Recognize of printed Arabic characters.

Fatma. El Khaly
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
RECOGNITION OF PRINTED ARABIC CHARACTERS

by

Fatma El Khaly

A Thesis submitted to the
Faculty of Graduate Studies and Research Through the
Department of Electrical Engineering in Partial
Fulfillment of the requirements for the
Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

1988
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L’auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L’auteur conserve la propriété du droit d’auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-50506-0
To my father
ABSTRACT

This thesis discusses the development of algorithms for the machine recognition of camera captured Arabic characters and their isolation from the printed text.

Arabic text is formed from a set of connected or individual fonts. Recognition algorithms based only on individual characters need to be supplemented with a separation algorithm.

Moment-invariant descriptors are investigated for the purpose of recognition of individual characters.

Meanwhile, to recognize connected printed characters, an algorithm for separation of individual characters is developed.
ACKNOWLEDGMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. M.A. Sid-Ahmed, for his advice, guidance and constant encouragement throughout the progress of this thesis.

I extend my sincerest thanks and gratitude to my husband, M.T. Boraie, and to my parents for their moral support, help and constant encouragement at all times.

Thanks are also due to Mrs. Zeleney for her diligence and excellence in typing this thesis.
TABLE OF CONTENTS

ABSTRACT .............................................. v
ACKNOWLEDGEMENTS ...................................... vi
LIST OF FIGURES ........................................ ix
LIST OF APPENDICES .................................... xi

CHAPTER

I. INTRODUCTION ........................................ 1

1.1 State of the Art (Literature Survey) ............... 2
1.1.1 Preprocessing .................................... 3
1.1.2 Recognition Techniques ............................ 4
1.1.3 Discriminator ..................................... 6
1.1.4 Separation of Connected Arabic Characters ....  6

1.2 Goals of this Thesis ................................. 7
1.3 Outline of the Thesis ................................. 7

II. UTILITY OF MOMENT IN Variant DESCRIPTORS IN THE RECOGNITION OF Arabic Characters .......... 8

2.1 Introduction .......................................... 8
2.2 Derivation of the Moment Invariant Descriptions .... 8
2.2.1 Central Moments ................................. 9
2.2.2 Normalized Central Moments ...................... 11
2.2.3 Orthogonal Moment Invariant ..................... 13
2.2.4 Moment Invariants Under General Linear Transformation ................ 19

2.3 Application of Using Moment Invariant Technique in Case of Arabic Characters .......... 23
2.3.1 Isolated Characters .............................. 23
2.3.2 Connected Characters .............................. 23

III. LETTERS SEPARATION ALGORITHM .................... 27

3.1 Introduction .......................................... 27
3.2 Characteristics of Arabic Characters ............... 27
3.2.1 Important Characteristics in the Recognition of Arabic Characters ........ 28
3.2.2 Almailim and Yamaguchi Separation Technique .......... 30

3.3 Definitions .......................................... 34
3.3.1 New Technique of Printed Arabic Character Separation ............. 34
IV. RESULTS AND CONCLUSION OF APPLYING THE NEW SEPARATION ALGORITHM AND THE MOMENT INVARIANTS IN RECOGNIZING PRINTED ARABIC CHARACTERS 42

4.1 Original Data 42
4.2 Scaled Characters 42
4.3 Thinned Characters 42
  4.3.1 Original Data Set 42
  4.3.2 Scaled with 0.5 Scaling Factor 43
  4.3.3 Scaled with 0.2 Scaling Factor 43
4.4 Separation of the Characters 43
4.5 Summary and Conclusion 54

REFERENCES 57

VITA AUCTORIS 75
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Flowchart of the procedure for calculating the moment invariant descriptors</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Sample of isolated Arabic characters</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>Sample of isolated Arabic characters</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Showing word written in connected Arabic characters</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Feature points</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Arabic alphabet</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Amounts calculated for the classification of Group 3 strokes</td>
<td>32</td>
</tr>
<tr>
<td>3.4a</td>
<td>Features, codes, and identification vectors of Group 3 strokes</td>
<td>33</td>
</tr>
<tr>
<td>3.4b</td>
<td>Features, codes, and identification vectors of Group 4 strokes</td>
<td>33</td>
</tr>
<tr>
<td>3.4c</td>
<td>Features, codes, and identification vectors of Group 5 strokes</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Mid line (J_{mid}), I_{max}, J_{min}, J_{max}, and J_{min} (borders) of the word</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Flowchart to explain the determination of the midline &quot;J_{mid}&quot;</td>
<td>37</td>
</tr>
<tr>
<td>3.7</td>
<td>Flowchart explaining the new separation Techniques</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>The word &quot;\text{\textbackslash a}&quot; before thinning algorithm is applied</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>The word &quot;\text{\textbackslash a}&quot; after thinning algorithm is applied</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>The word &quot;\text{\textbackslash a}&quot; after the first character has been separated</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>The word &quot;\text{\textbackslash a}&quot; after the second character has been separated</td>
<td>48</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Calculation for the moment invariant for the first and second characters and their location in the look-up table.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Sentence &quot;٥١&quot; after applying the thinning algorithm.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Sentence &quot;٥٢&quot; after the first character has been separated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Sentence after the second character has been separated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Sentence after the third character has been separated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Sentence after the fourth and fifth characters were separated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>I. Separation Algorithm</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>II. Recognition Algorithm</td>
<td>66</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

In the last two decades, there has been progress in the field of machine recognition techniques for "Latin, Chinese" characters [1], [2], [3].

Meanwhile, the machine recognition of Arabic characters has not yet been fully explored. Despite the fact that Arabic characters are used in writing many widespread languages like "Arabic, Persian, Urdu, etc." the published literature includes very little information about the computer recognition of Arabic characters, for example [4], [5].

In order to illustrate the difference between Arabic text and various other texts, the main characteristics of Arabic characters are provided next.

i) Arabic is always printed cursively from right to left.

ii) Arabic words consist of one or more connected portion, for example, the word فذ consists of two portions:

a) فذ which consists of two characters د، ف

b) ف which consists of one character.
iii) The discontinuities between portions are due to some characters that are not connectable from the left side with the succeeding portion, for example, the character.

iv) Every character has more than one shape depending on its position within a connected portion of the word, as example:

\[ \begin{array}{c}
\text{char1} \\
\text{char2} \\
\text{char3}
\end{array} \]

The emphasis in the recognition techniques has been concentrated on Latin and Chinese characters. However, these forms of printed text are usually presented as a set of isolated characters.

The shape of the character in these forms is independent of its location in the printed word contrary to Arabic text. In this thesis Moment Invariant Descriptors will be investigated to assess its suitability to discriminate between the different printed Arabic characters.

However, to complete the assessment the development of a separation procedure is developed in this thesis.

1.1 State of the Art (Literature Survey)

There are three principal components in most recognition techniques that have been used.

i) Preprocessing

Which differs from one technique to another, as an
example: if we use the Fourier shape descriptors techniques the images of the character need to be thresholded (transferred to a binary image) then the border of each character has to be detected before applying the Fourier descriptors.

ii) Recognition Techniques

All the recognition techniques are based on the features extracted from analyzing the character.

iii) Discriminators

Utilize these features in arriving at a recognition decision.

1.1.1 Preprocessing

i) Thresholding the Image - in which the image of the character is transformed into a binary image (containing two gray levels only). One gray level represents the character and the other the background [7], [8].

ii) Border Following of the Characters. Some techniques such as the Fourier shape descriptors require that the border of the character to be extracted first.

The border following scheme is applied to the binary representation of the original image, and where all the border points (pixels) are stored, and all the other pixels are deleted from the image [6].

iii) Thinning of the Character. Some techniques such as "the stroke vector sequences" [1], [3] require that the
binary representation of the original image be thinned, where the binary image of the characters will be transformed to a one pixel thick image.

Since most of the recognition techniques (such as the moment invariant approach) are insensitive to variation in size, shape and orientation of the character, but are sensitive to variation in the thickness of the character, therefore, thinning is required as part of the pre-processing. Various thinning algorithms are available in the literature [9], [10]. The algorithm developed by Tellache [8] was found suitable for Arabic characters.

