NEAR EAST UNIVERSITY

GRADUATE SCHOOL OF APPLIED SCIENCES

ANALYSIS AND APPLICATION OF LDPC CODES TO IMAGE TRANSMISSION AND RESTORATION

Sameh MASHAQI

Master Thesis

Department of Electrical and Electronic Engineering

Nicosia - 2007

ACKNOWLEDGMENTS

This work would not have been possible without the generous help of God and then the following people as well as their significant contribution to my work.

Dr. Ali SERENER: I would like to sincerely thank for his invaluable supervision, support and encouragement through this work and for introducing me to the world of communications.

Assoc. Prof .Dr. Adnan KHASHMAN: I would like to sincerely thank for his supervision, suggestions and his instructions through the master degree duration and undergraduate years and for always being kind and helpful to me all these years.

I would like especially to express my sincere thanks and dedication to my parents and family and gift them this work for their always constant love and supporting, spiritual and financial, in my decisions through the years.

I wish to thank the administration of Near East University for the scholarship of master program that made the work possible.

Finally, I would like to thank my friends specially Mohammed ELAMIN for his help in Matlab programming and their helpful ideas.

Sameh MASHAQI

ABSTRACT

Low-Density Parity-Check (LDPC) codes are one of the recent topics in coding theory today. Originally invented more than forty years ago, they have been the focus of many researchers in the last few years and are included in the latest digital video broadcasting via satellite standard (DVB-S2). Unlike many other classes of codes, LDPC codes are already equipped with a fast, probabilistic decoding algorithm. This makes LDPC codes not only attractive from a theoretical point of view, but also very suitable for practical applications. This thesis presents the application of LDPC error correction codes in digital image transmission and restoration on images with fixed size (256 by 256 pixels) in grayscale format. The results show better restoration of images by LDPC codes compared to other well-known image restoration methods when degradation on images is caused by additive white Gaussian noise (AWGN) only. The drawback is that the restoration process is much slower when LDPC codes are used. Consequently, LDPC codes can be used in applications such as medical imaging, deep-space communications and multimedia, where delay is not an issue.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

APP: A Posterior Probability

ARQ: Automatic-Repeat-Request

AWGN: Additive White Gaussian Noise

BCH: Bose-Chaudhuri-Hochquenghem

CDF: Cumulative Distribution Function

DSL: Digital Subscriber Line

ECC: Error Control Coding

FEC: Forward Error Correction

LLR: Log-Likelihood Ratio

LDPC: Low-Density Parity-Check

LSB: Least Significant Bit

MAP: Maximum A Posteriori

MSB: Most Significant Bit

NSC: Nonsystematic Convolutional

PDF: Probability Density Function

PSNR: Peak Signal-To-Noise Ratio

RMS: Root Mean Square

SNR: Signal-To-Noise Ratio

SOVA: Soft Output Viterbi Algorithm

TCM: Trellis Coded Modulation

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Traditional communication systems are made up of three major components: the transmitter, the channel and the receiver. The transmitter transmits a signal across a noisy channel which introduces distortion to that signal. The receiver receives the now distorted signal and attempts to recover the original signal.

In the design of any communication system the designers must consider the channel distortion as it will cause errors possibly rendering the received data unusable to the receiver. In general, a certain level of signal distortion may be acceptable but it may be necessary to design a system in which the receiver is capable of correcting the errors in the received data in order to bring the distortion down to an acceptable level. This can be accomplished through the use of an error-correcting coding scheme.

An error-correcting coding scheme adds two additional components to the communication system described above. A channel encoder which adds redundancy data to the transmitted data, and channel decoder which exploits this redundancy in order to find and correct errors caused by the channel noise.

Error control coding has been frequently used in many wireless systems as digital mobile communications started evolving in the early 1980s. By adding redundant data to transmitted information, coding techniques could detect and correct errors introduced in the channel.

Low-density parity-check (LDPC) codes, originally invented by Gallager in 1962 is a linear block code whose parity check matrix is composed of ‘0’ elements dominantly. But, since realization was regarded to be impossible in those days, it had been forgotten for a long time until Mackay rediscovered it in 1996. LDPC code shows good error correcting capability with iterative decoding by the sum-product algorithm.

One of the recent topics in coding theory today are LDPC codes as an ideal candidate for next generation communication systems like wireless, wireline, satellite, magnetic recording channels and fiber optical applications.

In the past few years, Gallager’s Low-Density Parity-Check (LDPC) code have received a lot of attention and tremendous efforts have been devoted to analyze and improve their error-correcting performance, and their performances have been the subject of much recent researchers experimentation and analysis. The interest in these codes stems from their near Shannon limit performance using iterative decoding on the

AWGN channel with rather low implementation complexity and therefore increasingly being applied for error control in various fields of data communications.

