<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>•Editorial Board</td>
<td>i</td>
</tr>
<tr>
<td>•Editor –in-Chief’s Introduction by Dele Oluwade</td>
<td>1</td>
</tr>
<tr>
<td>•Industry Perception of the Software Inspection Process: Nigeria Software Industry as a Case Study by Olalekan S. Akinola, Adenike O. Osifisan &amp; Opeoluwa B. Akinkunmi</td>
<td>3</td>
</tr>
<tr>
<td>•An Enhanced LZW Text Compression Algorithm by Adeniyi Moronfolu &amp; 'Dele Oluwade</td>
<td>13</td>
</tr>
<tr>
<td>•A Performance Analysis of Delta and Huffman Compression Algorithms by A.O. Oke, O.A. Fakolujo &amp; O.J. Emuoyibofarhe</td>
<td>41</td>
</tr>
<tr>
<td>•General Interest News</td>
<td>55</td>
</tr>
<tr>
<td>•Instruction to Authors, Subscribers and Advertisers</td>
<td>57</td>
</tr>
</tbody>
</table>

Editor-in-Chief: Dele Oluwade
EDITORIAL BOARD

The editorial board members are described below. Their respective areas of primary interest are also indicated.

EDITOR – IN – CHIEF

Dele Oluwade PhD, Senior Member IEEE
Federal University of Technology, Minna
E-mail: oluwade@sigmaxi.net; bamidele@aims.ac.za
(Computer Science & Information Technology; Discrete Modelling)

(ASSOCIATE) EDITORS

A.M. Awobode, PhD
University of Illinois at Urbana Champagne
Email: awobodea@yahoo.com
(Physics of Computation)

E.O. Ayoola, PhD
University of Ibadan
Email: eoayoola@ictp.it
(Mathematics of Computing; Numerical Analysis & Scientific Computation)

Emmanuel Oluremi Bejide PhD, MIEEE
University of Cape Town
Email: ebejide@ebe.uct.ac.za
(Communications Engineering)

Amos David, PhD
Universite Nancy 2
Email: Amos.David@loria.fr
(Computer Science; Strategic Information & Economic Intelligence Systems)

Khalil Ezzinbi, PhD
Cadi Ayyad University
Email: ezzinbi@ucau.ac.ma
(Mathematics of Computing)

A.O. Fakolujo PhD, Senior Member IEEE
University of Ibadan
Email: ao.fakolujo@mail.ui.edu.ng
(Communications Engineering)

Oliver E. Osuagwu PhD, MIEEE
Federal University of Technology, Owerri
Email: drosuagwu@yahoo.com
(Computer Science and Engineering; Data Communications & Network Engineering)

A.S. Ukpera, PhD
Obafemi Awolowo University, Ile-Ife
Email: aukpera@oauife.edu.ng
(Mathematics of Computing; Modelling)

Abdullahi Umar, PhD
King Fahd Univ. of Petroleum & Minerals
Email: aumar@kfupm.edu.sa
(Mathematics of Computing; Computer Algebra)

J. Michel Tchuenche, PhD
University of Tanzania
E-mail: jmt_biomaths@yahoo.com
(Mathematics of Computing; Mathematical Biology)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Submission of a manuscript and other editorial correspondence should be sent to the editor-in-chief electronically. Alternatively, it may be submitted through any other member of editorial board. In the latter case, the editor-in-chief should be copied for record purposes. Please see the inside back cover for details of Instruction to Authors, Subscribers and Advertizers.

Editorial Assistant: Ramatu Abdulsalam
(email: ramatayo@yahoo.com)
EDITOR-IN-CHIEF’S INTRODUCTION

Dele Oluwade, SMIEEE
Dewade Systems Consult,
G.P.O. Box 4222,
Minna, Niger State,
Nigeria
Email: dewadeconsult@yahoo.com

This issue marks the end of Volume 2 of the African Journal of Computing & ICT and the impending transition to the third year of the journal.

There is no doubt that the journal has faced a lot of challenges in her march to maturity. These challenges may not necessarily be peculiar to the journal but are inevitable as a result of the social climate of the environment in which the journal is published.

This edition contains 4 reviewed papers. In their paper, the trio of Olalekan Akinola, Professor Adenike Osofisan and Dr. Opeoluwa Akinkunmi examined the concept of software inspection in software engineering with a focus on the Nigerian software industry. By distributing some questionnaires and analyzing the responses, the authors conclude that software inspection is a highly neglected necessary step in the software development process in Nigeria. The paper is entitled, “Industry Perception of the Software Inspection Process: Nigeria Software Industry as a Case Study”.

The paper by Adeniyi Moronfolu and Dr. ‘Dele Oluwade, entitled, “An Enhanced LZW Text Compression Algorithm”, dwelt on the subject matter of data compression. In the paper, the authors studied both the popular Run Length Encoding (RLE) and Lempel-Ziv-Welch (LZW) text compression algorithms and derived a hybrid algorithm which enhances the performance of the latter without compromising the speed of execution.

In their own paper, “Face Image Processing, Analysis and Recognition Algorithms for Enhanced Optimal Face Recognition Systems Design: A Comparative Study”, Andrew Akpan and Dr. Reginald Osakwe present an x-ray of face image recognition systems. The authors studied ten different face image samples each of forty eight different individuals taken under different light intensities and analyzed them using thirteen different mathematical algorithms available from MATLAB® image processing toolbox. The analyses show significant differences for all the face image samples.
The last, but certainly not the least, paper in this edition is authored by the trio of A. O. Oke, Dr. O. A. Fakolujo and Dr. O. J. Emuoyibofarhe. This paper, like that of Adeniyi Moronfolu and Dr. ‘Dele Oluwade, is on data compression. In it, the authors perform a comparative analysis of the Delta and Huffman data compression algorithms by implementing them using the programming language C#. It is shown that Huffman algorithm performs better since it is able to achieve a higher reduction in file size.

The journal would like to acknowledge Dr. Ibrahim Haruna and John Al-Hassan who are additions to the list of reviewers. This is to appreciate all the (potential) authors, reviewers, editors, subscribers and advertizers. The journal would continue to receive submitted papers at any time of the year and publish accepted papers in any of the yearly two editions.
INDUSTRY PERCEPTION OF THE SOFTWARE INSPECTION PROCESS: NIGERIA SOFTWARE INDUSTRY AS A CASE STUDY

Olalekan S. AKINOLA¹, Adenike O. OSOFISAN ², Babatunde O. AKINKUNMI ³
Department of Computer Science,
University of Ibadan,
Ibadan,
Nigeria.
Email:¹Solom202@yahoo.co.uk (corresponding author),
²mamoshof@yahoo.com,
³bo.akinkunmi@mail.ui.edu.ng

Received 02 May, 2009

ABSTRACT

Software inspection has been well incorporated into the software development process of software development organizations across the developed countries in the world. A critical observation of the counterpart organizations in developing countries especially Africa shows that this software quality assurance is highly compromised in their software development process. A survey was conducted to feel the pulse of the software professionals in Nigeria on their perceptions of the software inspection as a software quality assurance activity, via a structured questionnaire research instrument. A total of 75 questionnaires were administered and only 68 were returned. 65 out of the returned were found to be useful. Briefly, the results show that software inspection is highly neglected as a step in their software development process either because they are not aware of the process or they feel it is a waste of time since they have good programmers who can turn out codes in just few days. The implication of these results is that bugs that may be painful may be hidden in the software artifacts being produced by the professionals.

Keywords: Software, Software Inspections, Software Industry, Perception, Nigeria

1. INTRODUCTION

One of the most effective ways to promote quality and productivity in software development is the use of software inspections as a step in the software development process [14,15,16]. The primary purpose of a software inspection is to identify defects existing within software work products developed throughout the
development process (e.g., user requirements specifications, design documents, code). It is not only used to correct the defect, but also to evaluate and improve the development process itself. Software inspection is a peer review in which a small group of software developers (usually between 3 to 8) examine another software developer's work [14,16].

As part of the effort to determine the extent to which software inspection as a step in software quality assurance is embraced and included in the software development process of the software development houses in Nigeria, a questionnaire was designed and administered to software developers in some software development houses within our reach in Ibadan and Lagos. Lagos happens to be one of the largest Nigerian commercial centres with majority of software houses sited in the centre [3,13].

A total of 75 questionnaires were administered and only 68 were returned. 65 out of the returned were found to be useful. Three others were either incompletely filled or were found to contain doubtful information. We made sure that only organizations having software development as one of their business areas were contacted for the questionnaire survey. The respondents were systems analysts, designers or programmers who have been in the software development business for at least five years. The data collected were analyzed using Statistical Package for Social Sciences (SPSS) version 12.0. Descriptive statistics such as frequency and simple percentages were used for the interpretation of the data collected. Next, we present the results from the industry survey.

The rest of this paper is as follows. In section 2, we present an overview of software inspection process, while the structure of our questionnaire research instrument is discussed in section 3. In section 4, results from the survey are presented. Discussion of the results are discussed in section 5 and section 6 concludes the paper with some recommendations.

2. AN OVERVIEW OF SOFTWARE INSPECTION

Software quality is the user’s perception of how software works. Inspection is one of the methods for ascertaining the quality of software systems throughout the system development lifecycle. Software inspection is an industry-proven type of peer review for detecting and removing defects as early as possible in the software development lifecycle, then reducing rework [7,9,18]. It is actually regarded as an industry’s best practice for delivering high-quality software [17]. Prior studies indicate that inspections can detect as little as 20% [6] to as much as 93% [8] of the total number of defects in an artifact (requirement document, design document or code document ) [12].

Starting from the landmark work of Fagan in 1976 to the modern global practice in software inspection, software inspections is made up of six major steps: planning, overview, preparation, inspection meeting, rework and follow up [8]. Activities involved in each of these steps have been widely reported in the literatures, for examples, [9,11].

Laitenberger [11] in his survey of software inspection technologies asserts that “One prevalent reason for the use of inspection technology in software projects is the
inevitability of defects. Even with the best development technologies in place, defects cannot be completely avoided. This stems from the fact that software development is a human-based activity, and, thus, prone to defects. A defect can be characterized as any product anomaly, that is, any deviation from the required quality properties that needs to be tracked and resolved”.

In terms of industry acceptance, many works in the literatures establish that software inspection has been duly incorporated into the software development process of software houses especially in the western nations of the world. For instance, [9] reports that software inspections have been adopted for many years by industrial organisations because of their impact on product quality and cost of non-quality. The work of Porter et al.,[14] at Lucent Technologies Naperville, USA, Nachiappan et al.,[12] at Nortel Networks, Research Triangular Park, NC, USA , and Basili et al., [4] at Software Engineering laboratory at NASA suggest that software inspection is widely accepted at industry level.

Chatzigeorgiou and Antoniadis [5] in their article reports that The number of software organizations that have incorporated formal reviews in their development process is constantly increasing and the belief that efficient inspections cannot only detect defects but also reduce cycle time, increases process visibility, improves programmers’ capability and lower costs is spreading. Tyran [16] and Harjumaa et al., [10] also reports that inspections have gained wide acceptance as a development tactic and can take up to 15 percent of the time allotted to a software project.

Considering the important roles played by software inspection in ascertaining the quality of software products, we thereby conducted a survey on the perception of and level of adoption of software inspection by the emerging software development houses in Nigeria.

3. RESEARCH INSTRUMENT

A close – ended structured questionnaire was designed for capturing the data used in this survey. The questionnaire is divided into two sections with a total of ten (10) cogent questions. Information on the biography of the respondents were sought for in section A while the perceptions of the respondents on software inspection as part of the software quality assurance measure were stated in section B. Question numbers 1 to 4 in Table 1 form the section A while question numbers 5 to 10 were in section B. A total of 75 questionnaires were administered and only 68 were returned. 65 out of the returned were found to be useful.
4. RESULTS

Figs. 1.1 to 1.10 show the pie charts of the results obtained from the survey study.

**Fig. 1.1 Major Business concern**

Fig. 1.1 shows that 50% of the software professionals surveyed were into software development. Others engaged in hardware cloning and networking in addition to software development.

**Fig. 1.2 Number of years of operation**

From Fig. 1.2, 49% of the professionals surveyed have been in the software development business for about 5 years while 32% said they have been in the industry for between 6 to 10 years. Only a few of them (3%) said they have been in the business for more than 15 years.

**Fig. 1.3 Number of software developers employed**

Fig. 1.3 shows that 46% of the software developers surveyed said between 6 to 10 professionals were employed by their organizations, while 42% said about 5 professionals were employed by their own organizations.
Fig. 1.4 Types of software produced (Tick as many)

Fig. 1.4 shows that majority (60%) of the software professionals surveyed are into the development of general application software, followed by web applications (34%).

Fig. 1.5 Awareness of software inspection as a major part of software development process

More than half (57%) of the professionals surveyed claimed that they were not aware of software inspection as a major part of software development process (Fig. 1.5). Out of the 43% that claimed the awareness of software inspection, Fig. 1.6 shows that only 17% of them did actually incorporate software inspection into their software development process.

Fig. 1.6 Real incorporation of software inspection in the organization’s software development process

Fig. 1.7 Inspection Method(s) adopted

Fig. 1.7 shows that meetingless (nominal) inspection method, where there is no interaction of inspectors on the software artifact is usually adopted by the fewer
practitioners for their inspection exercise (64%).