1.1.2 Recognition Techniques

i) Template Matching Technique. In this scheme, the digitized image of the character is used as a template and correlated with the image to discriminate between the character chosen and the various other characters [4].

The disadvantages of using this technique in the case of Arabic characters are:

a) The large numbers of reference template needed to account for the variation in size, shape and orientation.

b) The computational burden is high because of the dimensionality of the data.

ii) Fourier Shape Descriptors. In this scheme, a border following algorithm is applied to the thresholded
image to detect the border of each character. Thus, the Fourier Transform of the contour of each character is computed. Only components with significant values are retained and all the other components are simply ignored [11], [12].

The disadvantages of using this technique, in the case of Arabic characters, are:

a) Since it is based on the discrimination of boundary curves any discontinuity in the contour due to a noise associated with the thresholding algorithm will cause a fault in the recognition.

b) The derivatives of the Fourier coefficients could involve extensive arithmetic computations.

c) Characters should be isolated before processing.

iii) **Strokes Vector Sequences Technique.** In this scheme, the characters are described in terms of a set of geometrical features such as strokes at different orientation [1]. As an example, the character ☞ may be represented by a vertical stroke (|) on the right, a horizontal stroke (—) at the middle and another vertical stroke (|) on the left.

The disadvantage of using this technique is that most of the Arabic characters consist of open loops, closed loops and branches beside the vertical and horizontal lines which make it difficult (or require long processing) to recombine
these strokes into a character for recognition.

iv) **Moment Invariant.** In this scheme, the moments of the character pixels about the center of gravity are used as features.

These features have been modified by Hu [13] to yield a set of descriptors that are invariant to shift, translation and rotation.

1.1.3 **Discriminator**

A discriminator utilizes the features extracted from analyzing the character to arrive at a recognition decision. Discriminators can be classified into two classes.

i) **Numerical Discriminators** [14]. Numerical discriminators utilize the parameters of the features extracted from analyzing the character and computes a distance. The recognition of the character is made on the basis of the value of the distance.

ii) **Look-up Tables and Trees** [15]. In this scheme the data (learning set) is divided into groups and arranged in a table, each group representing a branch in the tree where the grouping is based on some similarity between the characters forming each group.

1.1.4 **Separation of Connected Arabic Characters** [5]

In this scheme, a thinning algorithm is applied to the thresholded image, followed by a segmentation, where
the start and end point of each stroke is extracted, followed by a classification of strokes, where the strokes are classified to one of five groups, then a calculation of feature vectors. The feature vectors are group dependent (i.e., different calculations are done for different groups), followed by a combination of strokes into a character string, when the strokes are classified into main and secondary strokes, then each secondary stroke is assigned to a main stroke. Finally, from the combination of stroke codes, a recognition can be obtained [8].

1.2 Goals of This Thesis

(a) Moment Invariant Descriptors will be investigated to assess their suitability to discriminate between the different printed Arabic characters.

(b) Develop an algorithm for segmenting connected printed Arabic text.

1.3 Outline of the Thesis

The utility of the moment invariant technique in the recognition of Arabic characters is discussed in Chapter II. A new separation algorithm is introduced in Chapter III. Chapter IV discusses the results and provides a conclusion of the work presented in this thesis.
CHAPTER II

UTILITY OF MOMENT IN Variant DESCRIPTORS IN
THE RECOGNITION OF ARABIC CHARACTERS

2.1 Introduction

Machine recognition of characters independent of position, size, and orientation has been a goal of much recent research. To achieve maximum utility and flexibility, the methods used should be insensitive to variation in shape and should provide for improved performance with repeated trials. The set of Moment Invariant descriptors introduced by Ming Kui Hu [13] meets all these conditions to some degree.

In this method the concept of moments, central moments, size normalization and principal axes are used. The method further uses "Moment Invariant" (moment referred to a pair of uniquely determined principal axes) to characterize each character for recognition.

2.2 Derivation of the Moment Invariant Descriptors

The two dimensional \((p+q)\)th order moment of a density distribution function \(\rho(x,y)\) can be expressed as follows:
\[ m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \rho(x,y) x^p y^q \, dx \, dy \]

where

i) \( p, q = 0, 1, 2, \ldots \)

ii) \( \rho(x,y) \) is a piecewise continuous function.

In the case of digital functions \( m_{pq} \) will be expressed by

\[ m_{pq} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} x^p y^q \rho(x,y) \]

where \( \rho(x,y) \) is a digital function.

2.2.1 Central Moments

The central moment can be expressed as

\[ \mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q \rho(x,y) \, dx \, dy \]

where \( \rho(x,y) \) is a continuous function.

In the case of a digital function:

\[ \mu_{pq} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q \rho(x,y) \]

where \( \rho(x,y) \) is a digital function and

\[ \bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}} \]

Note that: central moments do not change under any
translation. It is quite easy to express central moments in terms of the ordinary moments as follows:

\[ \mu_{00} = m_{00} = \mu \]

\[ \mu_{01} = \sum_{x} \sum_{y} (x-x')^0 (y-y')^1 \rho(x,y) \]
\[ = m_{01} - \frac{m_{01}}{m_{00}} m_{00} \]

\[ \mu_{10} = \sum_{x} \sum_{y} (x-x')^1 (y-y')^0 \rho(x,y) \]
\[ = m_{10} - \frac{m_{10}}{m_{00}} m_{00} \]

\[ \mu_{20} = \sum_{x} \sum_{y} (x-x')^2 (y-y')^0 \rho(x,y) \]
\[ = m_{20} - \frac{m_{10}}{m_{00}} m_{10} \]
\[ = m_{20} - \bar{x} \cdot m_{10} \]

\[ \mu_{02} = m_{02} - \bar{y} m_{10} \]

\[ \mu_{11} = \bar{y} m_{01} \]

\[ \mu_{30} = m_{30} - 3 \bar{x} m_{20} + 2 m_{10} \bar{x}^2 \]

\[ \mu_{12} = m_{12} - 2 \bar{y} m_{11} - \bar{x} m_{02} + 2 \bar{y}^2 m_{10} \]

\[ \mu_{21} = m_{21} - 2 \bar{x} m_{11} - \bar{y} m_{20} - 2 \bar{x}^2 m_{0} \]
2.2.2 Normalized Central Moments

Normalized Central Moments are invariant to any change in the size and translation.

i) **Algebraic Form and Invariants.** The homogeneous polynomial of two variables $u$ and $v$.

$$
\Delta = a_{p0}u^p + (\frac{p(p+1)}{2})a_{p-1,1}u^{p-1}v + (\frac{p(p-1)}{2})a_{p-2,2}u^{p-2}v^2 + \ldots + (\frac{p(p-1)}{2})a_{1,p-2}uv^{p-2} + a_{0p}v^p
$$

is called a binary form of order $p$.

Using the notation introduced by Cayley [18], the above form may be written as

$$
f = (a_{p0}; a_{p-1,1}; \ldots; a_{1,p-2}; a_{0p})(u,v)^p \quad (1)
$$

a homogeneous polynomial $I(a)$ of the coefficients $a_{p0}, \ldots, a_{0p}$ is an algebraic invariants of weight $w$ if

$$
I(a'_{p0}, \ldots, a'_{0p}) = \Delta^w I(a_{p0}, \ldots, a_{0p})
$$

where, $a'_{p0}, \ldots, a'_{0p}$ are the new coefficients obtained by substituting the following transformation into the original form $(1)$

$$
\begin{bmatrix}
u \\
v'
\end{bmatrix} = \begin{bmatrix}
a & \gamma \\
\beta & \delta
\end{bmatrix} \begin{bmatrix}
u' \\
v'
\end{bmatrix}
$$
\[ \Delta = \begin{vmatrix} a & \gamma \\ \beta & \delta \end{vmatrix} \neq 0 \]

if \( w = 0 \) the invariant is called an absolute invariant. If \( w \neq 0 \) it is called a relative invariant. \( \Delta \) may not be limited to the determinant of the transformation. If the algebraic form of order \( p \) has an algebraic invariant

\[ I(a'_p, \ldots, a'_0) = \Delta^w I(a'_p, \ldots, a'_0) \]

then the moment invariant of order \( p \) has the same invariant.

ii) Similitude Moment Invariants. Under the similitude transformation (change of size)

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

where \( \alpha \) is a constant; therefore, each coefficient of the homogeneous polynomial \( I(a) \) will be presented as

\[ a'_{pq} = \alpha^{p+q} a_{pq}. \]

Note that: (\( \alpha \) is not the determinant).