These codes have flexible block lengths and code rates, and may be used in the area of communications and data storage. The simulation results show that they have better bit-error-rate decoding performance and lower error floors in additive white Gaussian noise, their simple descriptions and implementations, and their amenability to rigorous theoretical analysis. LDPC codes have become strong competitors to turbo codes for error control in many communication and digital storage systems where high reliability is required. LDPC codes can be decoded with various decoding methods, ranging from low to high complexity and from reasonably good to very good performance.

This thesis explores low-density parity-check codes and designs and analyzes a system employing a new application or contribution of LDPC codes on the image transmission and restoration techniques in spatial domain for images with dimension of 256 by 256 pixels in grayscale format.

The aims of work presented in this thesis are:

- To investigate Gallager's LDPC error control codes and their structure.
- To design and simulate a Matlab program for the LDPC encoder and LDPC decoder.
- To investigate the performance of LDPC error control codes over AWGN channel.
- Investigate the image restoration techniques in spatial domain when the only degradation is caused by Gaussian noise.
- To ensure the LDPC error control codes performance over the image restoration techniques as a contribution of the thesis to get better results for filtering in spatial domain.
- To implement the real-life applications design and simulate Matlab program to use the major three filters of image restoration techniques in spatial domain to filter any image distorted by Gaussian noise.
- To investigate the differences for both systems with comparison criteria for original and reconstructed images using Matlab simulation.

This thesis organized into four chapters as follow:

Chapter 1 is an introduction to communication systems. Basics of structure of communication systems, error control coding and their types also are presented.

Chapter 2 presents the LDPC codes and their basics and the iterative decoding algorithms.

Chapter 3 presents a background of image restoration techniques and various filters that work in the spatial domain.

Chapter 4 presents the proposed system of LDPC code in detail and compares it with to image restoration techniques that work in the spatial domain. It also describes the comparison criteria.

Chapter 5 presents the results of the proposed system in detail and compares it with to image restoration techniques results that work in the spatial domain.

CHAPTER ONE

Communication System Overview

1.1 Overview

In some communication systems, only error-detecting capability is required, and messages received with errors are retransmitted. These are known as automatic-repeat-request (ARQ) systems. On the other hand, systems involving long round-trip delays suffer from retransmission. Such systems, rather, correct errors at the receiver without asking for retransmission and are known as forward error correction (FEC) systems. Figure 1.1 shows the model of a digital communication system using such an approach, and this is the model used in this thesis.

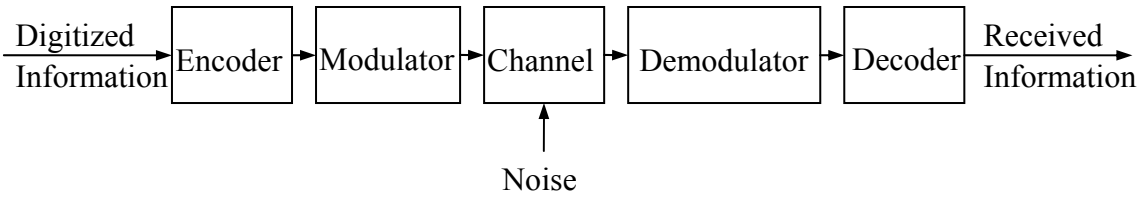


Figure 1.1 Digital Communications System

This chapter gives a brief overview of digital communication system. It begins with the background information about the components of the system and a brief introduction to error control coding.

1.2 Digital Communication Systems

A digital communication system is described as follows:

- A digital information source produces a finite set of possible messages.
- A typewriter is a good example of a digital source. There are a finite number of characters (messages) that can be emitted by this source.
- An analog information source produces messages that are defined on a continuum.

- A microphone is a good example of an analog source. The output voltage describes the information in the sound, and it is distributed over a continuous range of values.
- A digital communication system transfers information from a digital source to a destination.
- An analog communication system transfers information from an analog source to a destination.

Strictly speaking, a digital waveform is defined as a function of time that can have only a discrete set of values. If the digital waveform is a binary waveform, only two values are allowed. An analog waveform is a function of time that has a continuous range of values.

An electronic digital communication system usually has voltages and currents that have digital waveforms; however, it may have analog waveforms. For example, the information from a binary source may be transmitted to the destination using a sine wave of 1000 Hz to represent a binary 1 and a sine wave of 500 Hz to represent a binary 0. Here the digital source information is transmitted to the sink by use of analog waveforms. But this is still called a digital communication system.