Fig. 1.8 Why inspection is not incorporated

From Fig. 1.8, 43% of the practitioners were of the opinion that if there are good programmers on ground, there is no need of inspection. 32% said that software inspection step will amount to additional overhead cost for the organization, while 25% believed it is just a waste of time, “when there are deadlines waiting there for you”.

Fig. 1.9 Why Software Inspection should be incorporated into software development process

Fig. 1.9 shows that in most (82%) of the software houses surveyed, the same set of personnel does the software development process altogether. There were no clear demarcations of roles among the professionals.

Fig. 1.10 How the software process is carried out

Fig. 1.10 shows that the software professionals surveyed agreed to the fact that software inspection is necessary in the development of software because it will enhance productivity (53%) as well as skills (47%) of professionals.

5. DISCUSSION

Major results from this study show that most of the software houses covered in the survey are about 5 years old (49.2%) and are into general application software development (80%). The applications they develop are normally targeted to business organizations and possibly educational packages. 44.6% of them are into web application developments, targeted mainly towards the banking sector of the economy.
Only 43.1% of the organizations surveyed said that they were aware of the software inspection process as part of the major steps to achieve quality in software products; and 16.9% out of the 28 organizations that said they were aware of the process actually incorporate it into their software development process. A critical examination of the results shows that only 2 (3.1%) said they are using the formal meeting inspection method. Further verbal interrogations from these respondents revealed that they actually held regular inspection meetings on their software artifacts (design and code documents) with about 5 professionals.

7 (10.8%) said they use the meaningless inspection method. When they were asked how they do this, they said that the individual programmers will study his/her designs and codes, but may call a colleague to also cross-examine the document for him/her. In order to cross-examine this claim, further results from the study reveals that in most of the organizations covered in the survey, the same set of personnel (81.5%) does all the phases of the software development process (requirements analysis, design, coding and testing).

However, all the software organizations surveyed agree to the fact that the incorporation of software inspection technique will enhance their product quality, professional skills as well as decreasing development costs.

Results from this survey show that majority of the software houses in the country are either not aware of the quality assurance technique or think that it is just a waste of time. Their major emphasis is always on “good programmers who can turn out code in just few days”. The result also shows that there is no clear delineation of roles for the software professionals in most of the organizations surveyed. The same set of people does analysis, design, coding and testing on software projects being handled in most of the organizations. This is because most of these organizations have a few number of professionals employed.

Although, a few organizations are beginning to designate some staffers as “testers”, they are very few and not many resources are committed to their activities within the organizations. As a matter of fact, some organizations only deploy interns as testers.

Akinkunni et al (2004) had suggested that Nigerian software industry compromises several aspects of the software process, with attendant effect on the quality of the output. The result of this research confirms that position with respect to software inspection.

4. CONCLUSION

Software inspection is an essential constituent of the software quality assurance process. The fact that the software industry in Nigeria does not fully embrace industry-academic collaboration research calls for urgent attention. The industry is characterized with process compromise (not following the process of software development diligently), resistance to measurement and resistance to academic collaborations. To get information on their software projects is not easy. However, we attributed all these problems to the fact the industry is just coming into lime light, and that everybody wants to penetrate the market fast in the country. The result of which might lead to ‘software crisis’ and low quality turn out for the software products. The industry needs to address this issue urgently.
REFERENCES


Olalekan S. Akinola is a Lecturer of Computer Science at the University of Ibadan, Nigeria. He specializes in Software Engineering with special focus on Software Quality Assurance.

Adenike O. Osofisan is a Professor of Computer Science at the University of Ibadan, Nigeria. She is currently the Head of Department of Computer Science in the University.

Opeoluwa B. Akinkunmi is a Senior Lecturer in the Department of Computer Science, University of Ibadan, Nigeria. His research focuses on Knowledge Representation and Software Engineering.
ADVERTIZE YOUR PRODUCTS AND SERVICES IN THE

AFRICAN JOURNAL OF COMPUTING AND ICT

(ISSN 2006 -1781)

FOR DETAILED INFORMATION ON SUBSCRIPTION AND ADVERTIZEMENT RATES, PLEASE CONTACT THE PRINCIPAL CONSULTANT
(dewadeconsult@yahoo.com)
AN ENHANCED LZW TEXT COMPRESSION ALGORITHM

Adeniyi Moronfolu
% Department of Computer Science,
University of Ibadan,
Oyo State,
Nigeria.
Email: saintniyi@yahoo.com

&

‘Dele Oluwade
Dewade Systems Consult
G. P. O. Box 4222,
Minna, Niger State,
Nigeria.
Email: deleoluwade@yahoo.com

Received 21 October, 2006

Thesis Communication

ABSTRACT

Data compression is the art or science of representing information in a compact form. This compact representation is created by identifying and using structures that exist in the data. Data can be characters in text file, numbers that are samples of speech or image waveforms, or sequences of numbers that are generated by other processes. Most text files contain a lot of redundancies such as multiple occurrences of characters, words, phrases, sentences and even spaces. These redundancies are the structures which most compression algorithms tend to exploit to bring about reduction in the amount of data. Achieving a high amount of compression depends not only on the compression algorithm used but also on the types of data that is being compressed. The Lempel-Ziv-Welch (LZW) algorithm is known to be one of the best compressors of text which achieve a high degree of compression. This is possible for text files with lots of redundancies. Thus, the greater the redundancies, the greater the compression achieved. However, the LZW algorithm can be further enhanced to achieve a higher degree of compression without compromising the speed of compression and decompression. In this paper, an improved hybrid algorithm, called RLW algorithm, is presented. The algorithm combines the run length encoding (RLE) algorithm with the LZW algorithm with a view to enhancing the
performance of the LZW algorithm without compromising the speed of execution.

Keywords: Text compression, Lempel-Ziv-Welch (LZW) algorithm, Run Length Encoding (RLE) algorithm,

I. INTRODUCTION

In the last decade, there have been a transformation in the way we communicate, and the process is still under way. This transformation includes the ever-present, ever-growing internet; the explosive development of mobile communications; and the ever-increasing importance of video communication. Data compression is one of the enabling technologies for these aspects of the multimedia revolution. It would not be practical to put images, let alone audio and video, on websites if it were for data compression algorithms. A data compression technique is said to be lossless if the decompressed (i.e restored) data file is identical to or is an exact replica of the original data [9]. It is said to be lossy if it only generates an approximation of the original file. Lossy image compression is used in digital cameras, greatly increasing their storage capacities while hardly degrading picture quality at all. Cellular phones would not be able to provide communication with increasing clarity were it not for compression. The advent of digital TV service would not be possible without compression. Making a long distance call is possible by the use of compression. Voice compression is used in internet telephony for example, while audio compression is used for CD ripping and is decoded by MP3 players [11]. Even modern, fax machine and satellite TV service are using compression. The applications are endless.

Data compression operates in general by taking “symbols” from an input “text”, processing them, and writing “codes” to a compressed file. Symbols are usually bytes, but they could just as easily be pixels. To be effective, a data compression scheme needs to be able to transform the compression file back into an identical copy of the text [10]. Although, there are many types of compression techniques in existence today, the goals are all basically the same i.e. to make data as succinct as possible in the smallest possible time. This has direct application in storage conservation algorithm and in data transmission among many other applications [14]. The use of a compression algorithm is dependent on the efficiency of the algorithm, the nature of data to be compressed, as well as the elapsed time for which the compression and decompression can take place. Different compression algorithms are suited for different types of data e.g. JPEG (lossy) algorithm is appropriate for picture data compression, MPEG (lossy) algorithm for video data compression while Huffman, LZW, Run Length Encoding (RLE) lossless algorithms etc are suited for text data compression. In particular, the RLE algorithm is an algorithm used to reduce the size of a repeating string (called run) of characters. This is typically achieved by encoding a run of symbols into two bytes, namely a count and a symbol. The LZW, on the other hand, is an algorithm which maps strings of text characters into numeric code.

In [2, 3, 4, 5, 6, 7], a binary lossless text compression algorithm (called ‘code presentation’) was developed and applied to some model binary text data, notably American Standard Code for Information Interchange (ASCII), Extended Binary Coded Decimal Interchange Code (EBCDIC) and the International Telegraph
and Telephone Consultative Committee (CCITT) code.

In the present paper, an enhanced LZW compression algorithm, called RLW algorithm, is presented. This algorithm is an hybrid of the RLE and LZW algorithms which enhances the performance of the latter without compromising the speed of execution. A comparison of RLE, LZW and RLW in terms of the file sizes shows that the new algorithm (RLW) performs best.

II. RLE AND LZW ALGORITHMS

Run-length Encoding (RLE) is a technique used to reduce the size of a repeating string of characters. This repeating string is called a run. Typically, RLE encodes a run of symbols into two bytes; a count and a symbol. RLE can compress any type of data regardless of its information content, but the content of data to be compressed affects the compression ratio. RLE cannot achieve high compression ratios compared to other popular compression methods, but it is easy to implement and is quick to execute. Run-length encoding is supported by most bitmap file formats such as TIFF, BMP and PCX.

Consider a character run of 15 ‘A’ characters which normally would require 15 bytes to store: AAAAAAAAAAAAAAA becomes 15A. With RLE, this would only require two bytes to store; the count (15) is stored as the first-byte and the symbol (A) as the second byte. Consider another example, with 16 character string : 000ppppppXXXXaaa. This string of characters can be compressed to form 3(0)6(p)4(X)3(a). Hence, the 16 byte string would only require 8 bytes of data to represent the string. In this case, RLE yields a compression ratio of 2:1 [12]

For example, consider a string of text $S = \text{abaaabbbaaaabaaabbb}$. Running it through a RLE assuming run length of one is not allowed. The string will be compressed to $\text{ab3a3b3ab3a3b}$. There is thus a compression from 18 bytes 13 bytes.

Unlike the RLE, LZW (Lempel-Ziv-Welch) method maps strings of text characters into numeric code. To begin with, all characters that may occur in the string are assigned a code. For example, suppose that the string $S = \text{abaaabbbaaaabaaabbb}$ used above in the RLE example is to be compressed via LZW. Firstly, all occurring characters in string are assigned a code. In this case the occurring characters are “a” and “b” and are assigned 0 and 1 respectively. The mapping between character and their codes are stored in the dictionary. Each dictionary entry has two fields: key and code. The characters are stored in the key field while the numbers are stored in the code field. The LZW compressor repeatedly finds the longest prefix, $p$, of the unencoded part of $S$ that is in the dictionary and outputs its code. If there is a next character $c$ in $S$, then $pc$ (pc is the prefix string $p$ followed by the character $c$) is assigned the next code and inserted into the dictionary. This strategy is called the LZW rule [8].

Let $S = pc$ be an uncompressed string where p is the longest prefix of the uncompressed string found in dictionary and c is the suffix of the uncompressed string. To compress the string using LZW algorithm, the following steps are followed:

**Step 1:** Replace the p with q code where the dictionary key for q code must already be defined in the dictionary.

**Step 2:** Add new code inside the dictionary with key text $(q) \text{ fc}(p)$, where fc(p) represents the first character of the uncompressed string [8, 13].
To decompress using the algorithm, each code is fed as input one at a time and is replaced by the texts it represents. The code-to-text mapping can be reconstructed in the following ways: First, a predefined dictionary of all single occurring characters is setup. As before, the dictionary entries are code-text pairs. This time, however, the dictionary is searched for an entry with a given code (rather than with a given text). The first code in the compressed file corresponds to a single character and so may be replaced by the corresponding character. For all other codes p in the compressed file, two cases can be considered namely, the case in which code p is in the dictionary, and the case in which the code p is not in the dictionary. To decompress the compressed string the qp code is considered.

*Case 1:* Code p is found in dictionary. Replace p with the text (p) and add new code into dictionary based on key text(q)fc(text(q)), where p represents the current code, fc the first character and q the previous code.

*Case 2:* Code p is not found in dictionary. Replaced p with text(q)fc(text(q)) and add new code into dictionary based on key text(q)fc(text(q))[8, 13].

### III. COMBINED ALGORITHM (RLE+LZW)

Though, LZW is a good compressor of text, yet its performance can still be enhanced by passing the string to be compressed through a RLE Compressor first before passing it through a LZW compressor. This results in a better compression gain than using LZW compressor alone. The RLE+LZW Compression algorithm consists of two steps:

*Step 1:* Compress the source file using RLE first.

*Step 2:* Compress the output of the RLE compressor using LZW compressor.

Similarly, the RLE+LZW decompression algorithm consists of two steps:

*Step 1:* Decompress the source file using LZW first.

*Step 2:* Decompress the output of the LZW decompressor using the RLE decompressor.

As an example, the string S = abaaabbaaabaaabb which was earlier considered will be further used to demonstrate the little, yet significant, compression gain of the RLE + LZW combined compressor. To start with, assume the string is first passed through the RLE compressor, the output of the RLE compressor will be ‘ab3a3b3ab3a3b’. If this output is now passed through the LZW compressor, the following stages will occur until the whole string has been compressed. First, the predefined dictionary is setup with all the occurrence of characters in the string to be compressed. Table 3.1 below shows the initial configuration of the dictionary before compression begins and its various stages. To begin compression, set up the initial predefined dictionary as done in stage (1) of Table 3.1 below.