For the moment invariants

\[ \mu'_p = \alpha^{p+q+2} \mu_{pq}. \]
Since $\mu_{10}' = \mu_{01}' = 0$ by eliminating $\alpha$ between the zeroth order relation and the remaining ones, we have the following absolute similitude moment invariants

$$\frac{\mu_{pq}'}{(\mu_{p+q/2}^{'2} + 1)} = \frac{\mu_{pq}'}{(\mu_{(p+q)/2}^{'2} + 1)}$$

Note that: for the zeroth order $\mu_{pq}' = \alpha^2 \mu_{pq}$, $p+q = 2, 3, \ldots$, and $\mu_{10}' = \mu_{01}' = 0$.

2.2.3 Orthogonal Moment Invariant

Under the following proper orthogonal transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

since we have

$$\begin{vmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{vmatrix} = 1$$

The moment invariants are exactly the same as the algebraic invariant. If we treat the moments as the coefficient of an algebraic form

$$(\mu_{p0}, \ldots, \mu_{0p})(u, v)^p$$

under the following transformation
\[
\begin{bmatrix}
u \\
v
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
u' \\
v'
\end{bmatrix}
\] (3)

We can derive the moment invariants by the following algebraic method if we substitute both \(u, v\) and \(u', v'\) to the following transformation

\[
\begin{bmatrix}
U \\
V
\end{bmatrix} = \frac{1}{2}
\begin{bmatrix}
1 & i \\
1 & -1
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
U' \\
V'
\end{bmatrix} = \frac{1}{2}
\begin{bmatrix}
1 & i \\
1 & -1
\end{bmatrix}
\begin{bmatrix}
u' \\
v'
\end{bmatrix}
\] (4)

then

\[
U = \frac{1}{2}u + \frac{1}{2}iv \quad U' = \frac{1}{2}iu' + \frac{1}{2}iv'
\]

\[
V = \frac{1}{2}u + \frac{1}{2}iv \quad V' = \frac{1}{2}u' - \frac{1}{2}iv'
\]

and from Eq. (3), we get

\[
u = u'\cos \theta - v'\sin \theta
\]

\[
v = u'\sin \theta + v'\cos \theta
\]

\[
U = \frac{1}{2}(u'\cos \theta - v'\sin \theta) + \frac{1}{2}i(u'\sin \theta + v'\cos \theta)
\]

\[
= \frac{1}{2}(u'\cos \theta - \frac{1}{2}v'\sin \theta) + \frac{1}{2}(u'\sin \theta + \frac{1}{2}v'\cos \theta)
\]
\[ U = \frac{1}{2} u' \cos \theta + i v' \sin \theta \]
\[ V = \frac{1}{2} u' e^{i \theta} + \frac{1}{2} i v' \]

then
\[ U' = U e^{-i \theta} \]
\[ V' = V e^{i \theta}. \]

By substituting (4) and (5) in (2), we get

\[ (I'_{p0}, \ldots, I'_{0p})(u, v)^P = (\mu'_{p0}, \ldots, \mu'_{0p})(u, v)^P \]
\[ \equiv (\mu'_{p0}, \ldots, \mu'_{0p})(u', v')^P \]
\[ \equiv (I'_{p0}, \ldots, I'_{0p})(ue^{-i \theta}, ve^{i \theta})^P \]

where \( I'_{p0}, \ldots, I'_{0p} \) and \( I_{p0}, \ldots, I_{0p} \) are the corresponding coefficients after the substitution from the identity in \( U \) and \( V \) which lead to the following

\[ I'_{p0} = e^{i \theta} I_{p0}, \quad I'_{p-1,1} = e^{i(p-2)\theta} I_{p-2,1} \]
\[ I'_{0p} = e^{-i \theta} I_{0p}. \]

There are \((p+1)\) linearly independent moment invariants under proper orthogonal transformation \( \Delta = e^{i \theta} \).
From the identity of the first two expressions in (6) we get the following

\[
(I_{p0}, \ldots, I_{0p})(u,v)^P =
\]
\[
I_{p0}u^P + (P)I_{p-1,1}u^{P-1}v + (P)I_{p-2,2}u^{P-2}v^2
\]
\[
+ \cdots + (P)I_{1,p-1}uv^{P-1} + I_{0p}v^P
\]

then

\[
(\mu_{p0}, \ldots, \mu_{0p})(u,v)^P =
\]
\[
\mu_{p0}u^P + (P)\mu_{p-1,1}u + (P)\mu_{p-2,2}u^{P-2}v^2
\]
\[
+ \cdots + (P)\mu_{1,p-1}uv^{P-1} + \mu_{0p}v^P
\]

\[
U = \frac{1}{2}u + \frac{1}{2}iv \quad \text{and} \quad V = \frac{1}{2}u - \frac{1}{2}iv
\]

By substituting (9) into (7) we get the following

\[
I_{p0}(\frac{1}{2}u + \frac{1}{2}iv)^P + (P)I_{p-1,1}(\frac{1}{2}u + \frac{1}{2}iv)^P(\frac{1}{2}u - \frac{1}{2}iv)
\]
\[
\cdots (\frac{1}{2}u + \frac{1}{2}iv)^P = (P)[u^P + (P)u^{P-1}(iv) - (P)u^{P-2}v^2
\]
\[
+ \cdots + (P)u(iv)^{P-1}\n\]
\[
\cdots I_{p0} = \mu_{p0} + i(P)\mu_{p-1,1} - (P)\mu_{p-2,2}
\]
\[
+ \cdots + (P)\mu_{1,p-1}(i)^{P-1} + \mu_{0p}(i)^P
\]
then
\[
I_{p-1,1} = (\mu_{p0} + \mu_{p-2,2}) - i(p-2)(\mu_{p-1,1} + \mu_{p-3,3})
+ \ldots + (-i)^{p-2}(\mu_{2,p-2} + \mu_{0p}).
\]

\[
I_{p-r,r} = \left\{ \begin{array}{c}
(\mu_{p0}; \mu_{p-2,2}; \ldots; \mu_{p-2r,2r})(1,1) \\
(\mu_{p-1}; \mu_{p-3,3}; \ldots; \mu_{p-2r-1})(1,1)^r \\
(\mu_{2r,p-2r}; \mu_{2r+2,p-2r-2}; \ldots; \mu_{0p})(1,1)^r \\
(1,-i)^{p-2r}
\end{array} \right.
\]

where \( p-2r > 0, \)

Then,
\[
I_{p/2,p/2} = \mu_{p0} + (p/2)\mu_{1,-2,2} + (p/2)\mu_{p-4,4}
+ \ldots + \mu_{0p}.
\]

These \((p+1)\) I's are linearly independent linear functions of \(\mu\)'s and vice versa.

Similarly, since we have
\[
U' = Ve^{i\theta} \text{ and } V' = Ve^{-i\theta}
\]

and
\[
I'_{p0} = e^{-ip\theta}I_{p0}, \quad I'_{p-1,1} = e^{-i(p-2)\theta}I_{1,p-1}
\]
\[ I'_{1,p-1} = e^{-i(p-2)\theta}I_{p-1,1} \quad I'_{0p} = e^{i\theta}I_{p0} \]

by eliminating the factor \( e^{i\theta} \) in the second order equations

\[ I'_{20} = e^{i2\theta}I_{20} \quad \cdots \quad (1) \]

\[ I'_{20} = e^{-i2\theta}I_{02} \quad \cdots \quad (2) \]

by eliminating \( e^{i\theta} \),

\[ I'_{20} = I_{20}I_{02} \]

\[ I'_{11} = I_{11} \cdot \]

So far the second order moment has two independent invariants which are

\[ I'_{11}, I_{20}I_{02} \]

for the third order moment

\[ I'_{30} = e^{i3\theta}I_{30} \]

\[ I'_{30} = e^{-i3\theta}I_{03} \cdot \]

\( I_{30}I_{03} \) is an independent invariant and

\[ I'_{21} = e^{i\theta}I_{21} \]

\[ I'_{12} = e^{-i\theta}I_{1,2} \]
\[ I_{21}^{2} = I_{21}^{1} \] is an independent invariant for orders greater than or equal to 4 the independent invariants are

\[ I_{p}^{0,0}, I_{p-1,1}^{1,1}, p-1 \cdots I_{p-r,r}^{r,r}, p-r \]

If \( p \) is even we have also

\[ I_{p/2, p/2} \]

and also we can combine with the \((p-2)\)th order. All the independent moment invariants together form a complete system.