Digital communication has a number of advantages [1]:

- Relatively inexpensive digital circuits may be used.
- Privacy is preserved by using data encryption.
- Greater dynamic range (the difference between largest and smallest value) is possible.
- Data from voice, video, and data sources may be merged and transmitted over a common digital transmission system.
- In long-distance systems noise does not accumulate from repeater to repeater.
- Errors in detected data may be small, even when there is a large amount of noise on the received signal.
- Errors may often be corrected by the use of coding.

Digital communication also has disadvantages [1]:

- Generally, more bandwidth is required than that for analog systems.
- Synchronization is required.

1.2.1 Channel Encoder

An encoder is used to change a signal or data into a code. The code may serve any of a number of purposes such as compressing information for transmission or storage, encrypting or adding redundancies to the input code, translating from one code to another. This is usually done by means of a programmed algorithm, especially if any part is digital, while most analog encoding is done with analog circuitry.

1.2.2 Modulator

A modulator converts digital data into a carrier waveform by assigning a different waveform to each possible symbol. Thus, there are a finite number of different waveforms corresponding to the finite number of possible symbols. For example, in BPSK modulation, a waveform $s_0(t)$ is assigned to a binary '0' and $s_1(t)$ to a binary '1', where

$$s_0(t) = A\cos\left(2\pi f_c t\right)$$

$$s_1(t) = A\cos\left(2\pi f_c t + \pi\right),\ 0 \le t \le T \qquad (1.1)$$

here, A is the waveform magnitude, f_c is the carrier frequency and T is the waveform period [12].

Note that $s_0(t) = -s_1(t)$.

1.2.3 Communication Channels

The communication channel provides a connection through which the information-bearing signal propagates. It is perhaps the most important component of a communication system. There are many different types of physical communication channels, such as:

- Wired channels
- Wireless channels
- Fiber optic channels
- Underwater acoustic channels
- Storage channels

Different kinds of channels can have very different characteristics. In order to design an efficient digital communication system over a specific communication channel, our channel model describes the physical communication channel as well as

the properties of the equipments, such as antennas and amplifiers, necessary to access the channel. Notice that the major characteristic of a communication channels interested distorts the information-bearing signal. Some common channel defects are:

- Thermal noise in electronic devices
- Signal attenuation
- Amplitude and phase distortion
- Multi path distortion
- Finite-bandwidth (low-pass filter) distortion
- Impulsive noise

Based on knowledge of these channel defects, we construct the generic channel model. Suppose the symbol $s(t)$ denotes the transmitted signal at the output of the modulator. Then it is found that the following linear filter model (see Figure 1.2) sufficiently approximates the behaviors of many typical communication channels:

$$r(t) = \int_{-\infty}^{\infty} c(\tau, t)\ \ s(t-\tau)\ \ d\tau + n(t), \tag{1.2}$$

where $r(t)$ represents the received signal at the input of the demodulator, $n(t)$ is a random process which models the thermal and impulsive noises, and $c(\tau,t)$ is a linear time-varying filter which models the other channel distortions listed above. Note that the linear (time-varying) channel model in Equation (1.2) is very general and we work with simplifications of this model in many cases. Among the various common simplifications of the general model, the additive white Gaussian noise (AWGN) model is perhaps the most studied and most important. In the AWGN model, $c(\tau, t) = \delta(\tau)$ and Equation (1.2) reduces to

$$r(t) = s(t) + (t), \tag{1.3}$$

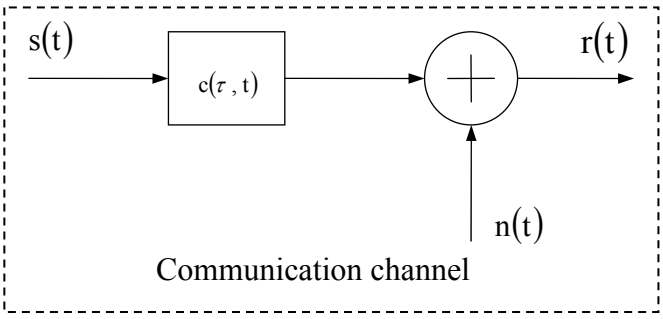


Figure 1.2 Linear Filter Channel Module

where $n(t)$ is a zero-mean wide-sense stationary Gaussian random process with autocorrelation function $R_n(\tau) = (N_0/2)\delta(\tau)$. The factor $N_0/2$ is called the two-sided noise spectral density of the noise $n(t)$. This model is primarily employed to represent the situation in which the only channel defect is the thermal noise in the electronic devices of a communication system [2].