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = null
Initial configuration (Predefined dictionary)

Stage 2

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>ab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 0
a has been compressed, ab added to dictionary

Stage 3

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>ab</th>
<th>b3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 01
ab has been compressed, b3 added to dictionary

Stage 4

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>ab</th>
<th>b3</th>
<th>3a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 012
ab3 has been compressed, 3a added to dictionary

Stage 5

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>ab</th>
<th>b3</th>
<th>3a</th>
<th>a3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 0120
ab3a has been compressed, 3b added to dictionary

Stage 6

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>ab</th>
<th>b3</th>
<th>3a</th>
<th>a3</th>
<th>3b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 01202
ab3a3 has been compressed, 3b added to dictionary

Stage 7

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 012025
ab3a3b3 has been compressed, b3a added to dictionary

Stage 8

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 0120254
ab3a3b3ab has been compressed, ab3 added to dictionary

Stage 9

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original string: ab3a3b3ab3a3b
Output code = 01202546
ab3a3b3ab3a has been compressed, 3a3 added to dictionary
Stage 10

<table>
<thead>
<tr>
<th>Key</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>a</th>
<th>3</th>
<th>b</th>
<th>3</th>
<th>3</th>
<th>a</th>
<th>b</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Final Dictionary)

Original string: ab3a3b3ab3a3b
Output code = 012025468
ab3a3b3ab3a3b has been compressed, 3a3 added to dictionary

Stage 11

Original string: ab3a3b3ab3a3b
Output code = 012025468
ab3a3b3ab3a3b has been compressed, nothing added to dictionary

The final compressed string of the RLE output through the LZW compressor is 012025468, which occupies 9 bytes.

Initially, there was a 18 byte string passed through the RLE compressor. However, the string was reduced to 13 bytes by the RLE compressor. The output of the RLE compressor is now passed through the LZW compressor and there is a further reduction from its 13 bytes to 9 bytes. Thus, the combined RLE and LZW reduces the original string from 18 bytes to 9 bytes – a 50% reduction. When the same sample string was manually run through the LZW compressor, it compresses it to 13 bytes. Thus, the combined RLE+LZW algorithm outperforms LZW compressor.

IV. PERFORMANCE EVALUATION

In order to verify and justify the manual walkthrough of the RLE + LZW combined algorithm done in Table 3.1 above, some sets of files were run through the implemented version of RLE, LZW and RLW (combined algorithm) to compare their result and performance. The implementation was carried out using C language due to the language’s support for manipulating strings, among other features. Table 4.1 below shows the comparative result of the compression. While LZW and RLW compete favourably, it was observed that when it comes to text files, RLW compresses a little better than LZW. Though the compression gain is very little, yet it is significant. It is also noteworthy that in some types of files that are mostly non-text-based, LZW seems to equally have a little performance gain than RLW. There are however some types of text-based files where LZW still has a performance gain over RLW. Such files are either small in size with little redundancies, or bigger in size and yet with little redundancies. In Table 4.1 below, the files D1, D2, D3, D4 and D8 are text based files while the remaining files (i.e. D5, D6 and D7), though containing text, are not text-based files.
Table 4.1: Comparison of Algorithms Performance for Different File Sizes and Format.

<table>
<thead>
<tr>
<th>Serial No</th>
<th>Name</th>
<th>Origin File Sizes (KB)</th>
<th>RL Size (KB)</th>
<th>LZW Size (KB)</th>
<th>RLW Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>19.5</td>
<td>10.6</td>
<td>2.27</td>
<td>2.03</td>
</tr>
<tr>
<td>2</td>
<td>D2</td>
<td>147</td>
<td>116</td>
<td>49.2</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>D3</td>
<td>73</td>
<td>50.2</td>
<td>25.2</td>
<td>24.9</td>
</tr>
<tr>
<td>4</td>
<td>D4</td>
<td>32.5</td>
<td>14.6</td>
<td>8.5</td>
<td>8.34</td>
</tr>
<tr>
<td>5</td>
<td>D5</td>
<td>227</td>
<td>213</td>
<td>102</td>
<td>103</td>
</tr>
<tr>
<td>6</td>
<td>D6</td>
<td>444</td>
<td>191</td>
<td>64.7</td>
<td>64.9</td>
</tr>
<tr>
<td>7</td>
<td>D7</td>
<td>22.7</td>
<td>22.5</td>
<td>10.1</td>
<td>10.2</td>
</tr>
<tr>
<td>8</td>
<td>D8</td>
<td>41.5</td>
<td>25.2</td>
<td>12.8</td>
<td>12.6</td>
</tr>
</tbody>
</table>

In file D4, for instance, whereas the original file size is 32.5KB, use of RLE compression algorithm reduces the size to 14.6KB, while LZW, on the other hand, reduces it to 8.5KB. By using the combined algorithm (RLW), the size is reduced to 8.34KB which gives the best compression result.

IV. DISCUSSION AND CONCLUSION

From the analysis carried out in previous sections, it can be observed that the combined RLE and LZW compressor will bring about a little but significant compression than RLE or LZW alone for most text files. Since both RLE and LZW algorithms are exploiting common redundancy in text-based file (i.e. repetitiveness or multiple occurrences of phrases), combining this positive feature of the two algorithms into a single algorithm can bring about an increased performance than the individual algorithm. This increased compression gain is more noticeable in files with multiple occurrences of phrases and spaces as characterized in most day-to-day text based files. Thus, the greater the redundancies, the greater the degree of compression achieved. However, there are cases when the LZW outperforms the new algorithm; this is not always a common encounter and is observed with either smaller files with little repetitiveness or larger files with little or no repetitiveness. On the hand, since the individual algorithms are fast in execution speed, it can be argued that the speed of the combined algorithm will also be fast. It is important that the order of compression be strictly followed i.e. RLE first, then followed by LZW. This is due to the fact that LZW, from experimental observation, has been found to be an optimal compressor such that no other compression algorithm can compress its output further. The study therefore demonstrates that text-based files with lots of redundancies can be compressed to a higher degree of compression by first compressing it through the RLE algorithm before compressing it through the LZW algorithm.

POSTSCRIPT

This paper is part of the M.Sc (Computer Science) project [1] of the first author written under the supervision of the second author.

REFERENCES

Adeniyi Moronfolu is a product of the University of Ibadan, Nigeria where he received a Master of Science (M.Sc) degree in Computer Science.

‘Dele Oluwade studied both Building Technology and Mathematics at the University of Ife / Obafemi Awolowo University (OAU), Ile-Ife, where he received a Bachelor of Science (B.Sc) degree in Mathematics. He also received a Postgraduate Diploma (PGD) in Computer Science from the University of Lagos, a Master of Science (M.Sc) degree in Mathematics from the O.A.U., Ile-Ife, as well as a Master of Philosophy (M.Phil) degree and a Doctor of Philosophy (Ph.D) degree in Computer Science from the University of Ibadan. He is a registered Computer Scientist and a Senior Member of the IEEE. He is a computer/IT consultant.
FACE IMAGE PROCESSING, ANALYSIS AND RECOGNITION ALGORITHMS FOR ENHANCED OPTIMAL FACE RECOGNITION SYSTEMS DESIGN: A COMPARATIVE STUDY

Vincent Andrew Akpan*, Member, IEEE
Department of Electrical & Computer Engineering,
Aristotle University of Thessaloniki,
GR-54124 Thessaloniki,
Greece.

*Email (Corresponding author): avincent@ieee.org

&

Reginald R. A. O. Osakwe
Department of Physics,
The Federal University of Petroleum Resources,
P.M.B. 1221, Effurun, Delta State,
Nigeria.
Email: reginaldosakwe2003@yahoo.co.uk

Received 31 October, 2009

ABSTRACT

In this paper, the variational properties of face images are investigated with the objective of proposing suitable algorithms and technique for enhanced optimal face recognition system design. Ten different face image samples each of forty-eight different individuals taken under different light intensities were processed. The resulting four hundred and eighty face image samples were split into a training set which constitutes the database of known face images and a test set which constitutes unknown faces. The first face images of each of the forty-eight individuals were evaluated and analyzed using thirteen different mathematical algorithms available from MATLAB® image processing toolbox. The results of the analyses show significant differences for all the face image samples considered. Based on the results from the face image analyses; two algorithms: 1) the principal component analysis (PCA) and 2) the eigenfaces were proposed. The two algorithms were applied simultaneously for enhanced optimal face recognition, where a particular face must satisfy the two algorithms for recognition. The simulation results show that the proposed face image evaluation techniques as well as the proposed PCA and the eigenfaces algorithms recognizes a known face image or rejects an unknown face based on the database contents to a high degree of accuracy. The combination of the proposed face recognition strategy can be adapted for the design of on-line real-time embedded face recognition systems for public, private, business, commercial or industrial applications.

Keywords: Eigenfaces, face image evaluation, principal component analysis, variational properties.
I. INTRODUCTION

Faces and fingerprints recognitions have become an interesting field of research in recent times due to the increased identity fraud in our society reaching unprecedented proportions. To this end much emphasis and research have been placed on emerging algorithms and automated technologies for face and fingerprint identification and recognition. Ideally a face or fingerprint detection system should be able to take a new face or fingerprint and return a name identifying that person if the person exists in the database, or return an error of non-existence if there is no correspondence of such face or fingerprint. Statistically, faces and fingerprints can also be very similar. Using standard image sizes and the same initial conditions, a system can be built that looks at the statistical relationship of individual pixels; so that from signal processing point of view, the face and fingerprint recognition problem boils down essentially to the identification of an individual based on an array of pixel intensities and in addition to whatever information can be gleaned from the image to assign a name to the unknown set of pixel intensities. Characterizing the dependencies between pixel values becomes statistically a digital signal processing problem.

The theoretical foundations of digital signal processing were laid by Jean Baptiste Joseph Fourier [1]. Major theoretical developments in digital signal processing theory were made in the 1930s and 1940s by Nyquist and Shannon, among others (in the context of digital communication) and by the developers of the Z-transform (notably by Zadeh and Ragazzini in the West, and Tsypkin in the East) [2], [3]. The history of applied digital signal processing (at least in the electrical engineering world) began around the mid-1960s with the invention of the fast Fourier transform (FFT) [2], [4]. However, its rapid development started with the advent of microprocessors in the 1970s [5], [6]. In the present discussion, for space limitation and simplicity, we shall limit our investigations to face recognition since it might be possible to apply the proposed algorithms and techniques to fingerprints.

Several approaches to the overall face identification and recognition [7], [8], [9] problems have been devised over the years with considerable attention paid to methods by which effective face identification and recognition can be replicated. The intuitive way to do face recognition is to look at the major features of the face and compare them to the same features on other faces. Extensive research in face statistical data compression technique using principal component analysis (PCA) technique [10], face features comparisons and recognition based on the eigenfaces technique [11], and the direct face classification scheme using discrete cosine transform (DCT) coefficients [12], [13] have been reported. In [14], a comparative study of face recognition algorithms has been conducted and several issues such as: 1) preprocessing and cropping of the faces to equal and consistent sizes, database creation for the preprocessed images, and 2) a major problem in the amount of light intensity on the original face images have been reported. The comparison of the eigenfaces algorithm with the fisherfaces algorithm [15], [16] have also been reported; where it has been recommended that both algorithms can be used for accurate face recognition results except with higher computational load in the later which might limit its application in real-time systems. In addition to the above issues, the face data compression techniques has provided additional challenges to the face identification and recognition problem such as: 1) generating worse face images which are different from the original image [10], and 2) the difficulties associated with the restoration
of the original face image leading to the inclusion of color information. This second problem has led to the new fractal image encoding research fields [17], [18], [19].

In general, the face identification and recognition problem becomes more complicated and computationally intensive as the number of the face images increases especially when used in real-time [20]. As reported in [11], a 90% success in face recognition is achieved using the eigenfaces algorithm and this algorithm has been widely used in research papers [20], [21], [22] but with greater light intensity on well-taken face images.

In the present study, we seek to investigate the variational properties of face images for enhanced optimal identification and recognition by evaluating the statistical data derived from a well-reduced form of each of the original face images under different angular positions and light intensities. Based on the results of the evaluation we propose two algorithms based on the: 1) PCA and 2) eigenfaces algorithms.

The rest of the paper is organized as follows: In Section II we present the material and methodology employed in this study. The evaluation algorithm, its analysis and results are also presented in this section. The proposed PCA and the eigenfaces algorithms are formulated in Section III. The applications of these two algorithms for enhanced face recognition together with their results are given in Section IV. A brief conclusion and recommendation are presented in Section V.

II. FACE IMAGE PROCESSING, EVALUATION AND ANALYSIS

A. Materials and Methodology for Obtaining the Faces

The material employed in this study were four hundred and eighty (480) joint photographic expert group (JPG) face image samples, shown in Fig. 1) taken with a Samsung® DIGIMAXS500/CYBER530 digital camera [23]. The 480 face image samples consist of 10 different face samples of 48 individuals (students) taken at The Federal University of Petroleum Resources, Effurun,
Delta State, Nigeria. Out of the 480 face image samples, the first 75% (360 face images) shown in Fig. 1 were used to form the database of known faces images while the remaining 25% (120 face images) were reserved for investigating the performance and validation of the proposed face recognition algorithms. Due to space limitation, only the first out of the 10 face image samples for each of the 48 individuals are shown here. The proposed evaluation algorithm in this study are written, compiled and analyzed using MATLAB® 2009a from The MathWorks Inc. [24] software running on an Inter® Core™ Duo CPU E6750@2.67GHz computer on a Windows™ XP platform.

