if $a \sim b$, we say that the alternatives $a$ and $b$ are equivalent with respect to this criterion. We say that an alternative $a$ in optimal (or best) with respect to a criterion $(<,\sim)$ if for every other alternative $b$ either $b < a$ or $a \sim b$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
We say that a criterion is it final if there exists an optimal alternative and this optimal alternative is unique.

By a **result of unit change** in a function \( f(z) \) to a unit that is \( c > 0 \) times smaller we mean a function \( f_c(z) = f(cz) \). By the **result of a unit change** in a family \( \phi \) by \( c > 0 \) we mean the set of all the functions that are obtained by this unit change from \( f \in \phi \). This result is denoted by \( \phi_c \). An optimality criterion on \( S_1 \) is **unit invariant** if for every two families \( \phi \) and \( \phi \), and for every number \( c > 0 \) the following two conditions are true:

1. If \( \phi \) is better than \( \phi \) in the sense of this criterion (i.e., \( \phi < \phi \)), then \( c\phi < c\phi \).
2. If \( \phi \) is equivalent to \( \phi \) in the sense of this criterion (i.e., \( \phi \sim \phi \)), then \( c\phi \sim c\phi \).

**Theorem:** If a family \( \phi \) is optimal in the sense of some optimality criterion that is final and unit-invariant, then every function \( f(s) \) from \( \phi \) is equal to \( f(|s|) = C|s|^\alpha \) for some constant \( \alpha \).

The proof of this theorem is given in [32].

Hence we in our case we can use \( C|\bar{s}|^\alpha \) as the optimal family to approximately represent

\[
E[(I(\bar{x} + \bar{s}) - I(\bar{x}))^2].
\]

Hence

\[
E[(I(\bar{x} + \bar{s}) - I(\bar{x}))^2] = C|\bar{s}|^\alpha
\]

This expected value can be calculated by the function

\[
f(s) = Cs^\alpha.
\]
Chapter 4

ALGORITHM FOR THE COMPUTATION OF THE FRACTAL DIMENSION FOR SOLDER JOINTS

4.1 Algorithm

This algorithm inputs a 2-D array that contains the intensity values $I(x) = I(x_1, x_2)$ in different pixels, and returns the corresponding value $\alpha$, i.e., the parameter for which for every $\delta$, the following approximate inequality holds:

$$E[(I(\bar{x} + \delta) - I(\bar{x}))^2] \approx C|\delta|^\alpha$$

(4.1)

We assume (as usual in computer graphics) that the first coordinate $x_1$ of a pixel goes from 0 to $N_p$ (so it can take values 0, 1, 2, ..., $N_p$), and that the second coordinate $x_2$ takes the values 0, 1, 2, ..., $N_p$. 

28
In our program, we set $N_p = 31$ (i.e., we consider a $32 \times 32$ image) because this is what we can get from the actual vision system.

Our algorithm will consist of two main parts. First, we will try to estimate the left-hand side of an approximate equality (4.1) from the given image and then, we will estimate $\alpha$.

To estimate the mathematical expectation in the left-hand side, we can simply take all possible vectors $\bar{s} = (s_1, s_2)$, where $s_1 = 0, 1, 2, \ldots, N_p$, and $s_2 = 0, 1, 2, \ldots, N_p$, and compute the value

$$A(\bar{s}) = \frac{1}{N(\bar{s})} \sum_{\bar{x}} (I(\bar{x} + \bar{s}) - I(\bar{x}))^2$$

where the sum is taken over all integer-valued vectors $\bar{x}$ for which both $\bar{x}$ and $\bar{x} + \bar{s}$ are meaningful pixel vectors (i.e., for which $0 \leq x_1 \leq N_p$, $0 \leq x_2 \leq N_p$, $0 \leq x_1 + s_1 \leq N_p$, and $0 \leq x_2 + s_2 \leq N_p$), and by $N(\bar{s})$, we denoted the number of all such vectors $\bar{x}$. Thus, we use an arithmetic average to approximate the mathematical expectation.

This is a well-known way to estimate the mathematical expectation of a random variable (in our case, of $(I(\bar{x} + \bar{s}) - I(\bar{x}))^2$) from a given sample, and it is known that the bigger this sample, the better the approximation (the accuracy of this approximation is $\sim \frac{1}{\sqrt{N}}$), where $N$ is a size of a sample.

We are dealing with totally $32 \times 32 = 1024$ points. $\bar{x}$ and for a given $\bar{s}$, not for all of them, $\bar{x} + \bar{s}$ is a meaningful pixel vector. Therefore, we often have only a few points in the sum (4.2), and thus, this sum is a very poor approximation to the mathematical expectation.

For example, for $\bar{s} = (0, N_p)$, if $x_2 > 0$, then $x_2 + s_2 > N_p$ and so $\bar{x} + \bar{s}$ is not a meaningful vector. Therefore, the only case when $\bar{x} + \bar{s}$ is a meaningful vector is when $x_2 = 0$. There are exactly $N_p = 32$, such vectors. For $\bar{s} = (1, N_p)$, we have a similar restriction $x_2 = 0$ and an additional restriction $x_1 + s_1 = x_1 + 1 \leq N_p$, which
leads to $x_1 \leq N_p - 1 = 31$. So for this $\bar{s}$, $N(\bar{s}) = 31$. Similarly, $N((2,32)) = 30$, $N((3,32)) = 29$, ..., $N((16,32)) = 16$, ..., $N((32,32)) = 1$. These numbers are not very large, and the corresponding accuracies are not very good (e.g., $1/\sqrt{16} = 1/4 = 25\%$). How to overcome this difficulty?

To do that, instead of computing the values for a given vector $\bar{s}$, we will add the values that correspond to 4 vectors

$$(s_1, s_2), (s_1, -s_2), (-s_1, s_2), (-s_1, -s_2)$$

which have the same value of $|\bar{s}| = \sqrt{s_1^2 + s_2^2}$. In other words, we take $s_1 \geq 0$ and $s_2 \geq 0$, and compute the average

$$B(\bar{s}) = \frac{1}{N_s(\bar{s})} \sum_{\pm} I(x_1 \pm s_1, x_2 \pm s_2) - I(x_1, x_2)^2 \quad (4.3)$$

where the sum is taken over all 4 possible combinations of signs $\pm$. and over all integer-valued vectors $\bar{x}$ for which both $\bar{x} = (x_1, x_2)$ and $(x_1 \pm s_1, x_2 \pm s_2)$ are meaningful pixel vectors. and by $N_s(\bar{s})$ ($s$ stands for "sum"), we denoted the total number of such pairs of vectors $\bar{x} = (x_1, x_2)$ and $(x_1 \pm s_1, x_2 \pm s_2)$.

To compute these numbers of pairs, we will use the following formulas:

1. When $s_1 = 0$ or $s_2 = 0$.
   then $N_s(\bar{s}) = N_p(N_p - s_1) + (N_p - s_1)N_p$.

2. When $s_1 > 0$ and $s_2 > 0$.
   then $N_s(\bar{s}) = 4(N_p - s_1)(N_p - s_2)$.

These formulas can be easily verified by counting the number of pairs: Examples are given in fig (4.1):

These formulas are easy to program.

The second part of the algorithm consists of estimating $\alpha$. We have approximate values of $B(\bar{s})$ for different $\bar{s}$, and we want to find $\alpha$ for which (for some
Figure 4.1: Examples illustrating the number of pairs for two vectors. The upper window of $3 \times 3$ pixels would have 6 pairs with $s_1 = 2$ and $s_2 = 0$. The lower window of $3 \times 3$ pixels would have 8 pairs with $s_1 = 2$ and $s_2 = 1$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
constant $C$):

$$B(\bar{s}) \approx C|\bar{s}|^\alpha$$  \hspace{1cm} (4.4)

We thus have several (approximate) equations for two unknown: $C$ and $\alpha$. In mathematical terms, we have an over determined system of equations. In mathematical statistics, there is a method for solving such system of equations that is called a least squares method. Normally, when we have a system of equations

$$f_1(x_1, x_2, ..., x_k) \approx y_1$$

$$...$$

$$f_m(x_1, x_2, ..., x_k) \approx y_m$$

with $k < m$ unknowns $x_1, ..., x_k$, we find $x_i$ from the condition that the mean square value of $f_i$ is the smallest, i.e.

$$\sum_{j=1}^{m} (f_j(x_1, ..., x_k) - y_j)^2 \rightarrow \min_{x_1, ..., x_k}$$

In principle such methods are applicable in the general case, for arbitrary non-linear functions $f_j$. However, they become computationally much simpler if we use them for linear functions $f_j$. Therefore, it is reasonable to try to reduce a system (4.4) of non-linear equations to a linear system. Hopefully, we can do that by applying logarithm to both parts of (4.4). As a result, we get the following system:

$$\ln B(\bar{s}) \approx \ln C + \alpha \ln |\bar{s}|$$  \hspace{1cm} (4.5)