1.2.4 Demodulator

The demodulator attempts to recover the transmitted waveform from the channel output $r(t)$. A matched filter followed by a sampler is typically used in an optimum demodulator. Figure 1.3 shows a fading channel with additive white Gaussian noise.

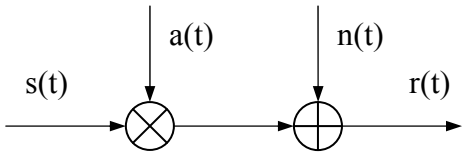


Figure 1.3 Fading Channel with Additive White Gaussian Noise

In Figure 1.4, s_i is given by Equation (1.1) where $i = 0, 1,$ $h(t)$ is the impulse response of the filter, T is the sampling time and the filter output $y(t)$ is

$$y(t) = r(t) * h(t) \tag{1.4}$$

where the symbol * denotes convolution. The demodulator output can be quantized into a finite number (Q) of levels. A demodulator with $Q > 2$ is called a soft decision demodulator. If the demodulator is quantized into $Q = 2$ levels, it is called a hard decision demodulator [12].

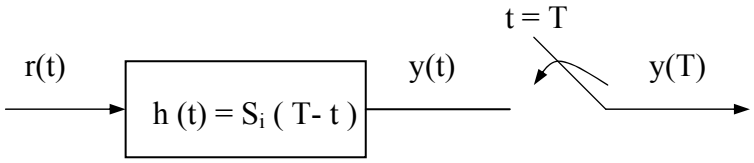


Figure 1.4 Matched Filter Demodulator

1.2.5 Channel Decoder

The channel decoder accepts the demodulator output and generates an estimate of the original information message that was presented to the channel encoder. The channel decoder can be designed to work with either hard or soft decision demodulators.

1.3 Error Control Codes

Error control coding (ECC) is a signal processing technique that protects digital information against transmission and storage errors. In contrast to its early days, ECC is now an integral part of almost all digital communications storage and computer systems. Error control coding (ECC) algorithms are widely used to improve performance in systems for digital communication and storage. Generally speaking, ECC allows a system to operate reliably at a low signal-to-noise ratio (SNR). A system that could not exist without ECC technology is compact disk (CD) digital audio [11].

Shannon's channel coding theorem states that a coding scheme always exists where by information can be transmitted and reconstructed with an error probability as small as desired. This holds true only if the data rate over the channel is less than the channel capacity. The theorem however, does not say how to find the appropriate coding for an information source and a channel. A major criterion for all communication systems is to achieve error free transmission. Errors, unfortunately, occur and methods of detecting and sometimes correcting errors are necessary [11].

Error control coding involves systematic addition of extra digits to the message. Extra check digits convey no information by themselves, but they make it possible to detect or correct errors [11].

The disadvantage of adding extra check digits is that the bandwidth of the channel increases.

There are two main types of error control codes, namely block codes and convolutional codes. There are many differences between block codes and convolutional codes. Block codes are based rigorously on finite field arithmetic and abstract algebra. They can be used to either detect or correct errors. Block codes accept a block of k information bits and produce a block of n coded bits. By predetermined rules, n-k redundant bits are added to the k information bits to form the n coded bits. Commonly, these codes are referred to as (n, k) block codes. Some of the commonly

used block codes are Hamming codes, Golay codes, BCH codes, and Reed Solomon codes (uses nonbinary symbols) [7].

This section will present primary information about convolutional codes and linear block codes and the other common codes.

1.3.1 Linear Block Codes

An (n, k) block code is completely defined by $M = 2^k$ binary sequences of length n called codeword. A code C consists of M code words c_i for $1 \le \mathrm{i} \le 2^k$.

$$C = \{\mathrm{c}_1, \mathrm{c}_2, ..., \mathrm{c}_M\}$$

Where each c_i is a sequence of length n with components equal to 0 or 1. A code is linear if any linear combination of two code words is also a code word. In the binary case this requires that if c_i and c_j are code words then $c_i \oplus c_j$ is also a code word, where $\oplus$ denote component-wise modulo-2 addition.

With this defection, it is readily seen that a linear block code is a linear block code is a k-dimensional subspace of an n-dimensional space. It is also obvious that the all zero sequence 0 is a code word of any linear block code since it can be written as $c_i \oplus c_j$ for any code word c_i. Note that according to the above definition linearly of a code only depends on the code words and not on the way that the information sequences (message) are mapped to the code words. However, it is natural to assume that if the information sequence x_1 (of length k) is mapped into the code word c_1 (of length n) then the information sequence x_2 is mapped into c_2, and then $x_1 \oplus x_2$ is mapped into $c_1 \oplus c_2$. Linear codes will be assumed to possess this special property from now on.