B. Processing of the Face Images

The processing of images is necessary to enhance fast and accurate image identification and recognition. Moreover, when the numbers face images are relatively large, each face image must be pre-processed by reducing its size and converting the image to a format requiring less memory storage capacity. In other to reduce the memory consumption for storing the face images, all the JPG face images were converted and resized to portable graymap (PGM) images of height 114 pixels and width 114 pixels, filtered to remove noise using an adaptive Wiener filter (AWF), and finally resized from their original height 640 pixels and width 480 pixels to 114 by 114 pixels respectively.

The PGM filtered and resized face images were then used for analysis in other to identify and recognized any face image within the database. The pre-processed face images (PPFI) used for analysis are the shown in Fig. 2 (note: rather than presenting all the 480 face image samples used in this work, only the first out of the ten face images for each of the 48 individual is shown in Fig. 2 for space economy). The pre-processing algorithm for

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE FACE IMAGE EVALUATION ALGORITHMS USING MATLAB®</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>% 1. Load the known face image from database (Only once for faster comparison)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>% 2. Incoming face image (INFI) for recognition.</td>
</tr>
<tr>
<td>In this study, this image is randomly from the 480 original face images (OFI) in terms of their number sequence between 1 and 480.</td>
</tr>
<tr>
<td>src_face = input('Please enter any number between 1 and 480 to recognize image = ');</td>
</tr>
<tr>
<td>% 3. Evaluate the input face image identity number to check if it is within the database limit. Otherwise prompt the user to enter a valid identity number.</td>
</tr>
<tr>
<td>% 4. Convert the JPG face images to PGM to reduce storage memory requirements and display result.</td>
</tr>
<tr>
<td>In this study, this image is randomly from the 480 original face images (OFI) in terms of their number sequence between 1 and 480.</td>
</tr>
<tr>
<td>src_face = input('Please enter any number between 1 and 480 to recognize image = ');</td>
</tr>
<tr>
<td>i_face = imread(['I_Fac.pgm']);</td>
</tr>
<tr>
<td>figure(1), imshow(i_face); title('OFI ##');</td>
</tr>
<tr>
<td>% 5. Filtering and resizing the PGM face images.</td>
</tr>
<tr>
<td>AWF_Face = wiener2(i_face,[3 3],0.8);</td>
</tr>
<tr>
<td>rf_Face = imresize(AWF_Face,([114 114]);</td>
</tr>
<tr>
<td>% 6. Reshape the filtered image matrices to comply with MATLAB’s toolboxes for analysis.</td>
</tr>
<tr>
<td>re_Face = reshape(rf_Face,DIM*LEN/nn,nn);</td>
</tr>
<tr>
<td>LEN_re = length(re_Face);</td>
</tr>
<tr>
<td>new_reFace = double(re_Face);</td>
</tr>
<tr>
<td>% 7. Analysis of the PPFI for each face image.</td>
</tr>
<tr>
<td>M_Face = mean(new_reFace);</td>
</tr>
<tr>
<td>S_Fcae = sum(new_reFace);</td>
</tr>
<tr>
<td>DCT_Face = dct2(new_reFace);</td>
</tr>
<tr>
<td>DCT_Face = dct2(new_reFace);</td>
</tr>
<tr>
<td>VAR_Face = sqrt(sum(new_reFace.^2)/LEN_re);</td>
</tr>
<tr>
<td>FFT_Face = fftshift(new_reFace,1);</td>
</tr>
<tr>
<td>Q_Face = detrend(new_reFace,'linear','true');</td>
</tr>
<tr>
<td>drf_Face = double(rf_Face);</td>
</tr>
<tr>
<td>[VEC, VAL] = eig(drf_Face);</td>
</tr>
<tr>
<td>CVM_Face = cov(new_reFace);</td>
</tr>
<tr>
<td>VCV_M_Face = diag(cov_face);</td>
</tr>
</tbody>
</table>

the face images are given in number one through number six in the algorithm of TABLE I. In the following subsection we present the evaluation algorithms used in the analysis of each face image.
Due to the fact that our ambition in the present investigation is high, we wish to stress here that the face images were not cropped so as to focus only on the foreface as can be seen in Fig. 1. This is to allow for the investigation and verification of the efficiency of the face identification and recognition strategies proposed in the present study; where a particular face sample can be taken amidst a crowd, criminal scenes or abnormal scenarios either with full light intensity and vice versa.

1) Evaluation of the Face Images using MATLAB®

Each of the face images were analyzed using thirteen different analysis algorithms available from MATLAB® in other to deduce the distinguishing and variational properties of each face image sample based on the results of their analysis. The thirteen variational properties investigated are summarized by the MATLAB® algorithms listed in number seven of TABLE 1 namely: standard deviation (STD_Face), mean (M_Face), summation (S_Face), singular value decomposition (SVD_Face), discrete Fourier transform (DCT_Face), basic variance (VAR_Face), fast Fourier transform (FFT_Face), compute the residual by subtracting the column mean from each column of the face image matrix (Q_Face), eigenvectors (VEC) and eigenvalues (VAL), covariance matrix (CVM_Face), variance of the covariance matrix (VCVM_Face), and standard deviation of covariance matrix (SDCVM_Face).

<table>
<thead>
<tr>
<th>Table II</th>
<th>PPI 1</th>
<th>PPI 2</th>
<th>PPI 3</th>
<th>PPI 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD_anal</td>
<td>mean(STD_Face);</td>
<td>For Fig. 3(a)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_anal</td>
<td>mean(M_Face);</td>
<td>For Fig. 3(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_anal</td>
<td>mean(S_Face);</td>
<td>For Fig. 3(c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD_anal</td>
<td>mean(SVD_Face);</td>
<td>For Fig. 3(d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCT_anal</td>
<td>sum(mean(DCT_Face));</td>
<td>For Fig. 3(e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR_anal</td>
<td>mean(S_Face);</td>
<td>For Fig. 3(f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT_anal</td>
<td>mean(mean(FFT_Face));</td>
<td>For Fig. 3(g)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q_anal</td>
<td>sum(diag(Q_face));</td>
<td>For Fig. 4(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVEC_anal</td>
<td>sum(diag(Vec_face));</td>
<td>For Fig. 4(c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAL_anal</td>
<td>sum(mean(Val_face));</td>
<td>For Fig. 4(d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVM_anal</td>
<td>mean(mean(CVM_Face));</td>
<td>For Fig. 4(e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCVM_anal</td>
<td>mean(VCVM_Face);</td>
<td>For Fig. 4(f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDCVM_anal</td>
<td>mean(SDCVM_Face);</td>
<td>For Fig. 4(g)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. The pre-processed and AWF face images (PPI) samples.
matrix (VCVM_Face), and STD of the covariance matrix (SDCVM_Face). These eleven parameters are selected because they form an integral part of the PCA, eigenfaces and eigenfisher algorithms for face recognition, image/signal processing and image data compression techniques.

C. Analysis of the Face Images

The analysis considered in the present study for evaluating all the thirteen parameters to investigate the variational properties of the face images are summarized in TABLE 2. The analyses were performed on the face images shown in Fig. 2. The results of these analyses are presented in Fig. 3 (a) – (f) and Fig. 4 (a) –
(g). The graphs of Fig. 3 and Fig. 4 show the variations in the properties for each of the face image samples evaluated in Section II-C. The DCT_Face, Q_Face, and VEC results of Fig. 3 (e), Fig. 4 (b) and Fig. 4(c) respectively show well distributed variations of the face images. Also the CVM, VCVM_Face and SDCVM of Fig. 4 (e), (f) and (g) respectively clearly distinguish each face image.

III. THE FACE IMAGE EVALUATION AND RECOGNITION ALGORITHMS

A. The PCA Algorithm Based on the SVD

Principal component analysis (PCA) is an orthogonal linear transformation tool that transforms data to a new coordinate system such that the greatest variance by any first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on [11], [25], [26]. PCA is theoretically the optimum transform for given data in least square terms and it involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible [25]. The PCA is the simplest of the true eigenvector-based multivariate analyses. Often, the PCA operation can be thought of as revealing the internal structure of the data in a way which best explains the variance in the data. Furthermore, PCA provides a strategy on how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structure that often underlie it.

Two methods, with their merits and demerits, have been proposed in [25] for the evaluation of PCA using the: 1) covariance method and 2) correspondence analysis. Based on the results of the face evaluation recognition of TABLE I of Section II-C presented in Section II-D, we proposed a SVD algorithm which will utilize the properties of the covariance matrix to achieve faster and more concise computation of the principal components of the face images. In the following we present the SVD algorithm to facilitate the solution of the PCA algorithm. The mathematical intuition of the SVD is detailed in [26] and [27]. While the SVD formulated in [26] is directly related to PCA; the SVD formulation in [27] avoids PCA directly. The approach presented here combines ideas from [26] and [27].

1) The formulation of the PCA Algorithm

Given an arbitrary \( m \times n \) matrix \( \Phi \), the objective of the PCA is to find some orthonormal matrix \( P \) by constructing another matrix \( \Theta = P\Phi \) such that \( C_e = \frac{1}{n-1}\Theta\Theta' \) is diagonalized. The rows of \( P \) are the principal components of \( \Phi \); where \( m \) is the number of measurement types, \( n \) is the number of samples (measurement trials), and is the \( C_e \) covariance matrix (an explanation of \( C_e \) is provided in Corollary 1 below). Thus, by rewriting \( C_e \) as

\[
C_e = \frac{1}{n-1}\Theta\Theta' = \frac{1}{n-1}(P\Phi)(P\Phi)' = \frac{1}{n-1}P\Phi\Phi'P' = \frac{1}{n-1}P\Phi\Phi'P' = \frac{1}{n-1}PAP'
\]

where \( A = \Phi\Phi' \) and by APPENDIX II, \( A \) is a symmetric matrix.

Then, by the theorems of APPENDICES III AND IV, we recognize that the symmetric matrix \( A \) is diagonalized by an orthonormal matrix of its eigenvectors. According to the theorem of APPENDIX IV, we can express \( A \)
as:
\[ A = E D E^T \]  \hspace{1cm} (2)

where \( D \) is a diagonal matrix and \( E \) is a matrix of eigenvectors of \( A \) arranged as columns. The matrix \( A \) has \( r \leq m \) orthonormal eigenvectors where \( r \) is the rank of the matrix \( A \). The rank of \( A \) is less than \( m \) when \( A \) is degenerate or all data occupy a sub-space of dimension \( r \leq m \).

The issue of degeneracy is handled here to maintain the orthogonality, we select an \((m-r)\) additional orthonormal vectors to “fill up” the matrix \( E \). We note that these additional vectors do not effect the final solution because the variances (see Corollary 1) associated with these directions are zero.

To obtain the principal components (PCs) of \( \Phi \), we select the matrix \( P \) in (1) as a matrix where each \( i \)th is an eigenvector of \( T \Phi \Phi = \Phi \Phi \).

By selecting and substituting \( T \Phi \Phi \) into (2), we obtain:
\[ T \Phi \Phi P DP = \] \hspace{1cm} (3)

Using (3) and the theorem of APPENDIX I (i.e. \( P^T = P \)), we recomputed (1) as follows:
\[ C_e = \frac{1}{n-1} P \Phi P^T = \]
\[ = \frac{1}{n-1} P (P^T DP)P^T = \frac{1}{n-1} (PP^T) D (PP^T) \]
\[ = \frac{1}{n-1} (PP^{-1}) D (PP^{-1}) = \frac{1}{n-1} D \] \hspace{1cm} (4)

As it is evident in (4), the choice of \( P \) diagonalizes \( C_e \) as we desired. Finally, we summarize the PCA procedure as follows: 1) the PCs of \( \Phi \) are the eigenvectors of \( \Phi \Phi \) or the rows \( p_i \) of \( P \); and 2) the \( i^{th} \) diagonal of \( C_e \) is the variance of \( \Phi \) along \( p_i \).

Thus, computing the PCs of \( \Phi \), reduces to 1) subtracting off the mean of each measurement type; and 2) computing the eigenvectors of \( \Phi \Phi \).

2) The formulation of the SVD Algorithm

Here, let \( \Phi \) be an arbitrary \( n \times m \) matrix and \( \Phi^T \Phi \) be a rank \( r \), square and symmetric \( m \times m \) matrix. Also let \( \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_r\} \) be the set of orthonormal \( m \times 1 \) eigenvectors (see APPENDIX IV second part for proof of orthonormality) with associated eigenvalues \( \{\lambda_1, \lambda_2, \ldots, \lambda_r\} \) for the symmetric matrix \( \Phi^T \Phi \) such that \( (\Phi^T \Phi) \hat{a}_i = \lambda_i \hat{a}_i \). \( \gamma_i = \sqrt{\lambda_i} \) are positive real and termed the singular values; and \( \{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n\} \) be the set of \( n \times 1 \) vectors defined by:
\[ \hat{b}_i = (1/\gamma_i) \Phi \hat{a}_i \] \hspace{1cm} (5)

where \( i \) and \( j \) are the length of \( m \). Before stating the last two additional properties required for deriving the SVD algorithm, it is necessary to first consider the Theorem below for the orthonormality and orthogonality properties of \( \Phi \).