Here, we know $|\bar{s}|$ and $B(\bar{s})$, so we can easily compute $\ln |\bar{s}|$ and $\ln B(\bar{s})$. The only unknown quantities are $\ln C$ and $\alpha$. In standard denotations for least squares method, we can rewrite this equation as:

$$a_0 + a_1x_k \approx y_k$$  \hspace{1cm} (4.6)
where $a_0 = \ln C$, $a_1 = \alpha$, index $k$ corresponds to different vectors $\tilde{x}$, $x_k = \ln |\tilde{s}|$, and $y_k = \ln B(\tilde{s})$. To find $a_1$, we can thus use standard formulas of the least squares method. We seek to minimize

$$\sum_{k=1}^{m}[a_0 + a_1 x_k - y_k]^2 \quad (4.7)$$

From multivariable calculus, the values of $(a_0, a_1)$ that minimize (4.7) will satisfy

$$\left(\frac{\partial}{\partial a_0} \sum_{k=1}^{m}[a_0 + a_1 x_k - y_k]^2\right)(\partial a_0) = 0, \quad \left(\frac{\partial}{\partial a_1} \sum_{k=1}^{m}[a_0 + a_1 x_k - y_k]^2\right)(\partial a_1) = 0$$

taking the partial derivatives we get:

$$ma_0 + \left(\sum_{k=1}^{m} x_k\right) a_1 = \sum_{k=1}^{m} y_k$$

$$\left(\sum_{k=1}^{m} x_k\right) a_0 + \left(\sum_{k=1}^{m} x_k^2\right) a_1 = \sum_{k=1}^{m} x_k y_k$$

These equations can be solved for a unique solution $(a_0, a_1)$.

$$a_1 = \left(\sum_{k=1}^{m} x_k \sum_{k=1}^{m} y_k - m \sum_{k=1}^{m} x_k y_k\right) / \left(\sum_{k=1}^{m} x_k^2 - m \sum_{k=1}^{m} x_k^2\right)$$

This value $a_1$ gives the desired value $\alpha$.

### 4.2 The main idea

Our main idea is that since fractal dimension (or, the parameter $\alpha$) is a reasonable characteristic of a solder joint, before calculating all previously known features, we estimate the value of $\alpha$. If $\alpha$ is out of order, then we do not have to compute anything else: the solder joint is defective. If $\alpha$ is okay, then we have to compute other features.
So for the joints for which $\alpha$ criterion works we save lots of time, because for them we need to compute only 1 characteristic and not 33 as before. For others we waste a little of time, because now we need to estimate 34 characteristics instead of 33 characteristics.

To check whether overall we save time or not we must figure out how often $\alpha$ criterion helps.

To do that, we made an experiment that is explained in Chapter 5.
Chapter 5

EXPERIMENTAL CONFIRMATION OF THE IDEA

5.1 Description of the hardware

To check whether the algorithm for the fractal dimension is able to distinguish a defective solder joint from a good solder joint, we used a machine vision system that was set up at the machine vision applications laboratory at UTEP. This machine vision system consists of a vision box, a camera, a computer and some light source. The vision box is a piece of hardware which comes along with Vision C software to control it. This vision box provides an interface of the camera to the computer. This box captures the image through the camera and converts it into the data which is the pixel values. This data can be manipulated using different functions which are provided in the VisionC libraries. We were using a Pulnix CCD camera with a 50mm lens. For considerable magnification a extension tube of 20mm is used.
Pictures are taken through the camera and is passed on to the vision box which is an INTELLEDEX 386P vision system. This vision box in turn passes on the necessary information to the computer. Before we start this experiment, a very important aspect of the whole system is the usage of proper lighting. At present the system used in Barlett [30] uses a diffused light source that gives a uniform illumination, but this kind of illumination leads to the fact that the observed brightness in any point is actually caused not only this very point, but by the neighboring points as well. In other words it means that when we analyze some area of the board, we add a noise to the resulting brightness. An illumination system should be able to provide best images that can yield the most contrast between the features of interest and the background. A specific lighting technique is dependent upon the characteristics of the objects being reviewed. In machine vision, four kinds of factors usually affect the performance of the illumination[5].

- The light reflection from shiny surfaces.
- The light absorption of dark objects.
- The contrast between inspected objects and the background.
- The interface of ambient lighting from the surroundings.

5.2 Solder joint illumination

We must analyze some of the features in order to set up the lighting system:

- Whether the reflectance of the solder surface will saturate the image
- If the lighting will provide three-dimensional characteristics of the solder joint onto a two-dimensional image plane.
• Local shaded areas of the solder should be avoided.

• The lighting system should be isolated from the surroundings, and not interfered with by the background lighting.

• The illumination system must be inexpensive, easy to set up, simple to maintain, and otherwise physically robust.

In order to build a good lighting system, and to meet the requirements a 360° fiber optic fluorescent ring lamp is used that was proposed by Chadha[13]. This ring lamp is relatively close to the joint, thus a surrounding invariant illumination environment is established.

5.3 Description of the experiment

The samples of the solder joints that we are working with are provided by Alcatel Co. The solder joints used by Barlett [30] were lab generated one’s which might not have all the characteristics of an actual solder joints. To avoid that possibility we are using the actual solder joints rather than using the solder joints generated in the laboratory. The inspection system is performed in the following steps.

• We take a optical image and use machine vision system to translate this image into pixel-by-pixel electronic one.

• It is not possible to process all the pixels, so we need to compress the resulting image, i.e., compute some characteristics that will be used in future computations. These characteristics are called features. So, we extract features from the image.

• After that, a classification algorithm is used to tell good joints from defective ones, and to tell what exactly is wrong in case of a defective joint.
Before the start of the experiment the solder joints were classified manually using the handbook of the solder joints used by Alcatel Co. From all the classes of defective solder joints created by this classification, one class turned out to be especially difficult to separate from good ones: namely, solder joint with a hole in the middle. So, we decided to test our new feature on exactly these hard-to-test solder joints. The experiment was conducted on 15 solder joints of which 10 were good solder joints and 5 with hole in the middle. 33 known and \( \alpha \) features were calculated under ideal conditions and under industrial conditions (in which we changed the lighting and changed the focus a little bit).

### 5.4 Experimental results

The values of 33 known features and of \( \alpha \) for good solder joints, and for solder joints with a hole in them are given in the Tables 5.1 — 5.14.

These features were calculated for solder joints under both ideal conditions and under industrial conditions.

The value of \( \alpha \) for good solder joints under ideal and industrial conditions is given in table 5.13. The range of possible values of \( \alpha \) is \([2.65, 2.81]\).

The value of \( \alpha \) for solder joints with a hole in them, under ideal and industrial conditions is given in table 5.14. The range of possible values of \( \alpha \) is \([2.38, 2.50]\).

Since the range of values of \( \alpha \) for a good solder joints and defective solder joints with a hole in it do not intersect we can thus use this feature to successfully classify a defective solder joint with a hole in it from a good solder joint. If \( \alpha \) does not belong to the interval \([2.65, 2.81]\) (i.e., if either \( \alpha < 2.65 \) or \( \alpha > 2.81 \)) then the solder joint is defective. If \( \alpha \in [2.65, 2.81] \), then it does not have a hole in it, although it must be tested further to double check that it has no other defects. For every other features out experiment (see Tables 5.1-5.12) shows that the intervals...
<table>
<thead>
<tr>
<th>Feature</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 1</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0.0594</td>
<td>0.0628</td>
<td>0.0615</td>
<td>0.0588</td>
<td>0.0447</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.0138</td>
<td>0.0140</td>
<td>0.0209</td>
<td>0.0126</td>
<td>0.0085</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.0308</td>
<td>0.0318</td>
<td>0.0627</td>
<td>0.0397</td>
<td>0.0069</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.1222</td>
<td>0.1242</td>
<td>0.0974</td>
<td>0.1060</td>
<td>0.1113</td>
</tr>
<tr>
<td>$m_0$</td>
<td>13.41</td>
<td>13.93</td>
<td>19.05</td>
<td>12.46</td>
<td>8.22</td>
</tr>
<tr>
<td>$m_1$</td>
<td>13.40</td>
<td>13.93</td>
<td>19.05</td>
<td>12.46</td>
<td>8.22</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.0097</td>
<td>0.0103</td>
<td>0.0112</td>
<td>0.0102</td>
<td>0.0067</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0.000145</td>
<td>-0.000166</td>
<td>-0.000052</td>
<td>0.000100</td>
<td>-0.000077</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.000145</td>
<td>0.000166</td>
<td>0.000052</td>
<td>0.000100</td>
<td>0.000077</td>
</tr>
<tr>
<td>$m_5$</td>
<td>8.94</td>
<td>9.29</td>
<td>12.70</td>
<td>8.31</td>
<td>5.48</td>
</tr>
<tr>
<td>$m_6$</td>
<td>1369.8</td>
<td>1351.4</td>
<td>1697.9</td>
<td>1214.5</td>
<td>1223.6</td>
</tr>
<tr>
<td>$m_7$</td>
<td>180.1</td>
<td>194.4</td>
<td>363.6</td>
<td>155.7</td>
<td>67.69</td>
</tr>
<tr>
<td>$m_8$</td>
<td>13.41</td>
<td>13.93</td>
<td>19.05</td>
<td>12.46</td>
<td>8.22</td>
</tr>
<tr>
<td>$m_9$</td>
<td>10.94</td>
<td>11.37</td>
<td>15.56</td>
<td>10.18</td>
<td>6.71</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>18370.4</td>
<td>18834.1</td>
<td>32359.9</td>
<td>15143.5</td>
<td>10059.5</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>0.0147</td>
<td>0.0160</td>
<td>0.0046</td>
<td>0.0097</td>
<td>0.0114</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>0.180319</td>
<td>0.150478</td>
<td>0.133737</td>
<td>0.181722</td>
<td>0.173311</td>
</tr>
</tbody>
</table>