Let us consider a (5, 2) code which is defined by

$$c = \{00000, 10100, 01111, 11011\}$$

It is very easy to verify that this code is linear. If the mapping between the information sequences and code words is given by

$$\begin{aligned} 00 &\rightarrow 00000 \\ 01 &\rightarrow 01111 \\ 10 &\rightarrow 10100 \\ 11 &\rightarrow 11011 \end{aligned}$$

the special property mentioned above is satisfied as well. If the mapping is given by

$$
\begin{aligned}
00 &\rightarrow 10100 \\
01 &\rightarrow 01111 \\
10 &\rightarrow 00000 \\
11 &\rightarrow 11011
\end{aligned}
$$

the special property is not satisfied. However, in both cases the code is linear.

Here are definitions of some basic parameters that characterize a code [3]:

- The Hamming distance between two code words c_i and c_j is the number of bits at which the two code words differ, and is denoted by $d(c_i, c_j)$.
- The Hamming weights, or simply the weight, of a code word c_i is the number of bits of the codeword that are equal to 1 and is denoted by $w(c_i)$.
- The minimum distance of a code is the minimum Hamming distance between any two different code words; i.e.,

$$d_{\min} = \min_{\substack{c_i, c_j \\ i \neq j}} d(c_i, c_j) \tag{1.5}$$

- The minimum weight of a code is the minimum of the weights of the code words except the all zero code word.

$$w_{\min} = \min_{c_i \neq 0} w(c_i) \tag{1.6}$$

- One of the important consequences of this fact is in any linear code $d_{\min} = w_{\min}$.

1.3.1.1 Generator and Parity Check Matrices:

In an *(n, k)* linear block code let the codeword corresponding to the information sequences $e_1 = (1000...0)$, $e_2 = (0100...0)$, $e_3 = (0010...0), \ldots, e_k = (0000...1)$ be denoted by g_1, g_2, $g_3, \ldots$, g_k respectively, where each of the g_i sequences is a binary sequence of length *n*. Now, any information sequence $x = (x_1, x_2, x_3, ..., x_k)$ can be written as

$$x = \sum_{i=1}^{n} x_i e_i \tag{1.7}$$

and, therefore, the corresponding code word will be

$$c = \sum_{i=1}^{n} x_i g_i \tag{1.8}$$

If we define the generator matrix for this code as

$$G \overset{def}{=} \begin{bmatrix} g_1 \\ g_2 \\ . \\ . \\ . \\ g_k \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & . & . & . & g_{1n} \\ g_{21} & g_{22} & . & . & . & g_{2n} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ g_{k1} & g_{k2} & . & . & . & g_{kn} \end{bmatrix} \tag{1.9}$$

then, we can write

$$c = xG \tag{1.10}$$

This shows that any linear combination of the rows of the generator matrix is a code word. The generator matrix for any linear block code is a $k \times n$ matrix of rank k (because by definition the dimension of the subspace is k). The generator matrix of a code completely describes the code. When the generator matrix is given, the structure of an encoder is quite simple [3].

For example, in above mentioned (5, 2) code, the code words corresponding to information sequences (10) and (01) are (10100) and (01111), respectively. Therefore,

$$G = \begin{bmatrix} 10100 \\ 01111 \end{bmatrix} \tag{1.11}$$

it is seen that for the information sequence (x_1, x_2), the code word is given by

$$(c_1, c_2, c_3, c_4, c_5) = (x_1, x_2)G \tag{1.12}$$

or

$$\begin{aligned} c_1 &= x_1 \\ c_2 &= x_2 \\ c_3 &= x_1 \oplus x_2 \\ c_4 &= x_2 \\ c_5 &= x_2 \end{aligned}$$

The above code has the property that the code word corresponding to each information sequence starts with a replica of the information sequence itself followed by some extra bits. Such a code is code systematic code and the extra bits following the information sequence in the code word are called the parity check bits. A necessary and sufficient condition for a code to be systematic is that the generator matrix be in the form

$$G = [I_k | P] \tag{1.13}$$

where I_k denotes a $k \times k$ identity matrix and P is a $k \times (n-k)$ binary matrix. In a systematic code, we have

$$c_i = \begin{cases} x_i, & 1 \le i \le k \\ \sum_{j=1}^{k} p_{ji} x_j, & k+1 \le i \le n \end{cases} \tag{1.14}$$

where all summations are modulo-2.