2.2.4 Moment Invariants Under General Linear Transformation

From the theory of algebraic invariants under the linear transformation

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} =
\begin{bmatrix}
    \alpha & \beta \\
    \gamma & \delta
\end{bmatrix}
\begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

and

\[
\Delta = \begin{vmatrix}
    \alpha & \gamma \\
    \beta & \delta
\end{vmatrix} \neq 0.
\]

For simplicity let \( A, B, C \) and \( a, b, c, d \) denote the second and third moments, respectively, then we may write the following two binary forms in terms of these moments as
\[(A,B,C)(u,u)^2\]
\[(a,b,c,d)(u,v)^3\]
\[Au^2 + Buv + Cv^2\]

We will have the following four algebraically independent invariants

\[I_1 = AC - B^2\]
\[I_2 = (ad-bc)^2 - 4(ac-b^2)(bd-c^2)\]
\[I_3 = A(bd-c^2) - B(ad-bc) + C(ac-b^2)\]
\[I_4 = a^2c^3 - 6abBC^2 + 6acC(2B^2-AC) + ad(6ABC-8B^2)\]
\[+ 9b^2AC^2 - 18bcABC + 6bdA(2B^2-AC) + 9c^2A^2C\]
\[+ 6cdBA^2 + d^2A^2\]

of weight \(w=2,6,4\) and 6, respectively. Since the moment invariants are exactly the same as the algebraic invariants, then if we treat the moments as the coefficient of an algebraic form the following set of moment invariants up to the third order, will be obtained.

\[\phi_1 = \mu_{20} + \mu_{02}\]
\[\phi_2 = (\mu_{30} - \mu_{02})^2 + 4\mu_{11}^2\]
\[\phi_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2\]
\[ \phi_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \]
\[ \phi_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] \]
\[ + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03}) \cdot [3(\mu_{30} + \mu_{12})^2 \]
\[ - (\mu_{21} + \mu_{03})^2] \]
\[ \phi_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] \]
\[ + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03}) \]

and one skew orthogonal invariant
\[ (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] \]
\[ - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] \]

This skew invariant is useful in distinguishing mirror images.

These methods can be generalized to accomplish pattern identification independent of position, size, and orientation.

Note that:

1. \( \mu \)'s in the above set of equations are the normalized central moments.

2. A flow chart showing the Moment Invariant procedure is shown in Fig. (2-1).
Fig. 2.1. Flowchart of the procedure for calculating the moment invariant descriptors.
2.3 Application of Using Moment Invariant Technique in Case of Arabic Characters

The moment invariant technique was applied to isolated (Figs. 2.2, 2.3) and connected characters (Fig. 2.4).

2.3.1 Isolated Characters

A 100% recognition rate was obtained in the case of isolated Arabic characters despite the fact that the majority of the characters have great similarity in their shape. The results will be discussed in detail in Chapter 4.

2.3.2 Connected Characters

In the case of connected characters an algorithm to isolate the character is developed and explained in detail in Chapter 3; also the recognition rate is introduced in Chapter 4.
Fig. 2.2. A sample of isolated Arabic characters.
Fig. 2.3. A sample of isolated Arabic characters.
Fig. 2.4. An example of a word written in connected Arabic characters.
CHAPTER III

LETTERS SEPARATION ALGORITHM

3.1 Introduction

The need for a separation algorithm arises from the fact that Arabic characters are usually connected.

A separation algorithm will be presented in this chapter which can be associated with most of the recognition techniques.

Almallim and Yamaguchi [5] developed a separation algorithm that can only be associated with the strokes vector sequence technique and cannot be applied to other forms of shape discrimination.

The method in [5] was described in section 1.1.4 in Chapter 1 of this thesis. Further details will be provided in section 3.2.2 in this chapter and compared to the proposed method.

3.2 Characteristics of Arabic Characters

i) Arabic words are always printed cursively from right to left.

ii) Arabic words consist of one or more connected portion.

iii) The discontinuities between portions are due
to some characters that are not connectable from the left side with the succeeding portion.

iv) Every character has more than one shape depending on its position within a connected portion of the word.

3.2.1 Important Characteristics in the Recognition of Arabic Characters

i) The cross, branch points inside characters, and the connection points between characters always fall near the writing line (to be called the midline below). This line provides useful context information (Fig. 3.1).

\[ \text{Fig. 3.1. Feature points} \]

\begin{center}
\begin{tabular}{ccc}
\hspace{2cm} & \hspace{2cm} & \\
\text{cross point} & \text{branch point} & \text{line end}
\end{tabular}
\end{center}

ii) Domains covered horizontally by characters overlap in many cases in handwritten texts (e.g., see characters "ta" and "kaf" in Fig. 3.2.

iii) Many characters differ only by the presence and the number of dots above or below the main part of the character shape.

iv) Most of the characters are formed by curves
<table>
<thead>
<tr>
<th>Character Name</th>
<th>Last</th>
<th>Mid</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ain</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Ba</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Ta</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Tha</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Meem</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Naa</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Kha</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Daa</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Thal</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Ra</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Za</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Seen</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Sheem</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Sad</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Dhad</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Taa</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Zaa</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Ain</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Gain</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Fa</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Daf</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Kaf</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Lam</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Noon</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Na</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Waw</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 3.2. Arabic alphabet "first", "middle" and "last" are the shapes of characters when they appear at the head, middle or tail of a connected portion of a word. "Single" is the shape of isolated characters.
and loops.

v) Many similar parts exist in different characters. For example, the shape (๑) appears in the characters "sad", "dhad", "tta", "zha", "gaf", and "fa", and also can appear alone as the character "meem".

3.2.2 The Almallim and Yamanichi Separation Technique [5]

The process consists of four phases. After the first phase of preprocessing, a word is segmented into "strokes" in the second phase. These strokes represent an approximation of the pen movement during writing, which is thought to be very useful information in the recognition of Arabic handwriting. In the third phase, strokes are classified, and then combined in several steps into a string of characters in the final phase. The four phases are explained in detail as follows:

i) **Preprocessing.** The image of the character is first thresholded (transformed into two gray level images); then, a thinning algorithm is applied on the thresholded image in which the foreground (character) is transformed to a one pixel thick character.

ii) **Segmentation.**

   a) Extraction of the start point (Fig. 3.3).
   
   b) Extraction of the end point (Fig. 3.3).

iii) **Classification of Strokes:**

   ...
Step 1. Grouping

Calculate the following four amounts:

a) The start and end points \((x_{a_1}, y_a), (x_{b_1}, y_b)\)
(Fig. 3.3).

b) The stroke length \((L)\), the number of points the
stroke consists of.

c) The frame of the stroke \(x_{\min'}, x_{\max'}, y_{\min'}, y_{\max'}\).

d) The connection point with the previous stroke
\(X_{\text{conc'}}, Y_{\text{conc}}\).

Based on these four calculations the stroke is classified
into one of the following five groups:

Group 1: Dots.

Group 2: "Hamza".

Group 3: Strokes that contain loops.

Group 4: Strokes with no loops, and end with a branch
or a crosspoint.

Group 5. Strokes with no loops, and with line end.

Step 2. Calculation of feature vector

The feature vectors are group dependent (i.e., different
calculations are done for different groups) (Figs. 3.4a),
(3.6), (3.4c).