Table 5.1: First 17 features from the first four feature sets calculated on five good solder joints under ideal lighting conditions.
Table 5.2: Next 16 features from the first four feature sets calculated on five good solder joints under ideal lighting conditions.
Table 5.3: First 17 features from the first four feature sets calculated on five good solder joints under ideal conditions.
Table 5.4: Next 16 features from the first four feature sets calculated on five good solder joints under ideal conditions.
Table 5.5: First 17 features from the first four feature sets calculated on five good solder joints under industrial conditions.
<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_{13} )</td>
<td>6.54</td>
<td>10.91</td>
<td>-17.08</td>
<td>12.15</td>
<td>3.92</td>
</tr>
<tr>
<td>( m_{14} )</td>
<td>0.647</td>
<td>0.778</td>
<td>0.0966</td>
<td>0.0919</td>
<td>0.171</td>
</tr>
<tr>
<td>( m_{15} )</td>
<td>0.00009</td>
<td>0.00007</td>
<td>0.0009</td>
<td>0.0001</td>
<td>0.0009</td>
</tr>
<tr>
<td>( m_{16} )</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>( m_{17} )</td>
<td>0.0015</td>
<td>0.0016</td>
<td>0.0015</td>
<td>0.0017</td>
<td>0.0013</td>
</tr>
<tr>
<td>( m_{18} )</td>
<td>0.267</td>
<td>0.298</td>
<td>0.415</td>
<td>0.176</td>
<td>0.162</td>
</tr>
<tr>
<td>( m_{19} )</td>
<td>325.2</td>
<td>296.5</td>
<td>317.9</td>
<td>326.4</td>
<td>311.1</td>
</tr>
<tr>
<td>( m_{20} )</td>
<td>0.0021</td>
<td>0.002</td>
<td>0.0022</td>
<td>0.0023</td>
<td>0.0016</td>
</tr>
<tr>
<td>( m_{21} )</td>
<td>0.0017</td>
<td>0.0016</td>
<td>0.002</td>
<td>0.0019</td>
<td>0.0016</td>
</tr>
<tr>
<td>( m_{22} )</td>
<td>0.0021</td>
<td>0.002</td>
<td>0.0022</td>
<td>0.0023</td>
<td>0.0016</td>
</tr>
<tr>
<td>( m_{23} )</td>
<td>0.0017</td>
<td>0.0016</td>
<td>0.002</td>
<td>0.0019</td>
<td>0.0016</td>
</tr>
<tr>
<td>( A )</td>
<td>1.03</td>
<td>1.02</td>
<td>1.02</td>
<td>1.02</td>
<td>1.01</td>
</tr>
<tr>
<td>( c_0 )</td>
<td>2.017</td>
<td>2.156</td>
<td>2.799</td>
<td>2.198</td>
<td>2.144</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>0.786</td>
<td>0.801</td>
<td>0.897</td>
<td>0.962</td>
<td>0.727</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>7.19</td>
<td>7.05</td>
<td>5.82</td>
<td>7.13</td>
<td>4.85</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>668.48</td>
<td>631.24</td>
<td>598.69</td>
<td>624.13</td>
<td>544.18</td>
</tr>
</tbody>
</table>

Table 5.6: Next 16 features from the first four feature sets calculated on five good solder joints under industrial conditions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0.0606</td>
<td>0.0761</td>
<td>0.0711</td>
<td>0.0819</td>
<td>0.0821</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.0198</td>
<td>0.02142</td>
<td>0.0236</td>
<td>0.0219</td>
<td>0.0222</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.0526</td>
<td>0.0499</td>
<td>0.0392</td>
<td>0.0367</td>
<td>0.0511</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.1962</td>
<td>0.1459</td>
<td>0.1403</td>
<td>0.1168</td>
<td>0.1269</td>
</tr>
<tr>
<td>$m_0$</td>
<td>13.79</td>
<td>18.42</td>
<td>20.61</td>
<td>14.12</td>
<td>16.54</td>
</tr>
<tr>
<td>$m_1$</td>
<td>13.79</td>
<td>18.42</td>
<td>20.61</td>
<td>14.12</td>
<td>16.54</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.0126</td>
<td>0.0174</td>
<td>0.0138</td>
<td>0.0133</td>
<td>0.0135</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0.00079</td>
<td>0.00091</td>
<td>0.00052</td>
<td>0.00064</td>
<td>0.0007</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.00079</td>
<td>0.00091</td>
<td>0.00052</td>
<td>0.00064</td>
<td>0.0007</td>
</tr>
<tr>
<td>$m_5$</td>
<td>8.91</td>
<td>10.51</td>
<td>11.05</td>
<td>14.74</td>
<td>9.14</td>
</tr>
<tr>
<td>$m_6$</td>
<td>1069.5</td>
<td>1219.3</td>
<td>1132.9</td>
<td>1001.7</td>
<td>1216.2</td>
</tr>
<tr>
<td>$m_7$</td>
<td>188.8</td>
<td>246.0</td>
<td>221.9</td>
<td>279.3</td>
<td>214.4</td>
</tr>
<tr>
<td>$m_8$</td>
<td>3.79</td>
<td>18.42</td>
<td>20.61</td>
<td>14.12</td>
<td>16.51</td>
</tr>
<tr>
<td>$m_9$</td>
<td>11.69</td>
<td>15.08</td>
<td>15.91</td>
<td>12.37</td>
<td>12.52</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>12969.3</td>
<td>16452.0</td>
<td>17061.8</td>
<td>17652.3</td>
<td>15725.8</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>.0626</td>
<td>.0574</td>
<td>.0596</td>
<td>.0396</td>
<td>.0411</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>.1254</td>
<td>.1338</td>
<td>.1491</td>
<td>.1256</td>
<td>.1317</td>
</tr>
</tbody>
</table>

Table 5.7: First 17 features from the first four feature sets calculated on five good solder joints under industrial conditions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{13}$</td>
<td>4.93</td>
<td>4.21</td>
<td>6.36</td>
<td>5.65</td>
<td>2.77</td>
</tr>
<tr>
<td>$m_{14}$</td>
<td>.981</td>
<td>.429</td>
<td>.525</td>
<td>.621</td>
<td>.552</td>
</tr>
<tr>
<td>$m_{15}$</td>
<td>.00097</td>
<td>.00094</td>
<td>.00091</td>
<td>.00086</td>
<td>.00089</td>
</tr>
<tr>
<td>$m_{16}$</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$m_{17}$</td>
<td>.0018</td>
<td>.0018</td>
<td>.0018</td>
<td>.0018</td>
<td>.0018</td>
</tr>
<tr>
<td>$m_{18}$</td>
<td>.222</td>
<td>.543</td>
<td>.369</td>
<td>.486</td>
<td>.592</td>
</tr>
<tr>
<td>$m_{19}$</td>
<td>201.6</td>
<td>255.1</td>
<td>217.6</td>
<td>204.8</td>
<td>228.7</td>
</tr>
<tr>
<td>$m_{20}$</td>
<td>.0036</td>
<td>.0036</td>
<td>.0035</td>
<td>.0036</td>
<td>.0039</td>
</tr>
<tr>
<td>$m_{21}$</td>
<td>.0026</td>
<td>.0029</td>
<td>.0026</td>
<td>.0026</td>
<td>.0027</td>
</tr>
<tr>
<td>$m_{22}$</td>
<td>.0036</td>
<td>.0036</td>
<td>.0035</td>
<td>.0036</td>
<td>.0039</td>
</tr>
<tr>
<td>$m_{23}$</td>
<td>.0026</td>
<td>.0029</td>
<td>.0026</td>
<td>.0026</td>
<td>.0027</td>
</tr>
<tr>
<td>$A$</td>
<td>1.02</td>
<td>1.02</td>
<td>1.03</td>
<td>1.03</td>
<td>1.02</td>
</tr>
<tr>
<td>$c_0$</td>
<td>1.961</td>
<td>1.758</td>
<td>1.501</td>
<td>2.131</td>
<td>1.246</td>
</tr>
<tr>
<td>$c_1$</td>
<td>0.814</td>
<td>0.961</td>
<td>1.01</td>
<td>0.788</td>
<td>0.791</td>
</tr>
<tr>
<td>$c_2$</td>
<td>6.22</td>
<td>7.73</td>
<td>9.29</td>
<td>8.12</td>
<td>7.21</td>
</tr>
<tr>
<td>$c_3$</td>
<td>601.83</td>
<td>756.96</td>
<td>718.68</td>
<td>664.39</td>
<td>659.16</td>
</tr>
</tbody>
</table>

Table 5.8: Next 16 features from the first four feature sets calculated on five good solder joints under industrial conditions.
Table 5.9: First 17 features from the first four feature sets calculated on five bad solder joints under ideal conditions.