**Theorem 1:**

For any arbitrary \( m \times n \) matrix \( \Phi \), the symmetric matrix \( \Phi^T \Phi \) has a set of orthonormal eigenvectors of \( \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_r\} \) and a set of associated eigenvalues \( \{\lambda_1, \lambda_2, \ldots, \lambda_r\} \) (see APPENDIX IV). The set of vectors \( \{\Phi \hat{a}_1, \Phi \hat{a}_2, \ldots, \Phi \hat{a}_r\} \) then form an orthogonal basis (see APPENDIX I for proof), where each vector \( \Phi \hat{a}_i \) is of length \( \sqrt{\lambda_i} \). Based on the dot product of any two vectors [26], we have the following:
\[ (\Phi \hat{a}_j) \cdot (\Phi \hat{a}_j) = (\Phi \hat{a}_j)^T (\Phi \hat{a}_j) = \gamma_j \Phi^T \Phi \hat{a}_j \]
\[ = \gamma_j (\lambda_j \hat{a}_j) = \lambda_j \hat{a}_j \cdot \hat{a}_j \]

So that \( (\Phi \hat{a}_j) \cdot (\Phi \hat{a}_j) = \lambda_j \delta_j \) \hspace{1cm} (6)

where \( j \) is the length of \( n \).

Equation (6) arises due to the fact that the set of eigenvectors of \( \Phi \) is orthogonal resulting in the Kronecker delta \( \delta_j \). Alternatively, (6) can be expressed from [26] as:
(\Phi \hat{\alpha}_i) \cdot (\Phi \hat{\alpha}_j) = \begin{cases} \lambda_j & i = j \\ 0 & i \neq j \end{cases} \quad (7)

which states that any two vectors in the set are orthogonal. The second property arises from (6) by realizing that the length squared of each vector can be defined as [26]:

\|\Phi \hat{\alpha}\|^2 = (\Phi \hat{\alpha}) \cdot (\Phi \hat{\alpha}) = \lambda_i \quad (8)

With Theorem 1, we can define the following two properties (see also APPENDIX I for proof of orthogonality):

\hat{\alpha}_i \cdot \hat{\alpha}_j = \begin{cases} 1 & if \ i = j \\ 0 & otherwise \end{cases} \quad (9)

\|\Phi \hat{\alpha}\| = \xi_i \quad (10)

Finally, we define the “value” version of SVD as a restatement of (5) which can be expressed as:

\Phi \hat{\alpha}_i = \xi_i \hat{\beta}_i \quad (11)

Equation (11) implies that \Phi multiplied by an eigenvector \hat{\alpha}_i of \Phi'\Phi is equal to a positive scalar \xi_i times another vector \hat{\beta}_i. The set of eigenvectors \{\hat{\alpha}_1, \hat{\alpha}_2, ..., \hat{\alpha}_r\} and the set of vectors \{\hat{\beta}_1, \hat{\beta}_2, ..., \hat{\beta}_r\} can both be orthonormal sets or bases in r-dimensional space.

To obtain the matrix form of the SVD, we begin by constructing the following three new matrices: \Psi, A and B as:

\[ \Psi = \begin{bmatrix} \xi_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & \xi_r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (12) \]

\[ \Lambda = [\hat{\alpha}_1 \hat{\alpha}_2 ... \hat{\alpha}_n] \]

\[ B = [\hat{\beta}_1 \hat{\beta}_2 ... \hat{\beta}_n] \quad (13) \]

where \Psi is a n \times m diagonal matrix with a few non-zero values \xi_1, \xi_2, ..., \xi_r such that \xi_1 \geq \xi_2 \geq ... \geq \xi_r are the rank-ordered set of singular values; and matrices A and B are

m \times m and n \times n respectively and the corresponding vectors are indexed in the same rank order. To handle the issue of degeneracy in A and B, we append an additional (m-r) and (n-r) orthonormal vectors to “fill up” the matrices for A and B respectively.

The mathematical intuition behind the construction of the matrix forms of (12) and (13) is to express all the n “value” equations of (11) into a single invertible form; so that the matrix multiplication for all the vectors can be accomplished in a single n iteration.

Each pair of associated vectors \hat{\alpha}_i and \hat{\beta}_i in (13) are stacked in the \text{i}^{th} columns along their respective matrices. The corresponding singular value \xi_i is placed along the diagonal (i.e. the \text{i}^{th} position) of \Psi in (12). So that (11) degenerates into the following form expressible as:

\[ \Phi \Lambda = B \Psi \quad (14) \]

where each column of A and B in (13) performs the “value” version of the decomposition using (11) subject to (10). Since A is orthogonal, multiplying both sides of (10) by \Lambda^{-1} = \Lambda^T leads to the final form of the decomposition which can be expressed as:

\[ \Phi = B \Psi A^T \quad (15) \]

From (11), the SVD technique implies that: 1) any arbitrary matrix \Phi can be converted to an orthogonal matrix B, a diagonal matrix \Psi and another orthogonal matrix A; 2) the columns of matrix A contain the eigenvectors of \Phi^T \Phi.

In the following discussion, we consider the relationship between PCA and SVD; and then we present a summary of the procedure for computing the PCs of \Phi using SVD.

\[ 3) \text{The Principal Components of the SVD Algorithm} \]

Considering the original matrix m \times n data matrix \Phi such that we can define a new n \times m matrix \Theta given below as:
\[ \Theta \equiv \frac{1}{\sqrt{n-1}} \Phi^T \]

where each column of \( \Theta \) has zero mean. To obtain the PCA using the SVD described above based on (15), it is necessary to first express (16) as a symmetric matrix \( \Theta^T \Theta \) which we express below as:

\[ \Theta^T \Theta = \left( \frac{1}{\sqrt{n-1}} \Phi^T \right) \left( \frac{1}{\sqrt{n-1}} \Phi^T \right) = \frac{1}{n-1} \Phi^T \Phi^T \]

So that \( \Theta^T \Theta = \frac{1}{n-1} \Phi \Phi^T = \Gamma_x \)

where \( \Gamma_x \) (similar to \( C_e \) in (1) and (4)) is the covariance matrix of \( \Phi \). From (15) and (17), it can be seen that \( A = \Phi^T \), \( B = 1/(n-1) \) and \( \Psi = \Phi \). So that by computing the SVD of \( \Theta \), the columns of matrix \( A \), as in (13), will give the eigenvectors of (17) that are the PCs of \( \Phi \) presented following the discussion of the corollary below.

**Corollary 1:**

Consider two sets of measurements with zero means which can be expressed as [26]:

\[ A = [\alpha_1, \alpha_2, \ldots, \alpha_n] \quad \text{and} \quad B = [\beta_1, \beta_2, \ldots, \beta_n] \]

where the \( n \) denotes the numbers of trial measurements. The variance \( \varsigma_i^2 \) and \( \varsigma_n^2 \) of \( A \) and \( B \) can be defined separately as:

\[ \varsigma_i^2 = \frac{1}{n} \sum_{i=1}^{n} \alpha_i^2 \quad \text{and} \quad \varsigma_n^2 = \frac{1}{n} \sum_{i=1}^{n} \beta_i^2 \]

The covariance between \( A \) and \( B \) can be generalized as [26]:

\[ \varsigma_{ab}^2 = \frac{1}{n} \sum_{i=1}^{n} \alpha_i \beta_i \]

The covariance measures the degree of the linear relationship between the two variables \( A \) and \( B \). A large positive value indicates positively correlated data while a large negative value denotes negatively correlated data. The absolute magnitude of the covariance measures the degree of redundancy. Additionally properties of \( \varsigma_{ab}^2 \) are: 1) \( \varsigma_{ab} = 0 \), if and only if \( A \) and \( B \) are uncorrelated; and 2) \( \varsigma_{ab}^2 = \varsigma_i^2 \), if \( A = B \).

By converting \( A \) and \( B \) into corresponding row vectors as \( \phi_i = [\alpha_1, \alpha_2, \ldots, \alpha_n] \) and \( \phi_j = [\beta_1, \beta_2, \ldots, \beta_n] \) respectively; the covariance for the unbiased estimate can be expressed as the following dot product matrix computation:

\[ \varsigma_{_{ij}}^2 = \frac{1}{n-1} \phi_i \phi_j^T \]

We collect the terms \( \phi_i \) and \( \phi_j \) in (20) with additional indexed row vectors \( \phi_i, \phi_j, \ldots, \phi_m \) appended to form a new \( m \times n \) matrix \( \Phi \) expressed as:

\[ \Phi = [\phi_1, \phi_2, \ldots, \phi_m]^T \]

so that each row of \( \Phi \) corresponds to all \( i \)th measurements of a particular type while each column corresponds to a set of \( j \)th measurements from one particular trial condition. Thus, the covariance matrix \( \Gamma_{ij} \) can be defined as:

\[ (\Gamma_{ij})_{ij} = \frac{1}{n} \Phi_i \Phi_j^T \]

It should be note that the \( ij \)th element of (21) is the dot product between the vectors of the \( i \)th measurement type with the vector of the \( j \)th measurement type base on (20). We here some summarize the properties of \( \Gamma_{ij} \) in (21) as follows: 1) \( \Gamma_{ij} \) is a square symmetric \( m \times n \) matrix; 2) the diagonal terms of \( \Gamma_{ij} \) are the variance of particular measurement types; and 3) the off-diagonal terms of \( \Gamma_{ij} \) are the covariance between measurement types. Moreover, \( \Gamma_{ij} \) captures the correlations between all possible pairs of measurements where the correlation values reflect the noise and redundancy in the measurements data types.
As in (17), \( \Theta^T \Theta \) equals the covariance matrix of \( \Phi \) and the principal components of \( \Phi \) are the eigenvectors of \( \Gamma_x \). The computation of the SVD of \( \Theta \) results in the eigenvectors of \( \Theta^T \Theta = \Gamma \) contained in the columns of matrix \( A \). This implies that: 1) the columns of \( A \) are the principal components of \( \Phi \); 2) matrix \( A \) spans the row space of \( \Theta = (1/\sqrt{n-1}) \Phi^T \) and \( A \) must also span the column space of \( \Theta = (1/\sqrt{n-1}) \Phi^T \); 3) by symmetry the columns of \( B \) produced by the SVD of \( (1/\sqrt{n-1}) \Phi^T \) is also the principal components.

Finally, we summarize the SVD procedure for obtaining the PCs of \( \Phi \) as follows: 1) organize the \( m \times n \) data of matrix \( \Phi \), where \( m \) is the number of measurements types and \( n \) is the number of samples (measurement trial types); 2) subtract off the mean for each \( m \) measurement types; and 3) compute the SVD or the eigenvectors of the covariance matrix \( \Gamma_x \).

### B. The Formulation of the Eigenfaces Algorithm

Eigenfaces are basically basis vectors for real faces and it is directly related to the Fourier analysis which reveals that the sum of weighted sinusoids at different frequencies can be recomposed back to the original signal. This implies that the sum of weighted eigenvectors can seamlessly reconstruct a specific face image.

To formulate the eigenfaces algorithm, we convert each face image to a vector \( \Gamma_n \) of length \( N = \text{image width (114 pixels)} \times \text{image height (114 pixels)} \) which forms the measurement data. To increase the information available for the known face images in the database, we use 10 samples of each of the 48 individuals as described in Section II-A above. Here we denote the face samples as “face space” of dimension \( N \), so that the average face image in the face space can be computed as follows:

\[
\varphi = \frac{1}{M} \sum_{n=1}^{M} \Gamma_n
\]  

(22)

where \( M \) is the number of face images in the face space.

The covariance matrix discussed in Section III-B is denoted here for the eigenfaces algorithm as \( C_\varepsilon \) of dimension \( N \) (number of pixels in the face images) which is expressed here from [11] as:

\[
C_\varepsilon = \frac{1}{M} \sum_{\eta=1}^{M} \Phi_\eta \Phi_\eta^T
\]

(23)

where \( \var \) and \( \text{cov} \) are the variance and covariance of their respective arguments (see Corollary 1 for details on variance and covariance); \( \Phi_\eta = \Gamma_\eta - \varphi_\eta \) is the difference between each face image under consideration and the average of the face images in the database; \( A = [\Phi_1, \Phi_2, \ldots, \Phi_M] \); and \( p_\eta \) is the pixel of face \( \eta \). Thus the eigenfaces algorithm reduces to the evaluation of the eigenvalues of \( C_\varepsilon \), which can be computationally intensive due to the large value of \( N \).

By using the data transformation properties provided by the PCA just described above, the \( N \)-dimensional covariance matrix (23) can be reduced to an \( M \)-dimensional matrix corresponding to the number of face images in the database. By constructing a new \( M \times M \) matrix \( L \) as follows:

\[
L = A^T A
\]  

(24)

According to the PCA [26], since we have only \( M \) images; there will be only \( M \) non-trivial eigenvectors.
Theorem 2:

Suppose that we can define the right hand side of (24) as:

\[ A^T A v_i = \mu_i v_i \]

where \( \mu_i \) is scalar and \( v_i \) is an eigenvector of \( L \). By multiplying both sides of \( A^T A v_i = \mu_i v_i \) by \( A \), we obtain:

\[ A A^T A v_i = \mu_i A v_i \]

where \( A v_i \) are the eigenvalues of the covariance matrix \( A^T A \) (where \( A^T A = A A^T \) from theorem of APPENDIX II).

From Theorem 2, it is easy to see that \( A v_i \) are eigenvalues of \( C_e \) by (23). So that we can use the \( M \) eigenvectors of \( L \) in (24) to form the \( M \) eigenvectors \( u \) of \( C_e \) as follows:

\[ u = \sum_{i=1}^{M} v_i \Phi_i \]  

(25)

Thus, \( u \) forms the eigenfaces basis which is a linearly independent basis set for the face space. It is evident that only \( M - k \) eigenfaces are needed to produce the complete basis for the face space to achieve a descent reconstruction of the original face image using on few eigenfaces \( (M') \); where \( k \) is the number of unique individuals in the database of known face images, and \( M' \) corresponds to the vectors with the highest eigenvalues within the face space. These eigenfaces also represent the PCs of the face image samples.