Combination of Strokes into a Character String

i) Classify the strokes into main and secondary
strokes.

ii) Assign each secondary to a main stroke.
Fig. 3.3. Amounts calculated for the classification of Group 3 strokes.
**Fig. 3.4(a). Features, codes, and identification vectors of Group 3 strokes**

<table>
<thead>
<tr>
<th>Strokes</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1: yes, 0: no, *: don’t care

1. Is the end point a line end?
2. $0 \leq x < 90$?
3. $x < 0.25$?
4. $(x+y)$ and $(x+y)$ and $(y+y)$?
5. $0.15 < \beta < 0.25$?

**Fig. 3.4(b). Features, codes, and identification vectors of Group 4 strokes**

<table>
<thead>
<tr>
<th>Strokes</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1: yes, 0: no, *: don’t care

1. $x < 0.15$?
2. $120 < \alpha < 220$?
3. $90 < \beta < 225$?
4. $210 < \alpha < 220$?
5. $(x+y) > (x+y)$ or $-210 < \alpha < 15$?

**Fig. 3.4(c). Features, codes, and identification vectors of Group 5 strokes**

<table>
<thead>
<tr>
<th>Strokes</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1: yes, 0: no, *: don’t care

1. Is the stroke almost a straight line?
2. $(x+y)$ and $(y+y)$ are shown below
3. $(y+y)$ and $(x+y)$ shown

---

*The right-hand point*
3.3. Definitions

a) Midline "J mid": The midline is the row that contains the maximum number of Black pixels (Fig. 3.5).

b) $I_{\text{max}}$: Since we scan the image from right to left in the case of Arabic characters, $I_{\text{max}}$ is defined as the first column that contains the first black pixels from the right side (Fig. 3.5).

c) $I_{\text{min}}$: In the first column that contains zero black pixels (Fig. 3.5).

d) $J_{\text{max}}$: Is the last row that contains zero black pixels (Fig. 3.5).

e) $J_{\text{min}}$: Is the first row that contains black pixels (Fig. 3.5).

3.3.1 New Technique of Printed Arabic Character Separation

The technique consists of three phases. In the first phase, which is preprocessing, the word (or line) is thresholded and then thinned.

In the second phase, the midline "Jmid", as well as the borders "$I_{\text{max}}$, $I_{\text{min}}$, $J_{\text{max}}$ and $J_{\text{min}}" are
determined.

In the third phase separation of each character is obtained and by calculating the moment invariants for each character, a recognition decision is arrived at (Figs. 3.6, 3.7).

1) Preprocessing

The image of the character is thresholded then a thinning algorithm is applied on the thresholded image in which the foreground (character) is transformed to a one pixel thick character.

2) Procedure to Determine the Midline and the Borders

a) Initializes \( I_{\text{max}} \) to zero.

b) Initializes \( C_{\text{max}} \) to zero.

c) Set \( I_{\text{min}} \) to a large number.

d) Scan the image row-wise and initialize a counter \( \text{count} \) to zero. Increment the counter, and print the location in case where \( \text{count} \) is greater than \( C_{\text{max}} \). In this case set \( C_{\text{max}} \) to be equal to \( \text{count} \) and \( J_{\text{mid}} \) to be equal to \( J \) (see Fig. 3.6).

3) Procedure for Character Separation

a) Find the values of \( I_{\text{max}}', I_{\text{min}}', J_{\text{max}}', J_{\text{min}}' \) and \( J_{\text{mid}}' \).

b) Define the first breakpoint as \( I_{\text{max}}'+3 \).
Fig. 3.5. Mid line (Jmid), I_{max}, J_{min}, J_{max}, and J_{min} (borders) of the word.
Fig. 3.6. Flowchart to explain the determination of the midline $J_{mid}$. 

Start

Initialize a counter $C_{\text{max}}$ to zero

Scan the image rowwise and initialize a counter "count" to zero

Read each pixel value and increment the counter "count" for each pixel found on the character

If count $> C_{\text{max}}$ yes.

Set $C_{\text{max}} = \text{count}$, $J_{\text{mid}} = J$

and print loc = i

If i $< i_{\text{min}}$ yes

STOP
START

Find Imin, Imax
Jmin, Jmax

Define the first break point as Bp[1] = Imax + 3

Initialize counter count = 1

Set I = Imax - 1

I=I-1

From Jmin to Jmax read pixel value and increment a counter "tt" for each pixel with a value representing a foreground, i.e., a pixel on the character

if tt <= I

yes

I_s = J
Lable the point as start point Is

... continued
Initialize counter \( t_l = 0 \)

For \( J = \text{min}, J_{\text{max}} \) read each pixel increment \( t_l = t_l + 1 \) for every pixel that is a point in the character

- Test if \( t_l > 1 \)
- Set flag false
- Define this point as endpoint \( i_e \)
  - if \( i = i_s \)
    - No break point found
  - no
    - Breakpoint is at \( (i_s - i_e) / 2 + i_e \)

\( l = k - l \)

... continued
Fig. 3.7. Flowchart explaining the new separation technique.
c) Initialize a counter "count" to 1.

d) Set $I = I_{max} - 1$. Since $I_{max}$ represents the column in which the word (or line start) we start our search from the next column.

e) Read the value of the pixels along the axis.

f) Increment a counter "tt" for each point found on the character.

g) If no points or only one point is found along the "J-axis" the line is labelled as the new start point $I_s$.

h) Initialize a counter "ttl" to zero.

i) For $J = J_{min}$ to $J_{max}$ read each pixel and increment the counter "ttl" for each point found on the character.

j) If "ttl" is greater than one set flag false and define this point as an endpoint $I_e$.

k) If $I_e = I_s$ this means that no breakpoint was found. Set $I = I_s - 1$ and check if $I$ is less than or equal to $I_{min}$. If yes, stop.

l) If $I > I_{min}$ repeat the procedure from Step #e.

m) If $I_e \neq I_s$ define a breakpoint "I_s" at $(I_s - I_e)/2 + i_e$ and set $I = I_s - 1$ then check if $I \leq I_{min}$. If yes, stop the procedure.

n) If $I > I_{min}$, repeat the procedure starting at Step #5.

The results of applying this algorithm is discussed in detail in the following chapter.
CHAPTER IV

RESULTS AND CONCLUSION OF APPLYING THE NEW
SEPARATION ALGORITHM AND THE MOMENT INVARIANTS
IN RECOGNIZING PRINTED ARABIC CHARACTERS

4.1 Original Data

By using the original data set to construct the database, recognition rates of 100% was achieved when the learning set and test set are the same, even for the case where characters have great similarity.

4.2 Scaled Characters

Two data sets with characters scaled to scale 0.5 and 0.2 were tested; the rage of recognition obtained were 98% and 95%, respectively.

4.3 Thinned Characters

Three sets of data were processed through thinning.

4.3.1 Original Data Set

In the case of original data set the recognition rate drops from 100% to 95%. This drop can mainly be accounted for by the noise and distortion introduced by the thinning procedure.
4.3.2 **Scaled with 0.5 Scaling Factor**

In this case the recognition rate dropped from 98% to 87% and this drop can be accounted also for by the noise and distortion introduced by the thinning as well as the scaling procedures.

4.3.3 **Scaled with 0.2 Scaling Factor**

In this case the recognition rate dropped from 95% to 82%.

4.4 **Separation of the Characters**

The new separation algorithm introduced in the previous chapter has been applied on different Arabic words.

The first word used was "ّ" which consists of two characters, the "ّ" and the "ّ". This word consists of only one portion (Figs. 4.1, 4.2). The thresholded thinned image of these characters is of size 128 x 128. The midline "Jmid" was found at J=96. The first breakpoint was found at I=64. The separation of the characters has been achieved and is shown in Figs. (4.3, 4.4 and 4.5).

The character "ّ" was chosen for its complexity and uniqueness of its shape.

The second word used to test the algorithm is "فدا" which consists of three characters, the "ف" "دا" and the "دا". This word consists of two portions. The thresholded thinned image of these characters is of size 128 x 128.
The midline "Jmid" was found at J=109. The first breakpoint was found at I=82. The separation of these characters has been achieved. The characters representing these words is chosen such that the "٠" has a dot on top of the character which will represent all the characters with dots above or below it.

The "٠" represents the characters that are not able to be connected from the left side and by these two examples most of the characteristics of the Arabic characters will be all covered.