<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0.090</td>
<td>0.115</td>
<td>0.124</td>
<td>0.087</td>
<td>0.095</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.025</td>
<td>0.034</td>
<td>0.034</td>
<td>0.026</td>
<td>0.028</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.069</td>
<td>0.034</td>
<td>0.059</td>
<td>0.017</td>
<td>0.027</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.079</td>
<td>0.267</td>
<td>0.171</td>
<td>0.084</td>
<td>0.187</td>
</tr>
<tr>
<td>$m_0$</td>
<td>17.83</td>
<td>36.98</td>
<td>32.06</td>
<td>22.22</td>
<td>29.01</td>
</tr>
<tr>
<td>$m_1$</td>
<td>17.83</td>
<td>36.98</td>
<td>32.06</td>
<td>22.22</td>
<td>29.01</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.017</td>
<td>0.015</td>
<td>0.019</td>
<td>0.015</td>
<td>0.014</td>
</tr>
<tr>
<td>$m_3$</td>
<td>-0.0012</td>
<td>0.0004</td>
<td>0.0003</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.0012</td>
<td>0.0004</td>
<td>0.0003</td>
<td>0.001</td>
<td>0.0003</td>
</tr>
<tr>
<td>$m_5$</td>
<td>11.89</td>
<td>24.65</td>
<td>21.38</td>
<td>14.82</td>
<td>19.34</td>
</tr>
<tr>
<td>$m_6$</td>
<td>1022.5</td>
<td>2352.5</td>
<td>1648.0</td>
<td>1461.7</td>
<td>1961.6</td>
</tr>
<tr>
<td>$m_7$</td>
<td>318.76</td>
<td>1368.78</td>
<td>1029.52</td>
<td>494.51</td>
<td>842.59</td>
</tr>
<tr>
<td>$m_8$</td>
<td>17.83</td>
<td>36.98</td>
<td>32.06</td>
<td>22.22</td>
<td>29.01</td>
</tr>
<tr>
<td>$m_9$</td>
<td>14.56</td>
<td>30.19</td>
<td>26.18</td>
<td>18.11</td>
<td>23.68</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>18238.1</td>
<td>87001.1</td>
<td>52846.8</td>
<td>324583.8</td>
<td>56912.6</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>0.074</td>
<td>0.029</td>
<td>0.018</td>
<td>0.069</td>
<td>0.022</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>0.095</td>
<td>0.128</td>
<td>0.054</td>
<td>0.098</td>
<td>0.066</td>
</tr>
</tbody>
</table>
Table 5.10: Next 16 features from the first four feature sets calculated on five bad solder joints under ideal conditions.
<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0.1007</td>
<td>0.0586</td>
<td>0.0799</td>
<td>0.0902</td>
<td>0.0644</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.0295</td>
<td>0.0154</td>
<td>0.0267</td>
<td>0.0285</td>
<td>0.0202</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.0721</td>
<td>0.0516</td>
<td>0.0576</td>
<td>0.0391</td>
<td>0.0224</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.0950</td>
<td>0.0819</td>
<td>0.0507</td>
<td>0.0522</td>
<td>0.0577</td>
</tr>
<tr>
<td>$m_0$</td>
<td>21.26</td>
<td>10.15</td>
<td>17.74</td>
<td>19.63</td>
<td>13.07</td>
</tr>
<tr>
<td>$m_1$</td>
<td>21.26</td>
<td>10.15</td>
<td>17.74</td>
<td>19.63</td>
<td>13.07</td>
</tr>
<tr>
<td>$m_2$</td>
<td>0.0184</td>
<td>0.0105</td>
<td>0.0165</td>
<td>0.0178</td>
<td>0.0125</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0.00093</td>
<td>0.00074</td>
<td>-0.00035</td>
<td>0.00081</td>
<td>0.00124</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.00093</td>
<td>0.00074</td>
<td>0.00035</td>
<td>0.00081</td>
<td>0.00124</td>
</tr>
<tr>
<td>$m_5$</td>
<td>14.18</td>
<td>6.77</td>
<td>11.83</td>
<td>13.09</td>
<td>8.72</td>
</tr>
<tr>
<td>$m_6$</td>
<td>1151.8</td>
<td>965.08</td>
<td>1069.7</td>
<td>1100.7</td>
<td>1044.9</td>
</tr>
<tr>
<td>$m_7$</td>
<td>453.10</td>
<td>103.39</td>
<td>315.51</td>
<td>356.14</td>
<td>171.26</td>
</tr>
<tr>
<td>$m_8$</td>
<td>21.26</td>
<td>10.15</td>
<td>17.74</td>
<td>19.63</td>
<td>13.07</td>
</tr>
<tr>
<td>$m_9$</td>
<td>17.36</td>
<td>8.29</td>
<td>14.48</td>
<td>16.03</td>
<td>10.67</td>
</tr>
<tr>
<td>$m_{10}$</td>
<td>24496.5</td>
<td>-9802.8</td>
<td>18984.3</td>
<td>21610.2</td>
<td>13662.4</td>
</tr>
<tr>
<td>$m_{11}$</td>
<td>0.0508</td>
<td>0.0705</td>
<td>0.0213</td>
<td>0.0158</td>
<td>0.0998</td>
</tr>
<tr>
<td>$m_{12}$</td>
<td>0.0490</td>
<td>0.0747</td>
<td>0.0682</td>
<td>-0.0271</td>
<td>0.0254</td>
</tr>
</tbody>
</table>

Table 5.11: First 17 features from the first four feature sets calculated on five bad solder joints under industrial conditions.
<table>
<thead>
<tr>
<th>Features</th>
<th>Solder joint 1</th>
<th>Solder joint 2</th>
<th>Solder joint 3</th>
<th>Solder joint 4</th>
<th>Solder joint 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{13}$</td>
<td>0.9648</td>
<td>1.06</td>
<td>-3.2</td>
<td>-0.5920</td>
<td>0.255</td>
</tr>
<tr>
<td>$m_{14}$</td>
<td>1.036</td>
<td>0.9433</td>
<td>0.3124</td>
<td>1.688</td>
<td>3.921</td>
</tr>
<tr>
<td>$m_{15}$</td>
<td>0.00090</td>
<td>0.00073</td>
<td>0.00113</td>
<td>0.00048</td>
<td>0.00031</td>
</tr>
<tr>
<td>$m_{16}$</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$m_{17}$</td>
<td>0.00166</td>
<td>0.00189</td>
<td>0.00177</td>
<td>0.00173</td>
<td>0.00178</td>
</tr>
<tr>
<td>$m_{18}$</td>
<td>0.4090</td>
<td>0.081</td>
<td>0.3561</td>
<td>0.1863</td>
<td>0.0543</td>
</tr>
<tr>
<td>$m_{19}$</td>
<td>113.02</td>
<td>144.2</td>
<td>146.12</td>
<td>59.79</td>
<td>53.20</td>
</tr>
<tr>
<td>$m_{20}$</td>
<td>0.00484</td>
<td>0.00283</td>
<td>0.00443</td>
<td>0.00434</td>
<td>0.00320</td>
</tr>
<tr>
<td>$m_{21}$</td>
<td>0.00438</td>
<td>0.00243</td>
<td>0.00386</td>
<td>0.00457</td>
<td>0.00304</td>
</tr>
<tr>
<td>$m_{22}$</td>
<td>0.00558</td>
<td>0.0034</td>
<td>0.0046</td>
<td>0.00528</td>
<td>0.00437</td>
</tr>
<tr>
<td>$m_{23}$</td>
<td>0.0036</td>
<td>0.00186</td>
<td>0.00369</td>
<td>0.00363</td>
<td>0.00187</td>
</tr>
<tr>
<td>$a$</td>
<td>1.03</td>
<td>1.02</td>
<td>1.03</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>$c_0$</td>
<td>0.4356</td>
<td>0.5334</td>
<td>2.5427</td>
<td>0.2108</td>
<td>0.0227</td>
</tr>
<tr>
<td>$c_1$</td>
<td>0.7117</td>
<td>0.7863</td>
<td>0.6308</td>
<td>0.6091</td>
<td>0.4871</td>
</tr>
<tr>
<td>$c_2$</td>
<td>3.38</td>
<td>2.01</td>
<td>3.96</td>
<td>2.20</td>
<td>0.077</td>
</tr>
<tr>
<td>$c_3$</td>
<td>859.5</td>
<td>582.9</td>
<td>839.3</td>
<td>832.4</td>
<td>694.6</td>
</tr>
</tbody>
</table>

Table 5.12: Next 16 features from the first four feature sets calculated on five bad solder joints under industrial conditions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 5.13: The value of $\alpha$ that was calculated for good solder joints.