Corollary 2:

Consider a set of vectors representing the weight \( (w_p) \) and height \( (h_p) \) components of an individual which can be expressed as the dot product of two vectors as follows [11]:

\[ w_p = \text{individual \cdot weight} \]
\[ h_p = \text{individual \cdot height} \]  

(26)

Projecting a given individual onto these vectors should yield the individual’s weight and height components in other to obtain the closest match between the individual and the set of individuals available in the database of the face images.

To accomplish the face recognition task, we begin by subtracting and projecting the mean of all the face images in the database onto the face space. According to Corollary 2, this corresponds to the dot product of each face image with one of the eigenfaces; which can be combined as matrices to obtain the weight matrix \( W \) as follows [11]:

\[ W = \begin{pmatrix} w_{1,1} & \cdots & w_{1,k} \\ \vdots & \ddots & \vdots \\ w_{M',1} & \cdots & w_{M',k} \end{pmatrix} \]  

(27)

where \( w_k = \mu_k (\Gamma_{new} - \Phi_k) \) and \( k = 1, 2, \ldots, M' \).

In this way, the face image of the same individual can be mapped fairly close to one another in the face space; so that the face recognition problem further reduces to finding the minimum Euclidean distance \( \varepsilon_i \) between a face image point \( \Omega_{new} \) and the database point \( \Omega_k \) which can be expressed as follows (by noting from (27) that \( \Omega = [w_1, w_2, \ldots, w_{M'}]^T \)):

\[ \varepsilon_i = \sqrt{\| \Omega_{new} - \Omega_k \|^2} \]  

(28)

The combination of the distribution in (28) and the PCA allows for dimensional reduction; where only the first several eigenfaces represent the majority information in the database and the computational complexities becomes significantly reduced.

As pointed out in [11], further verification on the decision to accept or reject the recognized face image can be achieved by considering the face space as an \( N \)-dimensional sphere encompassing all the
weight vectors in the entire database; so that an approximate radius of the space should be half the distance between the furthest points in the sphere according to the following expression derived from (28):

$$\theta_{\text{threshold}} = \frac{1}{2} \max (\varepsilon_i)$$ (29)

To justify whether a new face lies within this radius, it is necessary to compute the reconstruction error $\varepsilon$ between the face and its reconstruction using $M$ eigenfaces as follows:

$$\varepsilon_{\text{recon}} = \sqrt{\|\Phi_{\text{face}} - \Phi_{\text{recon}}\|^2}$$ (30)

where $\Phi_{\text{recon}} = \sum_{q=1}^{M} w_q \mu_q$ is the reconstructed face image. If the face image projects fairly well onto the face space and follows the face image distribution, then the error should be small, i.e. $\varepsilon_{\text{recon}} < \theta_{\text{threshold}}$ and can be compared with the $\theta_{\text{threshold}}$ of the face images in the database. However, a non face image will always lie outside the radius of the face space (i.e., $\varepsilon_{\text{recon}} > \theta_{\text{threshold}}$) and the algorithm should terminate as there is no point for comparing $\theta_{\text{threshold}}$ with the face images in the database.

Finally, the implementation of our eigenfaces algorithm based on an averaging technique where all weight vectors of a like individual within the database are averaged together. This creates a “face class” where an even smaller weight matrix represents the general faces of the entire system. When a new face image is introduced for recognition, the algorithm creates its weight vector by projecting the introduced face image (INFI) onto the face space. The INFI is then matched to the face class that minimizes the Euclidean distance based on the number of signature (num sig) required to recognize the INFI.

For the present study, our database has a total of three hundred and sixty face images, composed of thirty-six individuals with ten images each. The averaging technique thus yields a weight matrix with thirty-six vectors (thirty-six distinct face classes). Here we incorporate a face image counter ($\text{fin}_{\text{count}}$) which increments if the image matches correctly its own face class or does not change if the minimum distance matches to a face class of another person.

IV. IMPLEMENTATION OF THE PCA AND EIGENFACES ALGORITHMS FOR INFI RECOGNITION

The PCA and the face recognition algorithms in the present study is implemented in such a way that the user of the algorithm is prompted for the INFI identify as a numerical number between 1 and 480 which corresponds to the number of face images in the database. If any numerical value between 1 and 360 is supplied, the PCA and the face recognition algorithms are activated.

Initially, the principal components (PCs) of all the known 360 face images in the database were computed and stored in a database as image2_pca (see TABLE I). The PC of the INFI is then computed and compared with the 360 known face image PCs in the database. If the PC of the INFI (shown in the upper segment of Fig. 5 (a), (b), (c), (d), (e) and (f)) corresponds to any PC in the PC database; the corresponding face image and name of the INFI are displayed as shown in the middle segment of Fig. 5 (a), (b), (c), (d), (e) and (f).

However, if there is no corresponding PC for the INFI in the PC database, the comments shown in the middle part of Fig. 6 (a) and (b) are displayed (instead of Fig. 5) indicating non-existence of such face image in the database.

Similarly, the 360 face image sample were converted to $M$-dimensional matrices and stored in the database of known face images as image1_facedb (see TABLE II). The
Fig. 5. The results obtained when the incoming face image (INFI) with known identity is recognized by both the PCA and the eigenfaces algorithms with their respective names.

The eigenfaces algorithm is then evaluated for the INFI with \( \text{num}_{\text{sign}} = 30 \), projected and compared to the face space. If \( \epsilon_{\text{new}} < \theta_{\text{threshold}} \), the INFI is compare to the available face images in the database in terms of the projected matrices on
the face space; and the first $fim_{vmax} = 10$ highest eigenvalues that matches the INFI is selected to represent the INFI as shown in the lower segment of Fig. 5 (a), (b), (c), (d), (e) and (f). On the other hand if $e_{recon} > \theta_{\text{threshold}}$, it implies that the INFI is not available and there should be no need for comparison with face image samples in the database; and consequently the comments in the lower part of Fig. 6 (a) and (b) are displayed.

The comments shown in the middle and lower segments were obtained for INFI with numerical values ranging between 361 and 480 which correspond to the 120 test face images (i.e., the last 25% of the 480 face image samples). However, when numerical values outside the ranges 1 to 480 are used, the proposed algorithm responds with the comments shown in Fig. 7 and terminates automatically.

Because the PCA and the image processing algorithms are computationally intensive, the pre-computed values of the PCs and pre-processed face images (PPFIs) are loaded
only once; so that only the values of $PC$ and pre-processing of the $INFI$ is computed at all future instances. In this way, the comparison, identification and recognition process is enhanced in terms of speed; which suggest a possibility for real-time online implementation of our proposed technique.

V. CONCLUSION AND RECOMMENDATION

The variational properties of face images have been investigated based on numerical analysis. On the basis of the analysis, two algorithms: the principal component analysis and the eigenfaces algorithms have been proposed for efficient and enhanced optimal face recognition. These algorithms have been tested and validated with 360 known image face samples stored in a database and 120 unknown face image samples that were part of the database of known face images.

The performance of the proposed algorithms on 480 face image samples taken under varying angular positions and mostly under low light intensity conditions demonstrates the efficiency and adaptability of the proposed algorithms in critical and unprecedented situations. The implementation technique demonstrated in this work based on face image identity (i.e. the prompted numerical values) could be likened to bank account, identity card or passport numbers as a means of personal identification to checkmate crimes and fraudulent activities. However, a valid identity number only does not guarantee recognition if the face is not recognized.

The speed and accuracy of the proposed algorithms, based on the implementation strategy, show that it could be adapted for on-line face analysis and recognition in real-time. Further work could be on the parallel implementation of the proposed algorithms on real-time embedded reconfigurable computing machines such as field programmable gate arrays (FPGAs) or complex programmable logic devices (CPLDs) with real-time camera interfaced for on-line face analysis and recognition systems design.

APPENDIX I

Theorem: The inverse of an orthogonal matrix is its transpose

Proof:

Let $A = [a_1, a_2, \ldots, a_n]$ be an $m \times n$ orthogonal matrix, where $a_i$ is the $i^{th}$ column vector. The $ij^{th}$ element of $A^\top A$ is expressed as

$$(A^\top A)_{ij} = a_i^\top a_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Therefore, because $A^\top A = I$, where $I$ is an identity matrix; it follows that $A^{-1} = A^\top$.

APPENDIX II

Theorem: For any matrix $A$, $A^\top A$ and $AA^\top$ are symmetric.

Proof:

$$(AA^\top)^\top = A^\top (A^\top)^\top = AA^\top$$

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A$$

The equality of quantities in the above two expressions with their transpose completes this proof.

APPENDIX III

Theorem: A matrix is symmetric if and only if it is orthogonally diagonalizable.

Proof:

If $A$ is orthogonally diagonalizable, then $A$ is symmetric. By hypothesis, orthogonally diagonalizable implies that there exist some matrix $E$ such that $A = EDE^\top$, where $D$ is a diagonal matrix and $E$ is some special matrix which diagonalizes $A$. So that $A^\top$ can be computes as follows:

$$A^\top = (EDE^\top)^\top = E^\top D^\top E^\top = EDE^\top = A$$

It is evident that if $A$ is orthogonally diagonalizable, it must also be symmetric.
APPENDIX IV

Theorem: A symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors.

Proof:

Let $A$ be a square $n \times n$ matrix with associated eigenvectors $\{e_1, e_2, \ldots, e_n\}$. Let $E = [e_1, e_2, \ldots, e_n]$ where the $i^{th}$ column of $E$ is the eigenvector $e_i$. This theorem asserts that there exists a diagonal matrix $D$ where $A = EDE^T$.

This theorem is an extension of APPENDIX III. It provides a prescription of how to find the matrix $E$, the “diagonalizer” for a symmetric matrix. This theorem emphasizes that the diagonalizer is in fact a matrix of the original matrix’s eigenvectors. The proof is in two parts.

For the first part, let $A$ be some matrix, not necessarily symmetric, having independent eigenvectors (i.e. no degeneracy). Furthermore, let $E = [e_1, e_2, \ldots, e_n]$ be the matrix of eigenvectors placed in the columns. Let $D$ be a diagonal matrix where the $i^{th}$ eigenvalue is placed in the $i^{th}$ position. The proof is to show that $AE = ED$.

$$AE = [Ae_1, Ae_2, \ldots, Ae_n]$$
$$ED = [\lambda_1 e_1, \lambda_2 e_2, \ldots, \lambda_n e_n]$$

Evidently, if $AE = ED$, then $Ae_i = \lambda_i e_i, \forall i$.

This equation is the definition of the eigenvalue equation. Therefore, it must be that $AE = ED$. Using the result of APPENDIX III, it is easy to see that $A = (EDE)^{-1}$.

The second part, let $\lambda_1$ and $\lambda_2$ be distinct eigenvalues for eigenvectors $e_1$ and $e_2$; so that

$$\lambda_1 e_1 \cdot e_2 = (\lambda_1 e_1)^T e_2 = (Ae_1)^T e_2 = e_1^T A^T e_2$$
$$= e_2^T A e_1 = e_2^T (\lambda_1 e_1)$$

$\therefore \lambda_1 e_1 \cdot e_2 = \lambda_1 e_1 \cdot e_2$.

Equating the last equation gives $(\lambda_1 - \lambda_2) e_1 \cdot e_2 = 0$. Since we have conjectured that the eigenvalues are in fact unique, it must be the case that $e_1 \cdot e_2 = 0$. Therefore, the eigenvectors of a symmetric matrix are orthogonal.

Following our original postulate that $A$ is a symmetric matrix and by the second part of the proof, it can be seen that the eigenvectors of $A$ are all orthonormal (we choose the eigenvectors to be normalized). This means that $E$ is an orthogonal matrix, and by APPENDIX I, $E^T = E^{-1}$; so that we can express the final result as $A = (EDE)^{-T}$. Thus, a symmetric matrix is diagonalized by a matrix of its eigenvectors.

In the first part, we see that any matrix can be orthogonally diagonalized if and only if the matrix’s eigenvectors are linearly independent. In the second part, it has been seen that a symmetric matrix has the special property that all of its eigenvectors are not just linearly dependent but also orthogonal; and thus, completing the proof.

ACKNOWLEDGMENT

The authors wish to thank the students of The Federal University of Petroleum Resources, Effurun, Delta State, Nigeria, whose photographs form the database, for accepting to be part of this work.

REFERENCES


Vincent Andrew Akpan
(M’08) received his B.Sc. in Physics from The Delta State University (DELSU), Abraka, Nigeria in 1997 and his M.Tech. in Electronic Measurement & Instrumentation from The Federal University of Technology (FUTA), Akure, Nigeria in 2003. Between 1998 and 2004, he was a Graduate Assistant with the Department of Physics, DELSU. Since 2004 he has been an Assistant Lecturer with the Department of Physics Electronics, FUTA. He is currently on a PhD leave at the Aristotle University of Thessaloniki, Greece. He is the co-authored of a book; and has authored or co-authored more than 20 articles in refereed conference proceedings and journals as well as 7 advanced technical reports (on-line). He is interested in learning about the application of artificial intelligence to totally-integrated industrial network control systems and signal processing & image recognition systems design.

Mr. Akpan is a member of The IEEE, USA; The IET, UK; and The IOP, UK. He is one of the two Nigerian recipients of the 2005/2006 Bilateral Educational Agreement scholarship between The Federal Government of Nigeria and The Greek State Scholarships Foundation (IKY) for a Ph.D programme tenable in Greece.