The third case was a part of a line consisting of two words. In order to illustrate that the previous separation algorithm can also be applied for a line and can be extended to separate the characters in a printed text. The line consists of two words "٠٠٠" (Figs. 4.6, 4.7). In this example, there is one word of one portion and the second word consists of two portions and there is a space separating the two words from each other.

The thresholded thinned image of these words is of size 128 x 128. The midline is found at J=73 and the first breakpoint is found at I=113. The separation of these characters is achieved and is shown in (Figs. 4.8, 4.9 and 4.10).

The separation between the two words did not present any problems.
Fig. 4.1. The word "دو" before thinning algorithm is applied.
Fig. 4.2. The word "٥٥" after thinning algorithm is applied.
Fig. 4.3. The word "هو" after the first character has been separated.
Fig. 4.4. The word "ه" after the second character has been separated.
Fig. 4.5. Calculation for the moment invariant for the first and second characters and their location in the look-up table.
Fig. 4.6. Sentence "هو فدا" after applying the thinning algorithm.
Fig. 4.7. Sentence "هو قدا" after the first character has been separated.
Fig. 4.8. Sentence after the second character has been separated.
Fig. 4.9. Sentence after the third character has been separated.
Fig. 4.10. Sentence after the fourth and fifth characters were separated.
4.5 Summary and Conclusion

The principal objective of the research work was to develop an algorithm for the recognition of connected printed Arabic characters. In order to achieve this goal, the utility of the Moment Invariant technique was first investigated in the case of isolated Arabic characters. The set of moments descriptors developed by Hu in [13] was found to be invariant to translation and size despite the fact that there is a great similarity in the shapes of most Arabic characters. The recognition algorithm was applied to the 112 characters representing the text set. However, the rotational invariance property of the moment descriptors was not tested due to the fact that we were dealing with printed text and not handwritten text. Moment Invariant descriptors did not show any problem in case where the characters had broken parts or dots.

The separation algorithm developed in this thesis was applied on printed characters and proved successful.

In conclusion, the contributions stemming from this thesis can be summarized as follows:

1. The moments invariants technique developed by Hu [13] was investigated in the case of printed Arabic characters.

2. The application of a thinning technique developed in [10] and its effect on the recognition rate was reported.
3. The application of a scaling technique developed in [16] and its effect on the recognition rate was reported.

4. A new separation technique for separating the printed connected Arabic character was developed.
References


APPENDIX I

Separation Algorithm
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include <pip.h>

#define BASEADDR 0X26c
#define MODE 1
#define SPEED 0
#define CLASS 0
#define VIDTYPE 0
#define ZOOM 0
#define num 6
int jmid,imax,imin,j;
int bp[7];
int is,count,i,k,kk,ik;
float f[6];
float mo[22][6];

/* prototypes */
void main(void);
void mid(int *, int *, int *);
void separt(int , int , int , int , int , int *, int *, int *);
void trace(int , int *, int *, int *);
void recognize(int , int , int , int , float *);
void moment( int , int , int , int , float *);
void main()

******************************************************************************
PROCEDURE MAIN IS TO SEPARATE THE CONNECTED PRINTED CHARACTERS AND RECOGNIZE EACH CHARACTER SEPERAT
******************************************************************************
{
 int x1,y1,x2,y2;
 mid(&jmid,&imax,&imin);
 printf("jmid=%d\n",jmid);
 printf("imax=%d,imin=%d\n",imax,imin);
 separt(x1,y1,x2,y2,imax,imin,&count,&k,&kk);
 fg_exit();
}

void init_pip(void)

******************************************************************************
PROCEDURE TO INITIALIZE THE PIP
******************************************************************************
{
 fg_initfmt(BASEADDR,MODE,SPEED,CLASS,VIDTYPE,ZOOM);
 fg_chan (0);
fg_sync (0);
fg_sbuf (0);
fg_quad (1);
}

void mid(int *jmid, int *imax, int *imin)
/* ******************************************
PROCEDURE MID IS TO DETERMINE THE MIDLINE OF THE WORD
OR THE SENTENCES AS WELL AS THE BORDERS.
******************************************/
{
int m, loca;
int count, cmax;
*imax = 0;
*jmid = 0;
cmax = 0;
*imin = 128;
for(j=0; j<127; j++)
{
    count = 0;
    for(i=127; i>0; i--)
    {
        m = fg_pixr(i, j);
        if(m <= 1)
            count = count + 1;
        if(count > cmax)
        {
            cmax = count;
            *jmid = j;
            loca = i;
            if(loca > *imax)
                *imax = loca;
            if(loca < *imin)
                *imin = loca;
        }
    }
}
}

void separt(int x1, int y1, int x2, int y2, int imax, int imin, int *count,
int *k, int *kk)
/* ******************************************
PROCEDURE SEPARAT IS TO DETERMINE THE FIRST BREAKPOINT, THE
START POINT AND DRAW A RECTANGULAR AROUND THE CHARACTER
AFTER DETERMINE THE END POINT FROM SUBROUTINE TRACE.
******************************************/
{
int t, ii, il, ir, tt;
/* imax = *i; */
i = imax - 1;
*k = i;
bp[1] = imax + 25;
/* printf("bp[1]=%d\n",bp[1]); */
*count=1;
while(*k>=imin)
{
  tt=0;
  for(j=0;j<127;j++)
  {
    t=fg_pixmap((int)*k,(int)j);
    if(t <= 1)
      tt=tt+1;
  }
  if(tt <= 1)
  {
    is=*k;
    trace(is,*count,kk,k);
    bp[*count]=*kk;
  }
  *k=*k-1;
}

bp[*count+1]=imin-4;
for(ii=1;ii <=( *count ); ii++)
{
  ir = bp[ii];
  il = bp[ii + 1];
  x1=il;
  y1=10;
  x2=ir;
  y2=100;
  fg_setind(1);
  fg_rect(x1,y1,x2,y2);
  recognize(x1,y1,x2,y2,f);
}

void trace(int is,int *count,int *kk,int *k)
/* *******************************************************************************/
PROCEDURE TRACE IS TO DETERMINE THE ENDPOINT OF
THE CHARACTER AS WELL AS THE STARTPOINT OF
THE FOLLOWING CHARACTER
/* *******************************************************************************/
{
  int score,tt2,tt1,ie;
  *k=*k-1;
  /*printf("is=%d,k=%d\n",is,*k);*/
  score = 1;
  /*printf("k=%d\n",k);*/
  while(score)
  {
    tt1=0;
    for (j=10;j<= 100;j++)
    {
      tt2 = fg_pixmap((int)*k,(int)j)
      ;
    }
  
}
if (tt2 <= 1)
  ttl = ttl + 1;
}
  *k = *k - 1;

if (ttl > 1)
  {
  score = 0;
  }

  ie = *k + 2;
  if (ie < is)
  {
  *count = *count + 1;
  *kk = ((is - ie) / 2 + ie);
  }

#include <math.h>

void recognize(int x1, int y1, int x2, int y2, float *f)

PROCEDURE RECOGNIZE IS TO RECOGNIZE THE CHARACTER
  INSIDE THE RECTANGULAR DRAW BY PROCEDURE SEPART.