<table>
<thead>
<tr>
<th>Solder Joint</th>
<th>The value of $\alpha$ (ideal conditions)</th>
<th>The value of $\alpha$ (industrial conditions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint 1</td>
<td>2.65</td>
<td>2.68</td>
</tr>
<tr>
<td>Joint 2</td>
<td>2.67</td>
<td>2.74</td>
</tr>
<tr>
<td>Joint 3</td>
<td>2.71</td>
<td>2.69</td>
</tr>
<tr>
<td>Joint 4</td>
<td>2.73</td>
<td>2.76</td>
</tr>
<tr>
<td>Joint 5</td>
<td>2.81</td>
<td>2.74</td>
</tr>
<tr>
<td>Joint 6</td>
<td>2.75</td>
<td>2.76</td>
</tr>
<tr>
<td>Joint 7</td>
<td>2.68</td>
<td>2.74</td>
</tr>
<tr>
<td>Joint 8</td>
<td>2.74</td>
<td>2.66</td>
</tr>
<tr>
<td>Joint 9</td>
<td>2.70</td>
<td>2.78</td>
</tr>
<tr>
<td>Joint 10</td>
<td>2.72</td>
<td>2.80</td>
</tr>
</tbody>
</table>

Table 5.14: The value of $\alpha$ that was calculated for solder joints with a hole in them.

<table>
<thead>
<tr>
<th>Solder Joint</th>
<th>The value of $\alpha$ (Ideal conditions)</th>
<th>The value of $\alpha$ (Industrial conditions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint 1</td>
<td>2.43</td>
<td>2.47</td>
</tr>
<tr>
<td>Joint 2</td>
<td>2.45</td>
<td>2.42</td>
</tr>
<tr>
<td>Joint 3</td>
<td>2.45</td>
<td>2.38</td>
</tr>
<tr>
<td>Joint 4</td>
<td>2.50</td>
<td>2.42</td>
</tr>
<tr>
<td>Joint 5</td>
<td>2.51</td>
<td>2.42</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\( I_g, I_{ch} \) of possible values of this feature for good joints and for joints with a hole in them intersect. This means that no other feature taken by itself can successfully separate good solder joints from solder joints with a hole in them.

### 5.5 Conclusion

There are already 33 features being used in the previous automated vision system. We propose to add this feature to the existing set of 33 features. What we suggest is that before calculating the rest of the 33 features, the value of \( \alpha \) should be calculated first.

Let us consider that it takes 33 units of time to calculate the 33 features for one solder joint and 1 unit to calculate the value of \( \alpha \). Thus for a good solder joint the total time spent would be 34 units of time, but we need not calculate the rest of the 33 features in case the solder joint had a hole in it, since it would already be classified as defective based upon the value of \( \alpha \). On an average on a printed circuit board there were 5\% solder joints with a hole in it. So for these 5\% of joints we need to calculate only the value of \( \alpha \) and thus spend only 1 unit time, but in case of the rest 95\% of the joints we need to spend 34 units instead of the original 33 units. Thus the total time spent on a whole printed circuit board is 
\[
0.05 \times 1 + 0.95 \times 34 = 31.55 \text{ units}.
\]
So on average we save \( 33 - 31.55 = 1.45 \) units of time on every printed circuit board.

Hence we can say that by adding this feature to the list of features, not only can we distinguish the solder joints with a hole in them from a good one, but also reduce the computation time which is one of the major concerns while developing the automated inspection systems.
Bibliography


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
FILE *ptr2;

main()
{
    int a,b,k,i,j,row,col,kk,num_pair, jj, m, n, INT[32][32], count;
    double s[2000], int_value[2000];
    pts = fopen("fourierb.1", "r");
    count = 0;
    k = 0; row=32; col=32;
    for(a=0; a<32;++a) {
        for(b=0; b<32;++b) {
            fscanf(pts, "%d", &INT[a][b]);
        }
    }
    fclose(pts);
    kk=0;

    for(jj=0; jj<2000;++jj)
        int_value[jj] = 0;

    for(n=0; n<col;++n) {
        for(m=0; m<=n;++m) {
            if(n==0&&m==0) break;

            /* Array s[] is used to store the distance between two points */
            s[kk] = log(pow(((double)n*(double)n) +
                            ((double)m*(double)m)), .5));
    }
/ * First condition when one of the points is on the first row or column of the array */
if ( n*m == 0 ) {
/* num_pairs is used to store the number of pairs for a particular distance */
    num_pair = row*(col-n) + col*(row-n);
    for(i=0;i<row-n;++i){
        for(j=0;j<col;++j) {
            ++count;
/* int_value[] is an array used to store the value of the summation for a different distances */
        int_value[kk] = int_value[kk] + pow((INT[i][j] - INT[i+n][j]), (double)2.);
    }
    for(i=0;i<row;++i) {
        for(j=0;j<col-n;++j) {
            ++count;
        int_value[kk] = int_value[kk] + pow(( INT[i][j]- INT[i][j+n]),(double)2.);
    }
    int_value[kk] = log(int_value[kk]/(float)num_pair);
    kk= kk+1;
    count = 0;
Second case when the two points lie anywhere except for the first row and column or on the diagonal between the point (0,0) and (31,31) */
else if ( m != n ) {

    num_pair = 2*(col-n)*(row-m) + 2*(col-m)*(row-n);
    for(i=0;i<row-m;++i){
        for(j=n;j<col;++j) {
            ++count;
            int_value[kk] = int_value[kk] + pow((INT[i][j] - INT[i+m][j-n]), 2.);
        }
    }
    for(i=0;i<row-m;++i) {
        for(j=0;j<col-n;++j) {
            ++count;
            int_value[kk] = int_value[kk] +
                          pow((INT[i][j]-INT[i+m][j+n]),2.);
        }
    }
    for(i=0;i<row-n;++i) {
        for(j=m;j<col;++j) {
            ++count;
            int_value[kk] = int_value[kk] +
                          pow((INT[i][j]-INT[i+n][j-m]),2.);
        }
    }
}
for (i=0; i<row-m; ++i) {
    for (j=n; j<col; ++j) {
        ++count;
        int_value[kk] = int_value[kk] +
            pow((INT[i][j] - INT[i+n][j+m]), 2.);
    }
}

int_value[kk] = log(int_value[kk]/(float)num_pair);
count = 0;
kk = kk+1;

/* Third case when the second point lies on the diagonal between
   point (0,0) and (31,31) */
else if (m == n) {

    num_pair = 2*(col-n) * (row-m);

    for (i=0; i<row-n; ++i){
        for (j=n; j<col; ++j) {
            ++count;

            int_value[kk] = int_value[kk] +
                pow((INT[i][j] - INT[i+n][j-n]), 2.);
        }
    }
}
for(i=0;i<row-m;++i) {
    for(j=0;j<col-n;++j) {
        ++count;

        int_value[kk] = int_value[kk] +
        pow(( INT[i][j] -INT[i+n][j+n]),2.);
    }
}
if (int_value[kk] != 0 )
    int_value[kk] = log(int_value[kk]/(float)num_pair);
k = kk+1;

    count = 0;
}
k = k+1;

/* Function to calculate least squares is called */
least_square(s,int_value,kk);