By definition a linear block code *C* is a *k* dimensional linear subspace of the *n* dimensional space. From linear algebra, it is known that in order to take all sequences of length *n* that are orthogonal to all vectors of this *k* dimensional linear subspace, the result must be an $(n-k)$ dimensional subspace called the orthogonal complement of the *k* dimensional subspace. This $(n-k)$ dimensional subspace naturally defines an $(n, n-k)$ linear code which is known as the dual of the original (n,k) code *C*. The dual code is denoted by C^T. Obviously the code words of the original code *C* and the dual code C^T are orthogonal to each other. In particular, if the generator matrix of the dual code is denoted by *H*, which is an $(n-k) \times n$ matrix, then any codeword of the original code is orthogonal to all rows of *H*; i.e.,

$$cH^T = 0 \qquad \text{for all } c \in C \tag{1.15}$$

The matrix *H*, which is the generator matrix of the dual code C^T, is called the parity check matrix of the original code *C*. Since all rows of the generator matrix are code words, it is concluded that

$$GH^T = 0 \tag{1.16}$$

In the special case of a systematic code, where

$$G = [I_k | P] \tag{1.17}$$

the parity check matrix has the following form

$$H = [-P^T | I_k] \tag{1.18}$$

Note that in the binary case $-P^T = P$.

For the above mentioned (5, 2) code

$$G = \begin{bmatrix} 10100 \\ 01111 \end{bmatrix}$$

$$I = \begin{bmatrix} 10 \\ 01 \end{bmatrix}$$

$$P = \begin{bmatrix} 100 \\ 111 \end{bmatrix}$$

Note that in the binary case – $P^T = P^T$. Therefore, it is concluded that

$$P^T = \begin{bmatrix} 11 \\ 01 \\ 01 \end{bmatrix}$$

so that

$$H = \left[\begin{array}{c|c} 11 & 100 \\ 01 & 010 \\ 01 & 001 \end{array}\right]$$

1.3.1.2 Hamming Codes

Hamming codes are a class of linear block codes with $n = 2^m - 1, k = 2^m - m - 1$, and $d_{\min} = 3$, for some integer $m \geq 2$. With this minimum distance, these codes are capable of providing error-correction capabilities for single errors. The parity check matrix for these has a very simple structure. It consists of all binary sequences of length *m* except the all zero sequence. The rate of these codes is given by

$$R_c = \frac{2^m - m - 1}{2^m - 1} \tag{1.19}$$

which is close to 1 for large value of *m*. Therefore, Hamming codes are high-rate codes with relatively small minimum distance ($d_{\min} = 3$). Minimum distance of a code is closely related to its error-correcting capabilities. Therefore, Hamming codes have limited error-correcting capability.

For example, for the (7, 4) Hamming code $m = 3$ and, therefore, *H* consists of all binary sequences of length 3 except the all zero sequence. Parity check matrix can be generated in the systematic form as [3]

$$H = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

and the generator matrix is obtained to be

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

1.3.2 Convolutional Codes

Convolutional codes are one of the most widely used channel codes in practical communication systems. These codes are developed with a separate strong mathematical structure and are primarily used for real time error correction.

Convolutional codes convert the entire data stream into one single codeword. The encoded bits depend not only on the current k input bits but also on past input bits. The main decoding strategy for convolutional codes is based on the widely used Viterbi algorithm.

As a result of the wide acceptance of convolutional codes, there have been many advances to extend and improve this basic coding scheme. This advancement resulted in two new coding schemes, namely, trellis coded modulation (TCM) and turbo codes.

TCM adds redundancy by combining coding and modulation into a single operation (as the name implies). The unique advantage of TCM is that there is no reduction in data rate or expansion in bandwidth as required by most of the other coding schemes [8].

1.3.2.1 Convolutional Encoder

Convolutional encoders add redundancy to continued stream of input data by using a linear shift register. Each set of n output bits is a linear combination of the current set of k input bits and the m bits stored in the shift registers (Figure 1.5). The total number of bits that each output depends on is called the constraint length, and is denoted by *L*, where $L = m + 1$, and the total number of states of the code is 2^m. The rate, *r*, of the convolutional encoder is the number of *k* input data bits divided by the number of *n* coded output bits, or $r = k/n$. Outputs are determined by the connections to the modulo-2 adders. These connections are called the generator sequences or the generator polynomials of the code. For the example encoder given in Figure 1.5 generator polynomials are $g_0 = \begin{pmatrix} g_0^0 & g_0^1 & g_0^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$ and $g_1 = \begin{pmatrix} g_1^0 & g_1^1 & g_1^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ or in octal form $g_0 = 5_8$ and $g_1 = 7_8$, and the outputs are given by