{ 

double sqrt();
static float dmin, d;
int i, j, loc;
FILE *in;

moment(x1, y1, x2, y2, f);
in=fopen("table.c", "r");
dmin = (float)20.0;
loc = 0;
for (j = 0; j < 6; j++)
  {
  printf("%6.4f", f[j]);
  printf("\n");
  }
for (i = 1; i < 24; i++)
  {
  d = (float)0.0;
  for (j = 0; j < 6; j++)
    {
      fscanf(in, "%f", &mo[i][j]);
      d = (((f[j] - mo[i][j]) * (f[j] - mo[i][j])) + d ;
    }
  d = (float)sqrt(d);
  if (d < dmin)
    {
      loc = i;
      dmin = d;
    }
```c
void moment(int x1, int y1, int x2, int y2, float *f)
{
    double fabs(), sqrt(), log();
    double m00, m10, m20, m03, m30, m12, m21, m02, m11, m01;
    double a, b, yc, xc, u20, u02, u30, u03, u12, u21, u11, v;
    double c, d, e, r, g, n11, n20, n02, n12, n21, n30, n03;
    int m;
    m00 = 0.0;
    m01 = 0.0;
    m10 = 0.0;
    m20 = 0.0;
    m02 = 0.0;
    m12 = 0.0;
    m21 = 0.0;
    m30 = 0.0;
    m03 = 0.0;

    for(a = (double) y1; a < (double) y2; a++)
    for(b = (double) x1; b < (double) x2; b++)
    {
        m = fg_pixr((int)b, (int)a);
        if (m < (int)1)
        {
            /* compute up to the third order of moments*/
            m00 = m00 + 1.0;
            m10 = m10 + 1.0 * b;
            m01 = m01 + 1.0 * a;
            m11 = m11 + 1.0 * b * a;
            m20 = m20 + ((b * 1.0) * (b * 1.0));
            m02 = m02 + ((a * 1.0) * (a * 1.0));
            m12 = m12 + b * ((a * 1.0) * (a * 1.0));
            m21 = m21 + a * ((b * 1.0) * (b * 1.0));
            m30 = m30 + b * ((b * 1.0) * (b * 1.0));
            m03 = m03 + a * ((a * 1.0) * (a * 1.0));
        }
    }
    yc = m01 / m00;
    xc = m10 / m00;
    printf("xc=%6.4e,yc=%6.4e\n", xc, yc);
    /* compute central moments from ordinary moment*/
    u11 = m11 - yc * m10;
    u20 = m20 - xc * m10;
    u02 = m02 - (yc * m01);
    u30 = m30 - (3 * xc * m20) + 2 * (m10 * (xc * xc));
    u03 = m03 - (3 * xc * m02) + (2 * m01 * (yc * yc));
    u12 = m12 - (2 * yc * m11) - (xc * m02) + 2 * (yc * yc * m01);
    u21 = m21 - (2 * xc * m11) - (yc * m20) + (2 * xc * xc * m01);
    printf("u12=%6.4e,u21=%6.4e,u20=%6.4e,u03=%6.4e\n", u12, u21, u20, u03);
    /* to normalize the central moments*/
```
a = (m00*m00 );
b = (a *sqrt (m00));
n20 = (u20/a);
n02 = (u02/a);
n11 = (u11/a);
n12 = (u12/b);
n21 = (u21/b);
n30 = (u30/b);
n03 = (u03/b);
a = n20 - n02;
b = n30 - 3 * n12;
c = n30 + n12;
d =3 *n21-n03;
e = n21 + n03;
r =3 * n12 - n30;
g = 3 * n12 - n03;
v = (r*c)*(c*c);

/* compute the invariant moment*/

f[0] = fabs((n20 + n02)); /*
printf("hi i am here\n");
f[0]=(float)fabs(log(fabs((a*a)+4*(n11*n11))));
f[1] = (float)fabs(log(fabs((b*b)+(d*d))));
f[2] = (float)fabs(log(fabs((c*c)+(e*e))));
f[3] = (float)fabs(log(fabs((b*c)*(c*c)-(3*(e*e))+(d*e)*(3*(c*c))-
-f[4] = (float)fabs(log(fabs((a*(c*c))-(e*e)+(4*n11*c*e))));
f[5] = (float)fabs(log(fabs((v)-(3*(e*e))+(g*e)*(3*(c*c)))-(e*e))
}
APPENDIX II

Recognition Algorithm
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <math.h>
#include <float.h>
#include <pip.h>

#define BASEADDR 0x26c
#define MODE 1
#define SPEED 0
#define CLASS 0
#define VIDTYPE 0
#define ZOOM 0
#define num 6

cchar workbuffer [4][512];
int i,j,k,l;
float mo[18][6];
float f[6];

/* prototypes */
void main(void);
void table(int *,int *,int *);
void recognize(int *,int *,int *);
void init_pip(void);
void load_img(void);
void mask(int,int,int);
void keymask(int *,int *,int *);
void moment(int *,int *,int *,int,int,float *);
void wind_img(int *,int *,int *,int,int,int);
void movemask(int *,int *,int *,int,int,int);
void main()

******************************************************************************

PROCEDURE MAIN IS A PROGRAM IN WHICH A WINDOW IS DRAW AROUND EACH CHARACTER IN THE IMAGE CALCULATING THE MOMENT INVARIANT DESCRIPTORS AND DEFINE THE LOCATION OF EACH CHARACTER IN THE LOOK-UP TABLE.
******************************************************************************
{
static int x,y,size;
x=412;
y=1;
size=41;

init_pip();
load_img();
fg_sbuf(1);
mask(x,y,size);
recognize(&x,&y,&size);
 /**<table(&x,&y,&size);* /
 fg_exit();
}

void table(int *x, int *y, int *size)
 /**<table(&x,&y,&size);*/
 PROCEDURE TABLE IS TO CALCULATE THE MOMENT INVARIANT
 DESCRIPTORS FOR EACH CHARACTER IN THE IMAGE AND ARR-
 ANGED IN A TABLE (LOOK-UP TABLE).

 FILE *out;
 int number,i,j;
 static int xrel,yrel,srel;

 for(i=1;i<18;i++)
 {
  keymask(x,y,size);
  printf("enter number 1 if you stop moving\n");
  scanf("%d",&number);
  if (number==1) moment(x,y,size,xrel,yrel,srel,f);
  for(j=0;j<6;j++)
  {
   mo[i][j] = f[j];
  }
}

 out=fopen("table.c", "w");
 for(i=1;i<18; i++)
 {
  for(j=0;j<6;j++)
  {
   fprintf(out,"%6.4f ",mo[i][j]);
  }
  fprintf(out,"\n");
  fclose(out);
  printf("hi iam done\n");
}

#include <math.h>
void recognize(int *x, int *y, int *size)
 /**<table(&x,&y,&size);*/
 PROCEDURE RECOGNIZE IS TO DETERMINE THE CHARACTER LOCATION
 IN THE LOOK-UP TABLE.

 double sqrt();
 static int xrel,yrel,srel;
 static float dmin,d;
 int i,j,number,loc;
 FILE *in;
 keymask(x,y,size);
 printf("enter number 1 if you stop moving\n");
 scanf("%d",&number);
if (number = 1) moment(x, y, size, xrel, yrel, srel, f);
in=fopen("table.c", "r");
dmin=(float)200.0;
loc=0;
  for(j=0; j<6; j++)
  {
    printf("%6.4f", f[j]);
    printf("%\n");
  }
for(i=1; i<18; i++)
{
  d=(float)0.0;
  for(j=0; j<6; j++)
  {
    scanf((in,"%f", &mo[i][j]);
    d= ((f[j] -mo[i][j])*(f[j] -mo[i][j]))+d ;
  }
  d=(float)sqrt(d):
  if(d < dmin)
  {
    loc =i;
    dmin = d;
  }
}
printf("loc=%d, d=%f, dmin=%f\n", loc, d, dmin);

void init_pic(void)
/*****************************/
PROCEDURE INIT IS TO INITIALIZE THE SCREEN TO DISPLAY
THE IMAGE OF THE CHARACTERS
/*****************************/
{ fg_initfmt(BASEADDR, MODE, SPEED, CLASS, VIDTYPE, ZOOM);
  fg_chan (0);
  fgSync (0);
  fg_sbuf (0);
  fg_quad (1);
}
define BSIZE 4096
define QUAD 0
void load_img()
/*****************************/
PROCEDURE LOAD_IMG IS TO LOAD THE IMAGE OF THE
CHARACTERS TO THE BUFFER
/*****************************/
{
char workbuff[4096];
char file [20];
fg_quad (0);
printf("enter input file name
");
scanf("%s", file);
fg_frdisk(BSIZE, QUAD, file, workbuff);
} 
void mask(int x, int y, int size)
/**********************************************************************
PROCEDURE MASK IS TO DRAW A WINDOW(SEPARATE) AROUND THE
CHARACTER SO YOU CAN RECOGNIZE EACH CHARACTER SEPARATELY
***********************************************************************/
{
extern char workbuffer [4][512];
int s2,x1,x2,y1,y2;
s2=size/2;
x1 = x - s2;
x2 = x + s2;
y1 = y - s2;
y2 = y + s2;
fg_colr(x1,0,workbuffer[0]);
fg_colr(x2,0,workbuffer[1]);
fg_rowr(y1,0,workbuffer[2]);
fg_rowr(y2,0,workbuffer[3]);
fg_setind (i);
fg_border (x-s2,y-s2,x+s2,y+s2,1);
}
#define KEYPORT 0X60
#define KEYMASKPORT 0X21
#define KEYMASK 0X02