>Description

This function here is called by the main program and is used to calculate the least squares. The input to this function are the
arrays for the distances and the summation for these distances and a variable that contains the count for the possible values of these arrays.

```
void least_square(array, function, ii)
{ double array[], function[];
  int ii;
{
 int m, number, mx2, n, i, j, k, l, npo, lhold, ipo, jj, nmkpl, nmk,
    jm1, im1, degfr;
 int newval;
 float osigm, sw, epsil, hold, sum, sum2, sigma;
 double x[5000], y[5000], a[11][12], p[20], thld;

  number = ii;
  for (newval=0; newval<ii; ++newval) {
    x[newval+1] = array[newval];
    y[newval+1] = function[newval];
  }
  m = 1;
  osigm = 0;
  mx2 = m * 2;
  for( i = 1; i<=mx2; i++) {
    p[i]=0;
    for(j=1; j<=number; j++)
      p[i] = p[i] + pow(x[j], (float)i);
  }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
n = m+1;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
        k = i + j - 2;
        if (k <= 0)
            a[i][i] = number;
        else
            a[i][j] = p[k];
    }
}
npo = n+1;
a[1][npo] = 0;
for (j = 1; j <= number; j++)
    a[1][npo] = a[1][npo] + y[j];
for (i = 2; i <= n; i++) {
    a[i][npo] = 0;
    for (j = 1; j <= number; j++) {
        l = i - 1;
        a[i][npo] = a[i][npo] + y[j] * pow(x[j], (float)l);
    }
}
for (i = 1; i <= m; i++) {
    lhold = i;
    hold = a[i][i];
    for (k = 1; k <= n; k++) {
        if (((fabs(a[k][i]) - fabs(hold)) > 0) {
hold = a[k][i];
    lhold = k;
}

if(lhold != 0) {
    for(j=1; j<=npo; j++) {
        thld = a[i][j];
        a[i][j] = a[lhold][j];
        a[lhold][j] = thld;
    }
}

if ( a[i][i] != 0 ) {
    ipo = i+1;
    for(j = ipo; j<=npo; j++)
        a[i][j] = a[i][j]/a[i][i];
    for(k=ipo; k<=n; k++) {
        for ( j = ipo; j<=npo; j++) {
            a[k][j] = a[k][j] - a[k][i] * a[i][j];
        }
    }
} else
    printf(" unique soln does not exist \
");
}

a[n][npo] = a[n][npo]/a[n][n];
for(k=1; k<=m; k++) {
    nmkp1 = n-k+1;
nmk = n-k;
sum = 0;
for(jj=nmkp1;jj<=n;jj++)
    sum = sum + a[nmk][jj]* a[jj][npo];
a[nmk][npo] = a[nmk][npo] - sum;
}
sum = 0;
for( i=1;i<=number;i++) {
    sum2 = a[i][npo];
    for(j=2;j<=n;j++) {
        jm1 = j-1;
        sum2 = sum2 + a[j][npo] * pow(x[i],(float)jm1);
    }
    sum = sum + pow((y[i] - sum2), 2.);
}
degfr = number-m-1;
if(degfr <= 0) {
    sigma = 0;
    osigm = 0;
}
else
    sigma = sum/degfr;
printf(" the least square poly of degre %d follows \n",m);
for( i=1; i<=n;i++) {
    im1 = i-1;
    printf( " %f \n", a[i][npo]);
printf(" \n", sigma);
}
Appendix B

Calculation of the first four feature sets

This program is used to calculate the features from the first four set of features. These programs were developed at MVAL at The University of Texas at El Paso.

/************************* ******************************
Include various header and source file that are used later in the program
************************* *******************************/

#include "stdio.h"
#include "vwindow.h"
#include "vhardwar.h"
#include "conio.h"
#include "vip1.h"
#include "vip2.h"
#include "vgraphic.h"
#include "vmemory.h"
# include "vblob.h"
# include "ctype.h"
# include "vcolors.h"
# include "math.h"
# include "vconfig.h"
# include "stdlib.h"

VBLOB bloba;
FILE *fn1;
char filename[14];
int p[75][75];
float z[75][75];
float p_norm[75][75];
int nx, ny;
int m=0;
float x[75], y[75];
float V_csw, V_ofr, V_tot;
float sigma2;
float x_cm, y_cm, z_cm;
float Ixx, Iyy, Izz, Ixy;
float Ix, Iy;
int top=125, lft=75, rgt=150, bot=200;
int p_max;
int count, rowa, rowb, cola, colb, row_centre, col_centre, colplot, centre;
int win_x, win_y;
int r;
int thresh=30;
main()
{
    int i,j;
    int xx,yy;
    int r1,r2,r3,r4,rr;
    int e_lft,e_rgt,e_top,e_bot;
    printf(" Name the output filename -->");
    scanf("%s", &filename);
    fn1 = fopen(filename,"w");
    printf("Press any key to continue\n");
    getch();
    vactual();
    printf("\n Adjust the board (press any key to continue...)\n");
    getch();
    vsnap(0);
    vsnapwait();
    printf("The snap has been shot, press any key to retrieve it\n");
    getch();
    vdig(0);
    printf("Press any key to bring the box for selecting the image\n");
    getch();
    printf("Adjust the window (Press Q to continue...) \n");
    vbox(&top,&bot,&lft,&rgt);
    vplotellipse(top,bot,lft,rgt,RED,0);
    vwindow("WIND1",top,lft,bot-top+1,rgt-lft+1,"*ISEG");
nx=bot-top;
ny=rgt-lft;
printf("top=%.d, bot=%.d, lft=%.d, rgt=%.d \n", top, bot, lft, rgt);

for (i=0;i<=nx-1;i++)
{
    for (j=0;j<=ny-1;j++)
    {
        xx=i+top;
        yy=j+lft;
        p[i][j] = vgetpixel(xx,yy);
        if (p[i][j]>63) {printf("ERROR ");
        printf("%.d \n", p[i][j]);}
    }
}

win_x=15;win_y=15;
vwindow("WIND2", top+win_x, lft+win_y, bot-top-2*win_x,
rgt-lft-2*win_y,"*ISEG");
vithresh("WIND2", thresh);
vconan("*ISEG", 0, 0, thresh);
centre=((lft+rgt)/2);
vgetmaxblob(&bloba);
rowa=(int)bloba.mini;
rowb=(int)bloba.maxi;
cola=(int)bloba.minj;
colb=(int)bloba.maxj;
row_centre=(int) ((rowa+rowb)/2);
col_centre=(int) ((cola+colb)/2);

fprintf(fn1,"area= %d
",bloba.area);
printf("rowa= %d\n",rowa);
printf("rowb= %d \n",rowb);
printf("cola= %d \n",cola);
printf("colb= %d \n",colb);
printf("rowcentre= %d \n",row_centre);
printf("colcentre= %d \n",col_centre);

printf("centre of the blob is %d,%d\n",row_centre,col_centre);
if(bloba.area>45) printf("Excessive solder OR lead not found\n");
if(bloba.area<15) printf("Lead missing\n");
vplotcross(row_centre-5,row_centre+5,col_centre-5,
col_centre+5,RED,SET);
r1=row_centre-top;
r2=col_centre-lft;
r3=bot-row_centre;
r4=rgt-col_centre;
r=r1;
if (r>r2) r=r2;
if (r>r3) r=r3;
if (r>r4) r=r4;

printf("r1 = %d, r2 = %d, r3 = %d, r4 = %d, r = %d\n",
r1,r2,r3,r4,r);
rr=(int)r/5;
printf("rr = %d\n",rr);
e_top=row_centre-rr;
e_bot = row_centre + rr;
e_lft = col_centre - rr;
e_rgt = col_centre + rr;
getch();
vplorellipse(e_top, e_bot, e_lft, e_rgt, RED, 0);
e_top = row_centre - (int)r/4;
e_bot = row_centre + (int)r/4;
e_lft = col_centre - (int)r/4;
e_rgt = col_centre + (int)r/4;
getch();
vplorellipse(e_top, e_bot, e_lft, e_rgt, RED, 0);
e_top = row_centre - (int)3*r/5;
e_bot = row_centre + (int)3*r/5;
e_lft = col_centre - (int)3*r/5;
e_rgt = col_centre + (int)3*r/5;
getch();
vplorellipse(e_top, e_bot, e_lft, e_rgt, RED, 0);
/* e_top = row_centre - r;
e_bot = row_centre + r;
e_lft = col_centre - r;
e_rgt = col_centre + r;
getch();
vplotellipsee(e_top, e_bot, e_lft, e_rgt, RED, 0);*/

******************************************************************************
Call the function to calculate various features
******************************************************************************
circle();
pixel_max();
size_norm();
norm_gray();
cent_subwin();
outer_frame();
norm_vol();
norm_var();
cent_mass();
moment();
face_area();
gauss();
fclose(fn1);
}

/*****************************************************************************
   This function draws a circle based on the Breshemham's circle
drawing algorithm.
*****************************************************************************/

circle()
{
    int i=0,j=0,n=0;
    int x[300],y[300],pp[76];
    int row[300],col1[300],col2[300];
    int var;
/*getc();
for (i=0;i<=nx-1;i++)
{
    for (j=0;j<=ny-1;j++)
    {
        vplotpixel(top+i,lft+j,p[i][j],0);
    }
}*/
pp[0]=3-2*r;
x[0]=0;
y[0]=r;
for (; ;)
{
    if (x[i]<=y[i]){
        if(pp[i]<0){
            pp[i+1]=pp[i]+4*x[i]+6;
            x[i+1]=x[i]+1;
            y[i+1]=y[i];
        }
        else {
            pp[i+1]=pp[i]+4*(x[i]-y[i])+10;
            x[i+1]=x[i]+1;
            y[i+1]=y[i]-1;
        }
        i++;}
    else{i--;
        printf("i=%d \n ",i);
        break;};
for (j = 0; j <= i; j++) {
    x[i+j] = y[i-j];
    y[i+j] = -1 + x[i-j];