$$Y_k^0 = \sum_{i=0}^{m} g_0^i d_{k-i} \tag{1.20}$$

and

$$Y_k^1 = \sum_{i=0}^{m} g_1^i d_{k-i} \tag{1.21}$$

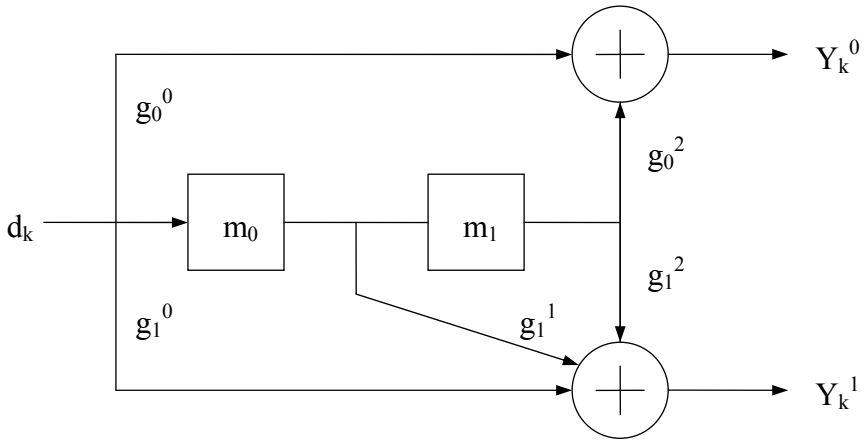


Figure 1.5 Rate 1/2 Encoder

The convolutional encoder given in Figure 1.5 is a nonrecursive Nonsystematic Convolutional (NSC) encoder, which is the most common form of convolutional encoder [5].

1.3.2.2 Convolutional Decoder

The convolutional decoder normally takes as input estimates for both the systematic bits and the coded bits along with preset information about the finite automation that was used. In this case, the estimates for the systematic bits come from the permuted results of the repetition decoder (for the first iteration, this information is null; that is, the probabilities are 0.5) and the occasional systematic estimate from the channel. These two probabilities are combined as if for a rate one-half repetition decoder:

$$p = \frac{p_1 p_2}{p_1 p_2 + (1 - p_1)\ (1 - p_2)} \tag{1.22}$$

Note that when one of the estimates is 0.5, the total information is that of the other estimate. Naturally, this formula will reappear in the repetition decoder component.

For each transmitted codeword, the estimates for the coded bits are constant; they come for the most part directly from the channel. Whenever the channel provides an estimate for a systematic bit, the information about the corresponding coded bit is set to null (i.e. probability that the coded bit was 1 is 0.5). The decoding algorithm for a convolutional code involves the belief propagation of messages containing probabilities for the state of the finite automation, sent forward and backward in time. The principle behind this is that at any point in time, the message traveling forward contains the combined information of all systematic and coded estimates before it, and the backward message contains information from later estimates. Thus, a certain input bit's extrinsic probability, taking into account all other estimates except for its own, can be found by combining the forward and backward messages with the corresponding output bit estimate [6].

1.3.2.3 The Viterbi Algorithm

Convolutional codes are widely used to encode digital data before transmission through noisy or error-prone channels. During encoding, k input bits are mapped to n output bits to give a rate k/n coded bitstream. The encoder consists of a shift register of kL stages, where L is described as the constraint length of the code. At the receiver, the bitstream can be decoded to recover the original data, correcting errors in the process. The optimum decoding method is maximum-likelihood decoding where the decoder attempts to find the closest "valid" sequence to the received bitstream. The most popular algorithm for maximum-likelihood decoding is the Viterbi algorithm. The possible received bit sequences form a "trellis" structure and the Viterbi algorithm tracks likely paths through the trellis before choosing the most likely path [4].

Encoding Process:

Convolutional encoder error-correction capabilities result from outputs that depend on past data values. Each coded bit is generated by convolving the input bit with previous uncoded bits. An example of this process is shown in Figure 1.6. The information bits are input to a shift register with taps at various points. The tap values are combined through a Boolean XOR function (the output is high if one and only one input is high) to produce output bits.