void keymask(int *x,int *y,int *size)
/**********************************************************************
PROCEDURE KEYMASK IS TO LOCATE THE BORDER OF THE
WINDOW TO BE DRAWN AROUND THE CHARACTER
***********************************************************************/
{
int ch,bflag,xrel,yrel,srel,step,i,intmask;
intmask=inp(KEYMASKPORT);
outp(KEYMASKPORT,intmask | KEYMASK);
step=1;
i = 0;
while ((ch = inp(KEYPORT)) != 0X1C)
{
bflag = 1;
srel = 0;
xrel = 0;
yrel = 0;
if (ch < 0X80)
{
switch(ch)
{
case 0X48 : yrel = -step;
break;
case 0X50 : yrel = step;
break;
case 0X4B : xrel = -step;
break;
case 0X4D : xrel = step;
break;
}
case 0x49 : xrel = step;
yrel = -step;
break;
case 0x51 : xrel = step;
yrel = step;
break;
case 0x47 : xrel = -step;
yrel = -step;
break;
case 0x4f : xrel = -step;
yrel = step;
break;
case 0x52 : srel = step;
xrel = 255 - *x;
yrel = 255 - *y;
break;
case 0x53 : srel = -step;
xrel = 255 - *x;
yrel = 255 - *y;
break;
case 0x4c : xrel = 255 - *x;
yrel = 255 - *y;
if(! (xrel == yrel)) bflag = 0;
break;
default : bflag = 0;
break;
}
if (xrel == yrel) movemask(x,y,size,xrel,yrel,srel);
if (i++ >= 5)
{
step = step + 2;
i = 0;
}
outp(KEYMASKPORT, intmask);
}
void movemask(int *x, int *y, int *size, int xrel, int yrel, int srel)
******************************************************************************
PROCEDURE MOVEMASK IS TO BE ABLE TO MOVE THE WINDOW ALONG
THE IMAGE FROM ONE CHARACTER TO ANOTHER
******************************************************************************
{
extern char workbuffer[4][512];
int s2,x1,x2,y1,y2;
s2 = *size/2;
x1 = *x - s2;
x2 = *x + s2;
y1 = *y - s2;
y2 = *y + s2;
*size += srel;
if (*size < 3) *size = 3;
if (*size > 101) *size = 101;
fg_colw(x1,0,workbuffer[0]);
fg_colw(x2,0,workbuffer[1]);
fg_roww(y1,0,workbuffer[2]);
fg_roww(y2,0,workbuffer[3]);
*x += xrel;
*y += yrel;
if (*x > (511-s2)) *x = *x - 513 + *size;
if (*y > (511-s2)) *y = *y - 513 + *size;
if (*x < s2) *x = 510 - *x;
if (*y < s2) *y = 510 - *y;
mask(*x,*y,*size);

void wind_img(int *x,int *y,int *size,int xrel,int yrel,int srel)
{
    char workbuffr [4096];
    char filename[20];
    int s2,x1,x2,y1,y2;
    *x += xrel;
    *y += yrel;
    *size += srel;
    s2 = *size/2;
    x1 = *x -s2;
    x2 = *x + s2;
    y1 = *y - s2;
    y2 = *y + s2;
    printf("enter output filename\n");
    scanf("%s",filename);
    fg_setwin(x1,y1,x2,y2);
    fg_wintodisk(BSIZE,filename,workbuffr);
}

#define num 6

void moment(int *x,int *y,int *size,int xrel,int yrel,int srel,float
PROCEDURE TO CALCULATE THE MOMENT INVARIANT DESCRIPTOR
FOR EACH CHARACTER LOCATED INSIDE THE DEFINED WINDOW

{ double fabs(),sqrt(),log();
double m00,m10,m20,m03,m30,m12,m21,m02,m11,m01;
double a,b,yc,xc,u20,u02,u30,u03,u12,u21,u11,v;
double c,d,e,r,g,n11,n20,n02,n12,n21,n30,n03;
int s2,x1,x2,y1,y2,m;
*x += xrel;
*y += yrel;
*size += srel;
s2 = *size/2;
x1 = *x - s2;
x2 = *x + s2;
y1 = *y - s2;
y2 = *y + s2;
m00 = 0.0;
m01 = 0.0;
```c
m10=0.0;
m21=0.0;
m12=0.0;
m02=0.0;
m20=0.0;
m30=0.0;
m03=0.0;

for(a = (double) y1; a < (double) y2 ; a++)
for(b =(double) x1; b < (double) .x2 ; b++)
{
  m =rg_pixr((int)b,(int)a);
  if (m<(int)1)
  {
    /* compute up to the third order of moments*/
    m00 = m00 +1.0;
    m10 = m10 +1.0 * b;
    m01 = m01 +1.0 * a;
    m11 = m11 +1.0 * b * a;
    m20 = m20 +((b *1.0)*(b * 1.0));
    m02 = m02 +((a *1.0)*(a * 1.0));
    m12 = m12 +b * ((a *1.0)*(a *1.0));
    m21 = m21 + a * ((b *1.0)*(b * 1.0));
    m30 = m30 + b * ((b *1.0)*(b *1.0));
    m03 = m03 + a * ((a *1.0)*(a *1.0));
  }
}
printf("m20=%6.4e,m30=%6.4e\n",m20,m30);
printf("hi i am here\n");
yc = m01 / m00;
xc = m10 / m00;
printf("xc=%6.4e,yc=%6.4e\n",xc,yc);
/* compute central moments from ordinary moment*/
u11 = m11 - yc * m10;
u20 = m20 - xc * m10;
u02 = m02 - (yc * m01);
u30 = m30 - (3 * xc * m20) + 2 *(m10 *(xc*xc));
u03 = m03 - (3 * yc * m02) +(2 * m01 *(yc*yc));
u12 = m12 - (2 *yc *m11) -(xc *m02) +2*(yc*yc*m01);
u21 = m21 - (2 * xc *m11)-(yc *m20) +(2*xc*xc*m01);
printf("u21=%10.6e,u12=%6.4e,u20=%6.4e\n",u21,u12,u20);
/* to normalize the central moments*/
a = (m00*m00);
b = (a *sqrt (m00));
n20 = (u20/a);
n02 = (u02/a);
n11 = (u11/a);
n12 = (u12/b);
n21 = (u21/b);
n30 = (u30/b);
n03 = (u03/b);
a = n20 - n02;
b = n30 - 3 * n12;
c = n30 + n12;
```
d = 3 * n21 - n03;
e = n21 + n03;
r = 3 * n12 - n30;
g = 3 * n12 - n03;
v = (r*c)*(c*c);

/* compute the invariant moment */

/* f[0] = fabs((n20 + n02)); */
printf("hi iam here\n");
f[0] = (float)fabs(log(fabs((a*a)+4*(n11*n11))));
f[1] = (float)fabs(log(fabs((b*b)+(d*d))));
f[2] = (float)fabs(log(fabs((c*c)+(e*e))));
f[3] = (float)fabs(log(fabs((b*c)*(c*c)-(3*(e*e))+(d*e)*(3*(c*c)))));
f[4] = (float)fabs(log(fabs((a*(c*c))-e*(e)+(4*n11*c*e))));
f[5] = (float)fabs(log(fabs((v)-(3*(e*e))+((g*e)*(3*(c*c)))-(e*e))));
Vita Auctoris

1957  Born on October 3, Cairo, Egypt.
1980  Graduated from Assiut University, Egypt with the degree of Bachelor of Science in Electrical Engineering.
1986  Candidate for the Master degree in Electrical Engineering, University of Windsor, Windsor, Ontario, Canada.