}

n = 2 * i;
for (j = 0; j <= n; j++) {
    x[n+j] = x[n-j];
    y[n+j] = -1 * y[n-j];
}

printf("Press any key to continue\n");
getch();
m = 2 * n;
for (j = 0; j <= m; j++) {
    x[m+j] = -1 * x[m-j];
    y[m+j] = y[m-j];
    row[j] = row_centre + y[m-j];
    col1[j] = col_centre + x[m-j];
    col2[j] = col_centre + x[m+j];
    vplotpixel(row[j], col1[j], RED, 0);
    vplotpixel(row[j], col2[j], RED, 0);
}

for (i = 0; i <= nx-1; i++) {
    if (top+i<row[0] || top+i>row[m])
        {

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
for(j=0; j<=ny-1; j++) p[i][j] = 0;
}
for(i=0; i<=m-1; i++)
{
var=row[i]-top;
for(j=0; j<col2[i]-lft; j++) {
p[var][j]=0;
}
for(j=col1[i]-lft+1; j<=ny-1; j++) {
p[var][j]=0;
}
}
getch();
for (i=0; i<=nx-1; i++)
{
    for (j=0; j<=ny-1; j++)
    {
        vplotpixel(top+i,lft+j,p[i][j],0);
    }
}

for(j=0; j<=m; j++){
vplotpixel(row[j],col1[j],RED,0);
vplotpixel(row[j],col2[j],RED,0);}
}
function to draw ellipse given the normalised co-ordinates

draw_ellipse(xx,yy)
float xx,yy;
{
  int i;
  int ttop,bbot,llft,rrgt;
  i=xx*m;
  ttop=row_centre-(int)(i);
  bbot=row_centre+(int)(i);
  i=yy*m;
  llft=col_centre-(int)(i);
  rrgt=col_centre+(int)(i);
  vplotellipse(ttop,bbot,llft,rrgt,RED,0);
}

function to calculate the maximum pixel value

pixel_max()
{
  int i,j;
}
p_max = p[0][0];
for (i=0;i<=nx-1;i++)
{
    for (j=0;j<=ny-1;j++)
    {
        if (p_max < p[i][j]) p_max = p[i][j];
    }
}
printf(fn1,"p_max = %d\n", p_max);

/********************************************
function for size normalization
********************************************/

size_norm()
{
    int i,j;
    float test_x,test_y;
    for(i=0;i<=m-1;i++)
    {
        test_x = 1+2*i-m;
        x[i]=test_x/(2*m);
    }
    for(j=0;j<=m-1;j++)
    {

test_y = 1 + 2 * j - m;
y[j] = test_y / (2 * m);
}

/******************************************************************************
function to calculate normalized maps of gray scales
*******************************************************************************/

norm_gray()
{
    int i, j;
    float test;
    test = (float)1 / p_max;
    printf("test = %f\n", test);
    for (i = 0; i <= nx - 1; i++)
    {
        for (j = 0; j <= ny - 1; j++)
        {
            z[i][j] = p[i][j] * test;
        }
    }
}

/******************************************************************************
function for calculating normalized volume
*******************************************************************************/
norm_vol()
{
    int i,j;
    float F=0;
    for(i=0;i<=m-1;i++)
    {
        for (j=0;j<=m-1;j++)
        {
            F=F+p[row Centre-top-r+i][col Centre-lft-r+j];
        }
    }
    printf(fn1,"F=%.f\n",F);
    V_tot=F/(m*m*p_max);
    printf(fn1,"V_tot = %.f\n",V_tot);
}

*******************************************************************************/

function to calculate variance of normalized gray level surface about its mean value
*******************************************************************************/

norm_var()
{
    int i,j;
    float sum=0;
for(i=0;i<=m-1;i++)
{
    for (j=0;j<=m-1;j++)
    {
        sum=sum+(z[row_centre-top-r+i][col_centre-lft-r+j]-V_tot)*
            (z[row_centre-top-r+i][col_centre-lft-r+j]-V_tot);
    
    }
}

sigma2=sum/(m*m-1);

return sigma2;
}

function for calculating the central subwindow volume

/*************************************************************/
\begin{verbatim}
x_mod = x[i];
y_mod = y[j];
if (x_mod < 0) x_mod = -1 * x_mod;
if (y_mod < 0) y_mod = -1 * y_mod;
if (x_mod < 0.1 && y_mod < 0.1)
    F = F + p[row_centre-top-r+i][col_centre-lft-r+j];

f printf(fn1,"F_cent = \%.1f\n",F);
area = 0.2 * 0.2 * m * m;
V_csw = F / (area * p_max);
f printf(fn1,"V_csw = \%.1f\n",V_csw);
}

/*******************************************************************/

function for calculating the outer frame region volume

/*******************************************************************/

outer_frame()
{
    int i,j;
    float F = 0;
    float area;
    float x_mod, y_mod;
    draw_ellipse(0.2, 0.2);
    draw_ellipse(0.3, 0.3);
\end{verbatim}
for(i=0;i<=m-1;i++)
{
    for(j=0;j<=m-1;j++)
    {
        x_mod=x[i];
        y_mod=y[j];
        if(x_mod<0) x_mod=-1*x_mod;
        if(y_mod<0) y_mod=-1*y_mod;
        if(x_mod>0.2&&y_mod>0.2&&x_mod<0.3&&y_mod<0.3)
            F=F+p[row_centre-top-r+i][col_centre-1ft-r+j];
    }
}

fprintf(fn1,"F_out= %f\n", F);

area = (0.3*0.3-0.2*0.2)*m*m;

V_ofr=F/(area*p_max);

/*V_ofr=F/(nx*ny*p_max);*/

fprintf(fn1,"V_ofr = %f\n",V_ofr);
}

 músamen\\
{\\
    int i,j;
    float X=0,Y=0;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
float test1, test2;
test1 = (float)1/m;
test2 = (float)1/m;
for (i = 0; i <= m-1; i++)
{
    for (j = 0; j <= m-1; j++)
    {
        X = X + 2 * (test1) * (test2) * z[row_centre-top-r+i][col_centre-lft-r+j] * x[i];
        Y = Y + 2 * (test1) * (test2) * z[row_centre-top-r+i][col_centre-lft-r+j] * y[j];
        /* printf("z[i][j] = \%f \n", z[i][j]); */
    }
}
x_cm = X / (2 * V_tot);
y_cm = Y / (2 * V_tot);
z_cm = 0;
printf(fn1, "x_cm = \%f \n", x_cm);
printf(fn1, "y_cm = \%f\n", y_cm);
}

/*******************************************/

function to calculate various moments about the center of mass
*******************************************/

moment()
{
    int i,j;
    float buf1=0,buf2=0,buf3=0,buf4=0,buf5=0,buf6=0;
    float test1,test2;
    float m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15;
    float m16,m17,m18,m19,m20,m21,m22,m23;
    test1=(float)1/m;
    test2=(float)1/m;
    for(i=0;i<=m-1;i++)
    {
        for(j=0;j<=m-1;j++)
        {
            buf1=buf1+(1/12.0)*((test2)*(test2)+(2*z[row_centre-top-r+i][col_centre-lft-r+j])*(2*z[row_centre-top-r+i][col_centre-lft-r+j])+2*(test1)*(test2)*z[row_centre-top-r+i][col_centre-lft-r+j]*(y[j]-y_cm)*(y[j]-y_cm);
            buf2=buf2+(1/12.0)*((test1)*(test1)+(2*z[row_centre-top-r+i][col_centre-lft-r+j])*(2*z[row_centre-top-r+i][col_centre-lft-r+j])+2*(test1)*(test2)*z[row_centre-top-r+i][col_centre-lft-r+j]*(x[i]-x_cm)*(x[i]-x_cm);
            buf3=buf3+2*(test1)*(test2)*z[row_centre-top-r+i][col_centre-lft-r+j]*(x[i]-x_cm)+(y[j]-y_cm)*(y[j]-y_cm);
            buf4=buf4+2*(test1)*(test2)*z[row_centre-top-r+i][col_centre-lft-r+j]*(x[i]-x_cm)*(y[j]-y_cm);
        }
    }
}
buf5 = buf5 + test1*test2*z[row_centre-top-r+i][col_centre-lft-r+j]*(y[j]-y_cm)*(y[j]-y_cm);
buf6 = buf6 + test1*test2*z[row_centre-top-r+i][col_centre-lft-r+j]*(x[i]-x_cm)*(x[i]-x_cm);

Ixx = buf1;
Iyy = buf2;
Izz = buf3;
Ixy = buf4;
Ix = buf5;
Iy = buf6;
m0 = Ixx;
m1 = Iyy;
m2 = Izz;
m3 = Ixy;
m4 = m3;
if (m4 < 0) m4 = -1*m4;
m5 = (m0 + m1 + m2)/3.0;
m6 = (m0 + m1)/(2.0*m2);
m7 = m0*m1 + m1*m2 + m2*m0;
m8 = sqrt((m0*m0 + m1*m1)/2.0);
m9 = sqrt((m0*m0 + m1*m1 + m2*m2)/3.0);
m10 = m0*m1/m2;
m11 = m3/m2;
if (m11 < 0) m11 = -1*m11;
m12 = (m0 - m1)/m2;
m13 = (m0 - m1) / m3;
m14 = m3 / (m0 - m1);
if (m14 < 0) m14 = -1 * m14;
m15 = m0 - m1;
if (m15 < 0) m15 = -1 * m15;
m16 = (m0 * m0 + m1 * m1) / (2 * m0 * m1);
m17 = 2 * m2 / (sqrt(m0 * m0 + m1 + m1));
m18 = (m0 - m1) * (m1 - m2) * (m2 - m0);
if (m18 < 0) m18 = -1 * m18;
m19 = (m0 * m0 - m1 * m1) / (m2 * m2);
if (m19 < 0) m19 = -1 * m19;
m20 = Ix;
m21 = Iy;
m22 = (Iy + Iy + sqrt((Ix - Iy) * (Ix - Iy) + 4 * Ixy * Ixy)) / 2.0;
m23 = (Ix + Iy - sqrt((Ix - Iy) * (Ix - Iy) + 4 * Ixy * Ixy)) / 2.0;
/* printf("Ixx = \t\%f\n", Ixx);
 printf("Iyy = \t\%f\n", Iyy);
 printf("Izz = \t\%f\n", Izz);
 printf("Ixx = \t\%f\n", Ixy);
 printf("Ix = \t\%f\n", Ix);
 printf("Iy = \t\%f\n", Iy); */
fprintf(fn1, "m0=%f \n", m0);
fprintf(fn1, "m1=%f \n", m1);
fprintf(fn1, "m2=%f \n", m2);
fprintf(fn1, "m3=%f \n", m3);
fprintf(fn1, "m4=%f \n", m4);
fprintf(fn1, "m5=%f \n", m5);
Function to calculate the Facted surface Area feature