Reginald A. O. Osakwe
received his B.Sc. degree in Physics from the University of Lagos, Nigeria in 1979; his M.Sc. in solid state Physics from the Purdue University, West Lafayette, Indiana, U.S.A. in 1984; and his Ph.D in electronics from the University of Ibadan, Nigeria in 1995.

Between 1991 and 2007, he was a lecturer with the Department of Physics, Delta State University, Abraka, Nigeria. Since 2007 he has been with the Department of Physics, Federal University of Petroleum Resources, Effurun, Delta State, Nigeria where he is currently a Senior Lecturer. During the periods 1998 – 2009 he has been a visiting lecturer to the Shell Intensive Training Programme (SITP), Warri, Delta State under the Shell Petroleum Development Company (SPDC) of Nigeria. His research interests are on microprocessor architecture, embedded computer systems, chaotic dynamics, and image and signal processing systems.
Dr. Osakwe is a member of the Nigeria Institute of Physics (NIP). He was a recipient of The Federal Government of Nigeria Scholarship to study in the U.S.A. leading to the award of an M.Sc.
A PERFORMANCE ANALYSIS OF DELTA AND HUFFMAN COMPRESSION ALGORITHMS

A.O. Oke
Department of Computer Science and Engineering,
Ladoke Akintola University of Technology, Ogbomoso,
Nigeria
Email: alitemi2006@yahoo.com

&

O.A. Fakolujo
Department of Electrical & Electronic Engineering,
University of Ibadan, Ibadan
Nigeria
Email: ao.fakolujo@mail.ui.edu.ng

&

O.J. Emuoyibofarhe
Department of Computer Science and Engineering
Ladoke Akintola University of Technology,
Ogbomoso,
Nigeria
Email: eojustice@gmail.com

Received 30 June, 2009

ABSTRACT
With the recent trend in Information and Communication Technology, Storage and Transfer of data and Information are two vital issues which have Cost and Speed implication respectively. Large volume of data (text or image) is constantly being processed on the internet or on a Personal Computer, which has led to the Upgrade of current System. Hence the need for compression, which reduces storage capacity and effect Speed of transfer. Data Compression is the act of reducing the size of a file by minimizing redundant data. In a text file, redundant data can be frequently occurring characters or common vowels. This research involves a comparative performance analysis of Huffman and Delta Compression schemes. A compression program is used to convert data from an easy-to-use format (ASCII) to one optimized for compactness. Huffman and Delta algorithms were implemented using C#. Result was also presented on the efficiency of the former based on three parameters: the number of bit, compression ratio and percentage of compression. It was discovered that Huffman algorithm for data compression performs better, since it can store / transmit the least number of bits. The average compression percentage for Huffman and Delta algorithm was found to be 39% and 45% respectively. Which simply implies that for a large text file, Huffman algorithm will achieve a 39% reduction in the file size and as such increase the capacity of the storage medium.

Keywords: Data compression, Huffman algorithm, Delta algorithm
I. INTRODUCTION

Data compression is the act of forcing data together to occupy less space or pressing data together. Data compression reduces the size of a file by minimizing repetitive data. In a text file, repetitive data can be frequently occurring characters, such as the space character, or common vowels, such as the letters e and a, it can also be frequently occurring character strings. Data compressing creates a compressed version of a file by minimizing this repetitive data and as such produces faster transmission [12].

Most human communication is inherently redundant, this does not imply waste; rather human beings use that redundancy as a continual crosscheck on what information is really being sent and meant. For example, in face-to-face conversation much more information is being exchanged than just the words. Facial expressions, tones of voice, limb positions and movement, and other less obvious cues all contribute to the information stream flowing between two people having a conversation. Even though much of the information is duplicated but compression tends to remove duplication which is referred to as the redundancy. In other words, data compression could be said to deal with how to take a large collection of binary bits, the life blood of digital technologies, and replace this collection with a small compressed version of the original bits [11]. In data communications, compression is a technique applied either in advance to information to be transmitted or dynamically to an information stream being transmitted. The underlying technology is essentially the same in both cases: removal of repetitive information or expression of the information in a more compact form is used to reduce the total number of bytes that must pass over a communication medium in order to reduce the time the medium is occupied by a given transmission to a minimum [13].

A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for instance) which contains the same information but whose length is as small as possible. Data compression has important application in the areas of data transmission and data storage. Many data processing applications require storage of large volumes of data, and the proliferation of computer communication networks is resulting in massive transfer of data over communication links. Compressing data to be stored or transmitted reduces the costs involved. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data at a higher, thus faster, level of the storage hierarchy and reduce the load on the input / output channels of the computer system.

This research involves a comparative performance analysis of Huffman and Delta compression schemes. Huffman and Delta algorithms were implemented using C#. The three parameters used for evaluating the efficiency of the former over the later are: the number of bits stored / transmitted, compression ratio and percentage of compression.
II. CODING

A code is a mapping of source messages (words from the source alphabet) into code words (words of the code alphabet). The source messages are the basic units into which the string to be represented is partitioned. These basic units may be single symbols from the source alphabet, or they may be strings of symbols. Coding is a very general term encompassing any special representation of data, which satisfy a given need. In communications systems, coding can be said to be the altering of the characteristics of a signal to make the signal more suitable for an intended application, such as optimizing the signal for transmission, improving transmission quality and fidelity, modifying the signal spectrum, increasing the information content and providing error detection and/or correction.

Data and programs are almost invariably entered in alphanumeric form, and the internal operation of computers makes extensive use of alphanumeric codes. Alphabetic characters must be coded in binary form in order to store in computer memories (binary form because computer memories process it data in binary). These characters are letters such as A, a, B, b and so on. A wide spread code for alphabetic characters is the American Standard code for information interchange (ASCII). ASCII [1] contains all the uppercase and lowercase English letters, the ten numeric digits and symbols. Another popular code for representing characters is the Extended Binary Coded Decimal Interchange Code (EBCDIC). ASCII has become an international standard published by the American National Standards Institute. It is an 8-bit code. 7 bits define one of 128 characters and the 8th bit is an optional parity bit.

III. HUFFMAN AND DELTA CODING

Early compression work mainly focused on disk space savings and I/O performance [3,5,7]. However, later works acknowledges that compression can also lead to better CPU performance, as long as decompression costs are low [10]. Dictionary based domain compression, a lightweight compression method where data values are mapped to fixed length codes, is used in the implementation of many applications and has been a subject of extensive research [2]. Entropy coding techniques, including Huffman encoding [6], are considered heavy-weight techniques. Both Huffman and arithmetic encoding have been studied and modified [3, 9, 10, 17].

The Huffman algorithm and its variant, the Canonical Huffman coding scheme, has been reported in [9] to be effective and efficient in data compression. The canonical Huffman coding is a reordering of the nodes of a Huffman tree so that shorter codes have longer length value than longer codes with small length. In [14], C-Store was used to delta code data for column-wise storage and compression. Huffman coding assigns each column value a variable length code chosen so that more frequent values within the column are assigned shorter codes. The basic idea here is assigning short code words to those input blocks with high probabilities and long code words to those with low probabilities. Huffman code can be categorized under the lossless fixed to variable code. The algorithm for constructing a Huffman code to represent items with given relative frequencies of occurrence
proceeds in two phases [8]. First one constructs a Huffman tree based on the relative probabilities. Second, using the Huffman tree, one constructs the code words that go with the given relative frequencies. A simple example will illustrate the idea involved as shown in Fig 3.1 and Table 3.1 in the Appendix (All other figures and tables in this paper can also be found in the Appendix). A code is needed to represent 10 types of information (a - j) with relative frequencies.

In all there are 100 items to represent. One begins the tree by assigning a vertex of a tree for each item, labeled so that the relative frequencies increase from left to right. We construct a tree by adding a new vertex to the previously existing construction by selecting the two vertices (say v and w) in the current structure with the smallest labels (in a case of a tie, any two is picked) and adding a new vertex to the structure with a label which is the sum of the labels at v and w. The new vertex is placed above and between the two vertices v and w. The tree is now used to construct the code words associated with the ten pieces of information as explained below. The top vertex of the tree (the root) is labeled with a blank string. As we move down the tree, the next vertex one gets to is labeled with a binary string by appending 0 or 1 to the right according to whether one moves to the next vertex by taking a right branch or a left branch. (e.g. “a” vertex has the string”010” associated with it). This process winds up assigning short codeword to characters which occur frequently and long codeword to characters that rarely occur.

Recall the use of the Greek letter delta (δ) in science, engineering, and mathematics to denote the change in a variable, the delta encoding also refers to several techniques that store data as the difference between successive samples (or characters), rather than directly storing the samples themselves. Fig. 3.2 shows an example of how this is done. The first value in the delta-encoded file is the same as the first value in the original data. The subsequent values in the encoded file are equal to the difference (delta) between the corresponding value in the input file, and the previous value in the input file.

Delta coding is a branch of universal code. With universal codes it is not necessary to know the exact probabilities with which the source messages appear; but it is sufficient to know the probabilities distribution only to the extent that the source messages can be ranked in probability order; by mapping messages in order of decreasing probability to code words in order of increasing length. Delta encoding is one of the universal coding schemes, which map the set of positive integers onto the set of binary code words [4].

To code a number, the following algorithm holds:

Step 1: Write it in binary.
Step 2: Count the bits, remove the leading one, and write that number in binary preceding the previous bit string.
Step 3: Subtract 1 from the number of bits written in step 2 and prepend (affix) the zeros. The code begins:
Explanation of Table 3.3: Source message is the alphanumeric characters, Frequency is the number of occurrence of the character in the message, and rank is the order of most frequent character: the most frequent character is ranked one (1), followed by the next most frequent and etc. Codeword is the code generated for Delta algorithm.

IV. RESULTS

The message “go go gophers” was implemented using Huffman and Delta algorithm. The code generated from the two algorithms along with the ASCII equivalent is as shown in Table 4.

<table>
<thead>
<tr>
<th>Integer</th>
<th>Binary code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>0101</td>
</tr>
<tr>
<td>4</td>
<td>01100</td>
</tr>
<tr>
<td>5</td>
<td>01101</td>
</tr>
<tr>
<td>6</td>
<td>01110</td>
</tr>
<tr>
<td>7</td>
<td>01111</td>
</tr>
<tr>
<td>8</td>
<td>00100000</td>
</tr>
</tbody>
</table>

IV.1. THE LENGTH OF BIT IN STORAGE MEDIUM

Standard Code (ASCII)
Number of characters: 13
Number of bit per character: 8
Word length: 13 * 8
Total bits (in storage): 104 bits

Huffman Code
Since it represents character with variable length:
Total bits = the sum of the number of bit represented by each character.
Total bits (in storage): 37 bits

Delta Code

It also represents its character with variable length:
Hence Total bits (in storage): 43 bits

IV.2. COMPRESSION CHART

This chart provides a graphical representation of the result analyzed based on the compression algorithms. The samples tested are as follows:
S1 = go go gophers
S2 = this is a boy
S3 = go go gophers gum gum gophers
S4 = go go gophers gum gum gophers go go gum
S5 = peter piper picked a peck of pickle pepper
S6 = go go gophers gum gum gophers go go gum gum gophers
S7 = peter piper picked a peck of pickle pepper a peck of pickle pepper peter piper picked, if peter piper picked a peck of pickle pepper where is the peck of pickle pepper peter piper picked

The first parameter of evaluation is the number of bits stored/transmitted using uncompressed ASCII, compressed HUFFMAN and DELTA bits against character text, as shown in Table 4.1a and analyzed in Table 4.1b. The same information is represented in Fig. 4.1. The second parameter is the compression ratio of both algorithms. Fig. 4.2 shows that Huffman has a higher Compression ratio. The third parameter of evaluation is percentage of compression. From Fig. 4.3, Huffman algorithm has the least percentage of compression.

IV.3. COMMUNICATION TRANSMISSION STREAM

The sample “go go gophers” was transmitted over a communication line, using a serial data transfer, Figs. 4.4, 4.5 and 4.6 show the bit stream of the message sent for the normal ASCII format, the generated Huffman and
delta code. In serial transfer of data, a character is transmitted with a start bit, two stop bits and the character bit of the different format. In the order start bit, character bit: starting with the least significant bit and stop bit [15,16].

V. DISCUSSION OF RESULT

The compression algorithm that performs better was evaluated on three parameters. These are: the number of bits stored/transmitted, the compression ratio and the percentage of compression. It was discovered that Huffman algorithm for data compression performs better, since it can store / transmit the least number of bits as shown in Fig.4.1. Fig.4.2 shows Huffman having the higher compression ratio, the implication is that it compresses more than Delta algorithm. Fig.4.3 also reveals Huffman having the least percentage of compression. The average compression percentage was calculated for all samples as 39% and 45% for Huffman and Delta algorithm respectively. This simply implies that for a large text file, Huffman algorithm will achieve a 39% reduction in the file size and as such increase the capacity of the storage medium. In transmission, the effect will be to reduce the time the transmission channel is occupied and as such result in faster transmission as shown in Fig.4.5.

It is also observed that when we have a text file having a number of characters occurring frequently, there is every tendency of achieving a better compression than one with characters that rarely occurs taking a larger percentage. This can be seen from the difference that occurs in the number of bit in Table 4.2 for samples S1 and S2. 104 being the same number of bit for both samples in ASCII (uncompressed form) but 37 and 43 for sample S1; 40 and 52 for sample S2 in Huffman and Delta (compressed form) respectively.

VI. CONCLUSION

Huffman algorithm was discovered to be the better of the two schemes on the premise of the parameters of comparison, which accounts for the use of this algorithm in many compression processes. The advantage of Huffman coding is in the average number of bits per character transmitted. In addition, communication costs are beginning to dominate storage and processing costs, so that Huffman algorithm, which is a variable “length-coding scheme”, reduces communication costs and consequently results to faster transmission.
APPENDIX

Table 3.1: Information to be Coded

<table>
<thead>
<tr>
<th>Information</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>17</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>27</td>
<td>5</td>
<td>4</td>
<td>13</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 3.1: Huffman Tree

Table 3.2: Code table

<table>
<thead>
<tr>
<th>X</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>010</td>
<td>11110</td>
<td>1010</td>
<td>110</td>
<td>00</td>
<td>1011</td>
<td>1110</td>
<td>100</td>
<td>011</td>
<td>11111</td>
</tr>
</tbody>
</table>

Original data stream: 17 19 24 24 21 15 10 89 95 96 95 94 94 90...

Delta encoding: 17 2 5 0 -3 -6 -5 79 6 1 0 -1 -1 0 4...

Fig. 3.2: Difference Between Successive Samples
Table 3.3: Elias code generation.

<table>
<thead>
<tr>
<th>Source message</th>
<th>Frequency</th>
<th>Rank</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>8</td>
<td>1</td>
<td>delta (1) = 1</td>
</tr>
<tr>
<td>f</td>
<td>7</td>
<td>2</td>
<td>delta (2) = 0100</td>
</tr>
<tr>
<td>e</td>
<td>6</td>
<td>3</td>
<td>delta (3) = 0101</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>4</td>
<td>delta (4) = 01100</td>
</tr>
<tr>
<td>space</td>
<td>5</td>
<td>5</td>
<td>delta (5) = 01101</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>6</td>
<td>delta (6) = 01110</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>7</td>
<td>delta (7) = 01111</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>8</td>
<td>delta (8) = 00100000</td>
</tr>
</tbody>
</table>

Table 4.1a: Bits generated for the different codes

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
<th>HUFFMAN Code</th>
<th>DELTA Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>11100111</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>01101111</td>
<td>01</td>
<td>0100</td>
</tr>
<tr>
<td>Space</td>
<td>00100000</td>
<td>111</td>
<td>0101</td>
</tr>
<tr>
<td>P</td>
<td>11101000</td>
<td>000</td>
<td>01100</td>
</tr>
<tr>
<td>H</td>
<td>11101000</td>
<td>0011</td>
<td>01101</td>
</tr>
<tr>
<td>E</td>
<td>01100101</td>
<td>0010</td>
<td>01110</td>
</tr>
<tr>
<td>R</td>
<td>01110010</td>
<td>1101</td>
<td>01111</td>
</tr>
<tr>
<td>S</td>
<td>11110011</td>
<td>1100</td>
<td>00100000</td>
</tr>
</tbody>
</table>

Table 4.1b: Analysis of the Generated Code

**Standard code (ASCII) will store the message as**

```
G 01101111 00100000 11100111 01101111 00100000
  g  o      Space  g  o      Space
11100111 01101111 11110000 11101000 01100101 01110010
  g  o      p      h      e      r
11110011
  s
```

**Huffman code will store the message as**

```
10 01 111 10 01 111 10 01 000 0011 0010
  g  o      space  g  o      space  g  o      p      h      e
1101 1100
  r      s
```

**Delta code stores message as**

```
1 0100 0101 1 0100 0101 1 0100 01100 01101 01110 01111 00100000
  g  o      space  g  o      space  g  o      p      h      e
  r      s
```
Table 4.2: Number of bits stored / transmitted

<table>
<thead>
<tr>
<th>TEXT</th>
<th>ASCII</th>
<th>HUFFMAN</th>
<th>DELTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>104</td>
<td>37</td>
<td>43</td>
</tr>
<tr>
<td>S2</td>
<td>104</td>
<td>40</td>
<td>52</td>
</tr>
<tr>
<td>S3</td>
<td>232</td>
<td>93</td>
<td>110</td>
</tr>
<tr>
<td>S4</td>
<td>312</td>
<td>120</td>
<td>131</td>
</tr>
<tr>
<td>S5</td>
<td>344</td>
<td>139</td>
<td>150</td>
</tr>
<tr>
<td>S6</td>
<td>440</td>
<td>173</td>
<td>192</td>
</tr>
<tr>
<td>S7</td>
<td>1488</td>
<td>633</td>
<td>673</td>
</tr>
</tbody>
</table>

![Uncompressed and Compressed data](image)

Fig. 4.1: Uncompressed and Compressed data
Fig. 4.2: Compression ratio of Huffman versus Delta

Fig. 4.3: Compression Percentage of Huffman and Delta
Fig. 4.4: Bit stream for transmission of “go go gophers” in ASCII

Fig. 4.5: Bit stream for transmission of “go go gophers” in HUFFMAN
Fig. 4.6. Bit stream for transmission of “go go gohers” in DELTA
REFERENCES

Alice Oluwafunke Oke is a lecturer in the Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria. She graduated with B.Tech (Hons) Computer Engineering (1998) and M.Tech Computer Science (2005) from the same Institution. She has publications in reputable journals and her research interest is in image analysis, specifically image restoration algorithms. She is a full member of the following professional bodies: Computer Professionals (Registered Council of Nigeria) CPN; Council for the Regulation of Engineering in Nigeria (COREN) and Associate member, Nigeria Computer Society (NCS). She is presently working on her Ph.D in Computer Science.

O. Justice Emuoyibofarhe (Ph.D., MNCS, MCPN, MIEEE) is a Senior lecturer in the Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria and also in charge of the software simulation laboratory. He is a very good blend of both software and hardware Knowledge. He has been assembling, installing and maintaining systems in both single user and multi-user environment for close to a decade.

O. A. Fakolujo graduated from the University of Ife (Now Obafemi Awolowo University), where he earned the B. Sc. degree in Electrical/Electronic Engineering with first class honours and had the best result in his graduating class. He was a University Scholar during his undergraduate programme. Dr Fakolujo earned his doctorate (Ph. D.) degree from the University of London and the Diploma of Membership of the Imperial College (D.I.C.). He is a Senior Lecturer in the Department of Electrical & Electronic Engineering, University of Ibadan. He had at various times, between 1992 and January 31, 2010, been the Acting Head of his Department. Professionally, Dr. Fakolujo is a Corporate Member of the Nigerian Society of Engineers (MNSE), Senior Member of the Institute of Electrical & Electronics Engineers (SMIEEE) and a COREN Registered Engineer (R.Eng). Dr. Fakolujo has published in reputable journals (locally and internationally). His research interest includes, Microelectronics, Computer and Microprocessor-Based System Design, Signal processing, Biometrics and Design of Security Systems.
IEEE COMPUTER SOCIETY CERTIFICATION PROGRAMME

The IEEE Computer Society organizes a Certified Software Development Professional (CSDP) programme for software developers and software engineers. The goal is to increase the global awareness of the programme and achieve dramatic growth in the number of CSDP certification holders worldwide. All Senior Members of the IEEE and all licensed engineers are eligible to take the certification examination. Others who have an advanced software engineering degree and 2 years of experience, or alternatively, have 4 years of software development/engineering experience, are also eligible. The testing fee for IEEE members is $450 (consisting of $100 application fee and $350 examination fee) and $550 for non-members (consisting of $100 application fee and $450 examination fee); these fees are subject to tax charges. Detailed information about the programme can be obtained from:

http://www.computer.org/certification

Enquiries on the certification programme may be directed to:

csdp@computer.org

IEEE COMMUNICATIONS SOCIETY CERTIFICATION PROGRAMME

The IEEE Communications Society (COMSOC) has launched her Wireless Communication Engineering Technologies Certification (WCET) programme in September 2008. Detailed information about the ComSoc WCET certification may be obtained from:

http://www.comsoc.org/cert

or

http://www.ieee-wcet.org/application
ADVERTIZE YOUR PRODUCTS AND SERVICES IN THE

AFRICAN JOURNAL OF COMPUTING AND ICT

(ISN 2006 -1781)

FOR DETAILED INFORMATION ON SUBSCRIPTION AND ADVERTIZE-
MENT RATES, PLEASE CONTACT THE PRINCIPAL CONSULTANT
(dewadeconsult@yahoo.com)
AFRICAN JOURNAL OF COMPUTING AND ICT
(IEEE NIGERIA COMPUTER CHAPTER)
ISSN 2006-1781

INSTRUCTIONS TO AUTHORS, SUBSCRIBERS AND ADVERTIZERS

MODE OF SUBMISSION
Papers intended for consideration, and other editorial correspondence, should be sent to the Editor – in – Chief preferably as email attachment in MS WORD (or in exceptional case, in PDF with LATEX source file included). The email should preferably have in its SUBJECT line ‘PAPER FOR AFRICAN J. COMP. & ICT’. A prospective author may suggest to the Editor-in-Chief names of reviewers suitable for his/her paper. The editor is however not obliged to use any of the suggested reviewers. Every submission should be prepared in a form that approximates the general IEEE style guide\(^1\). Each paper should be accompanied by an informative Abstract and a list of keywords. The title of the paper and the names, institutional (and email) addresses of the author should be clearly indicated. The body of the paper should begin with an Introduction and end with a Discussion and/or Conclusion section. References should follow the format for journals, conference proceedings and books


A reference should be cited in the body in square brackets, e.g. [4], and all citations should be included in the list of references at the end of the paper. Accepted papers would be published in a double column format. Every author (or group of authors) of an accepted paper shall be required to pay a publication charge of US 100.00 (or the Nigerian currency equivalent, which is presently approximately NGN 15,000), to defray administrative/processing cost and the reviewers’ honoraria; an author whose paper is not recommended for publication would only be required to pay a fraction of this amount to cover the reviewers’ honoraria. This charge shall be paid to the designated bank account of the journal. Every author of a published paper is entitled to a reprint of his/her paper.

SCOPE OF ARTICLES
Submissions to the journal must be relevant to computer scientists, computer engineers, information technology professionals, engineering scientists and allied practitioners in the industry and the academia. Papers are welcome from pure/ applied computing and ICT, and allied areas including, software engineering, computer communication/networks, human computer interaction, data mining, discrete/data structures, intelligent library buildings, differential equations of computing, design & analysis of algorithms, information systems, bioinformatics, data compression, information management etc. Papers are welcome from Nigeria, Africa and the international community. The journal considers original research papers, case reports and review papers. An author must clearly justify the importance of his submission to the readership of the journal. A prospective author need not be a member of the (Computer Chapter) IEEE Nigeria Section before his/her paper is published. The journal currently publishes only a print edition, twice in a year, one in the first half of the year and the other in the second half. Online edition would soon follow. Papers should be easily readable and be written in lucid language comprehensible to readers who are not working within the immediate subject area.

INFORMATION TO SUBSCRIBERS AND ADVERTIZERS
The subscription fee for one single copy of the journal is US 14.00, excluding postage (or equivalent Nigerian currency, which is presently approximately NGN2,000). Discount shall be offered to individuals, institutions and organizations making bulk purchases. All correspondence on subscription and advertisement should be directed to the Principal Consultant at the address below.

AFRICAN JOURNAL OF COMPUTING AND ICT is published for the IEEE Nigeria Computer Chapter by Dewade Systems Consult (Computer Services, Training & Consultancy), Off Upper Niger River Basin Development Authority Main Gate, Bosso Road, Taidna, G.P.O. Box 4222, Minna, Niger State, Federal Republic of Nigeria; URL: http://www.dewadeconsult.com; Email: dewadeconsult@yahoo.com; Telephone: +234(0) 8138936969, (0)8033608519

---

\(^1\) http://www.ieee.org/organizations/pubs/transactions/stylesheets.htm
AIM AND OBJECTIVE OF JOURNAL

_African Journal of Computing and ICT_ was established by the Nigeria Computer Chapter of the Institute of Electrical and Electronics Engineers (IEEE) Inc, New York, USA to publish concise technical papers on Computer Science, Information Technology, Information Management and related IEEE designated fields, in a relatively short time. First time review of a submitted paper is expected to be completed within 6 months. The goal of the journal is to contribute to rapid industrial development of Nigeria in particular, and Africa in general, via the publication of topical and relevant papers from a theoretic/academic and practical/industrial points of view. Submitted papers should contain substantial ideas which have the potential of aiding an invention or a technological breakthrough for African use. All correspondence pertaining to the Computer Chapter of the IEEE Nigeria Section should be sent to the Chair of the Chapter, Dr. ‘Dele Oluwade (bamideleoluwade@computer.org). General information on the aim and objective as well as membership of IEEE, IEEE Computer Society and the IEEE Nigeria Section can be found on the following websites respectively: [http://www.ieee.org](http://www.ieee.org); [http://computer.org](http://computer.org); [http://www.ieee.org/nigeria](http://www.ieee.org/nigeria). Correspondence to the IEEE Nigeria Section should be sent to the Chair of the Section, Engr. Tunde Salihu (tunde_salihu@yahoo.com).

_African Journal of Computing and ICT is published for the IEEE Nigeria Computer Chapter by Dewade Systems Consult, Nigeria (Reg.No. BN MN 0000524). Email:dewadeconsult@yahoo.com_