```c
f_aced_area()
{
    int i,j;
}
```
float d1, d2, d3, d4, d5;
float D1, D2, A123 = 0, A234 = 0;
float buf1, buf2, buf3, buf4, buf5;
float A, AA;
float test1, test2;
test1 = (float) i / (m - 1);
test2 = (float) i / (m - 1);
for (i = 0; i <= m - 2; i++)
{
    for (j = 0; j <= m - 2; j++)
    {
        buf1 = z[row_centre-top-r+i+1][col_centre-lft-r+j] - z[row_centre-top-r+i][col_centre-lft-r+j];
        d1 = sqrt(1 + buf1 * buf1);
        buf2 = z[row_centre-top-r+i][col_centre-lft-r+j+1] - z[row_centre-top-r+i][col_centre-lft-r+j];
        d2 = sqrt(1 + buf2 * buf2);
        buf3 = z[row_centre-top-r+i+1][col_centre-lft-r+j] - z[row_centre-top-r+i+1][col_centre-lft-r+j+1];
        d3 = sqrt(1 + buf3 * buf3);
        buf4 = z[row_centre-top-r+i+1][col_centre-lft-r+j] - z[row_centre-top-r+i+1][col_centre-lft-r+j+1];
        d4 = sqrt(1 + buf4 * buf4);
        buf5 = z[row_centre-top-r+i][col_centre-lft-r+j+1] - z[row_centre-top-r+i][col_centre-lft-r+j+1];
        d5 = sqrt(1 + buf5 * buf5);
        D1 = (d1 + d2 + d3) / 2.0;
D2=(d3+d4+d5)/2.0;
A123=sqrt(D1*(D1-d1)*(D1-d2)*(D1-d3));
A234=sqrt(D2*(D2-d3)*(D2-d4)*(D2-d5));
AA=AA+A123+A234;

A=test1*test2*AA;
fprintf(fn1,"FACETED AREA = %f \n",A);
}

/*******************************                              
 * Gaussian features                                                   *
 * First the discrete data is approximated as quadric polynomials.     *
 * A window of 3x3 pixels is used                                       *
*************************************************************/

gauss()
{
    int i,j;
    int NN=3;
    float MM;
    float a00,a10,a01,a20,a11,a02;
    float A1,A2,A3,A4;
    float BX1,BX2,BX3,BY1,BY2,BY3;
    float BXX,BYY,BXY;
    float E,F,G,g;
    float L,M,N,b,H,K;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
float Fx,Fy;
float pplus=0,pminus=0,Kavg=0,Havg=0,bavg=0,gavg=0;
float test1,test2;
float sqrtg;
test1=(float)1/(m-1);
test2=(float)1/(m-1);

MM=(NN-1)/2.0;
A1=-1.732;
A2=sqrt((3/MM)*(MM+1)*(2*MM+1));
A3=(28*MM*MM*MM*MM*MM/45)+(14*MM*MM*MM*MM/9)+
   (10*MM*MM*MM/9)-(2*MM/30);
A4=MM*(MM+1)/3;

/*printf("A1 = %f, A2 = %f, A3 = %f, A4 = %f\n", A1,A2,A3,A4);*/
for(i=1;i<=m-1;i++)
{
  for(j=1;j<=m-1;j++)
\[
\begin{align*}
\text{a10} &= (z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j-1}]*(-1) + \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j}]*(-1) + \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j+1}]*(-1) + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j-1}]*0 + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j}]*0 + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j+1}]*0 + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j-1}]*1 + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j}]*1 + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j+1}]*1 ) * A1*A2; \\

\text{a01} &= (z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j-1}]*(-1) + \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j}]*0 + \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j+1}]*1 + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j-1}]*(-1) + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j}]*0 + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j+1}]*1 + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j-1}]*(-1) + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j}]*0 + \\
& z[\text{row\_centre-top-r+i+1}][\text{col\_centre-lft-r+j+1}]*1 ) * A1*A2; \\

\text{a20} &= ((z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j-1}]+ \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j}] + \\
& z[\text{row\_centre-top-r+i-1}][\text{col\_centre-lft-r+j+1}])*(1-A4)+ \\
& (z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j-1}]+ \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j}] + \\
& z[\text{row\_centre-top-r+i}][\text{col\_centre-lft-r+j+1}])*A4+
\end{align*}
\]
\[ (z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j-1]+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j]+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j+1]) \]

\[ *(1-A4))A1*A3; \]

\[ a11=(z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j-1]*(-1)*(-1)+ z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j]*(-1)*0+ z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j+1]*(-1)*1+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j-1]*0*(-1)+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j]*0*0+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j+1]*0*1+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j-1]*1*(-1)+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j]*1*0+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j+1]*1*1)A2*A2; \]

\[ a02=(z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j-1] *(1-A4)+ z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j]*A4+ z[row\_centre\_top\_r+i-1][col\_centre\_lft\_r+j+1] *(1-A4)+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j-1] *(1-A4)+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j] *(1-A4)+ z[row\_centre\_top\_r+i][col\_centre\_lft\_r+j+1] *(1-A4)+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j-1] *(1-A4)+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j] *(1-A4)+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j+1] *(1-A4)+ z[row\_centre\_top\_r+i+1][col\_centre\_lft\_r+j+1] *
(1-A4))A1*A3; \]
Now that the quadric functions have been approximated at each pixel, find the first derivative and the second derivatives

\[
\begin{align*}
F_x &= a_{10}A_1A_2 + (2a_{20}A_1A_3)x[i] + (a_{11}A_2A_2)y[j]; \\
F_y &= a_{01}A_1A_2 + (2a_{02}A_1A_3)y[j] + (a_{11}A_2A_2)x[i]; \\

E &= 1 + F_xF_x; \\
G &= 1 + F_yF_y; \\
F &= F_xF_y; \\
g &= 1 + F_xF_x + F_yF_y; \\
sqrt{g} &= \sqrt{g}; \\

L &= 2a_{20}A_1A_3/\sqrt{g}; \\
N &= 2a_{02}A_1A_3/\sqrt{g}; \\
M &= a_{11}A_2A_2/\sqrt{g}; \\
b &= L*N - M*M; \\
H &= (E*N + G*L - 2F*M)/(2g); \\
K &= b/g; \\

if (K > 0) \text{ pplus = pplus + 1; } \\
if (K < 0) \text{ pminus = pminus + 1; } \\
K_{avg} &= K_{avg} + test1*test2*K; \\
H_{avg} &= H_{avg} + test1*test2*H; \\
b_{avg} &= b_{avg} + test1*test2*b;
\end{align*}
\]
gavg = gavg + test1 + test2 * sqrt(g);

} }

fprintf(fn1,"pplus = %f\n",pplus);
fprintf(fn1,"pminus = %f\n",pminus);
fprintf(fn1,"Kavg = %f\n",Kavg);
fprintf(fn1,"Havg = %f\n",Havg);
fprintf(fn1,"bavg = %f\n",bavg);
fprintf(fn1,"gavg = %f\n",gavg);
}
CURRICULUM VITAE

Navdeep Singh Chadha was born on August 30, 1969 in Agra, U.P., India. The younger of the two sons of Major Harsaran Singh Chadha and Dr. Tarvinder Kaur Chadha, he graduated from D.A.V. College, Chandigarh, India, in the spring of 1987 and entered the Electronics Engineering department of Guru Nanak Dev University (G.N.D.U), Amritsar, India, in the fall of 1987. While pursuing a bachelor’s degree in electronics engineering, he worked with Electronics System Punjab India Ltd., Chandigarh, India, a computer manufacturer company, as an Engineer Trainee in the summer of 1990. He received his bachelor’s degree in Electronics Engineering degree from G.N.D.U in the spring of 1991. He was elected as a vice president to the electronics students association in fall of 1990. He represented his department in basketball, soccer, and athletics team. In the fall of 1991, he entered the Graduate School at The University of Texas at El Paso. During his course of graduate studies he held positions of teaching assistant in Fall of 92 and as a research assistant in Fall of 91 and Spring of 92 and 93 at the Machine Vision Applications Laboratory (M.V.A.L) at UTEP.

Permanent address: C-3, G.N.D. University
Amritsar, Punjab-143005 India.

This thesis was typed by Navdeep Singh Chadha.