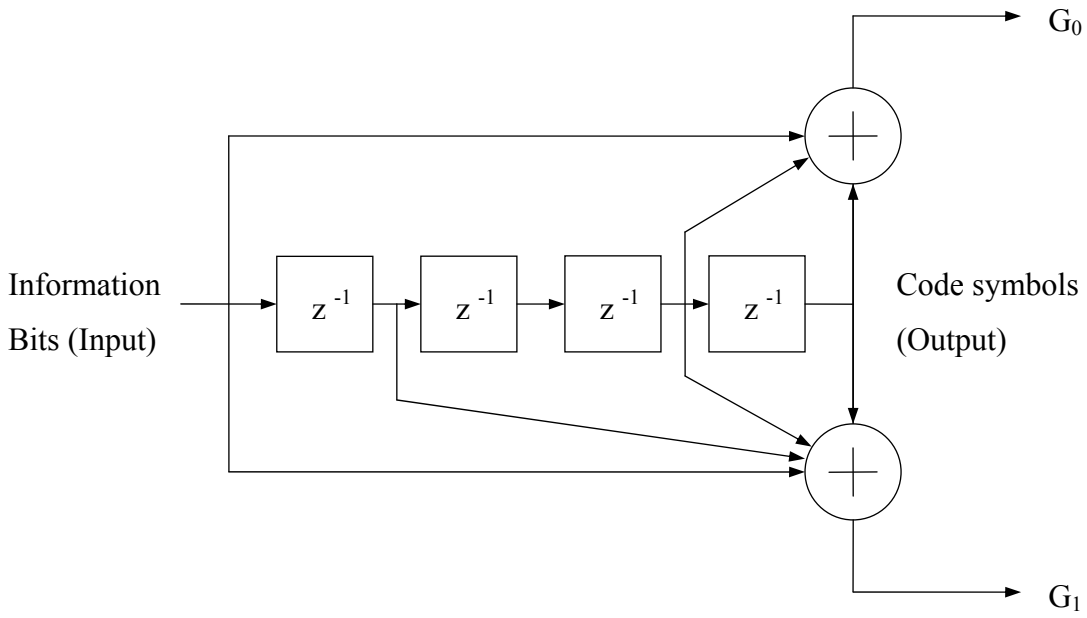


Figure 1.6 Constraint Length 5 Rate $1/2$ Convolutional Encoder

Error correction is dependent on the number of past samples that form the code symbols. The number of input bits used in the encoding process is the constraint length and is calculated as the number of unit delays plus one.

In Figure 1.6, there are four delays. The constraint length is five. The constraint length represents the total span of values used and is determined regardless of the number of taps used to form the code words. The symbol L represents the constraint length. The constraint length implies many system properties; most importantly, it indicates the number of possible delay states [4].

Coding Rate:

Another major factor influencing error correction is the coding rate, the ratio of input data bits to bits transmitted. In Figure 1.6, two bits are transmitted for each input bit for a coding rate of $1/2$.

For a rate $1/3$ system, one more XOR block produces one more output for every input bit. Although any coding rate is possible, rate $1/n$ systems are most widely used due to the efficiency of the decoding process.

The output-bit combination is described by a polynomial. The system, as shown in Figure 1.6, uses the polynomials:

$$G_0(x) = 1 + x^3 + x^4 \tag{1.23}$$

$$G_1(x) = 1 + x + x^3 + x^4 \tag{1.24}$$

Polynomial selection is important because each polynomial has different error correcting properties. Selecting polynomials that provide the highest degree of orthogonality maximizes the probability of finding the correct sequence [4].

Decoding Process:

Convolutionally encoded data is decoded through knowledge of the possible state transitions, created from the dependence of the current symbol on past data. The allowable state transitions are represented by a trellis diagram.

A trellis diagram for a $K = 3$, rate $1/2$ encoder is shown in Figure 1.7. The delay states represent the state of the encoder (the actual bits in the encoder shift register), while the path states represent the symbols that are output from the encoder. Each column of delay states indicates one symbol interval.

The number of delay states is determined by the constraint length. In this example, the constraint length is three and the number of possible states is $2^{k-1} = 2^2 = 4$. Knowledge of the delay states is very useful in data decoding, but the path states are the actual encoded and transmitted values.

The number of bits representing the path states is a function of the coding rate. In this example, two output bits are generated for every input bit, resulting in 2-bit path states. A rate $1/3$ (or $2/3$) encoder has 3-bit path states; rate $1/4$ has 4-bit path states, and so forth. Since path states represent the actual transmitted values, they correspond to constellation points, the specific magnitude and phase values used by the modulator.

The decoding process estimates the delay state sequence, based on received data symbols, to reconstruct a path through the trellis. The delay states directly represent encoded data, since the states correspond to bits in the encoder shift register.

In Figure 1.7, the most significant bit (MSB) of the delay states corresponds to the most recent input and the least significant bit (LSB) correspond to the previous input. Each input shifts the state value one bit to the right, with the new bit shifting into the MSB position. For example, if the current state is 00 and a 1 is input, the next state is 10; a 0 input produces a next state of 00.

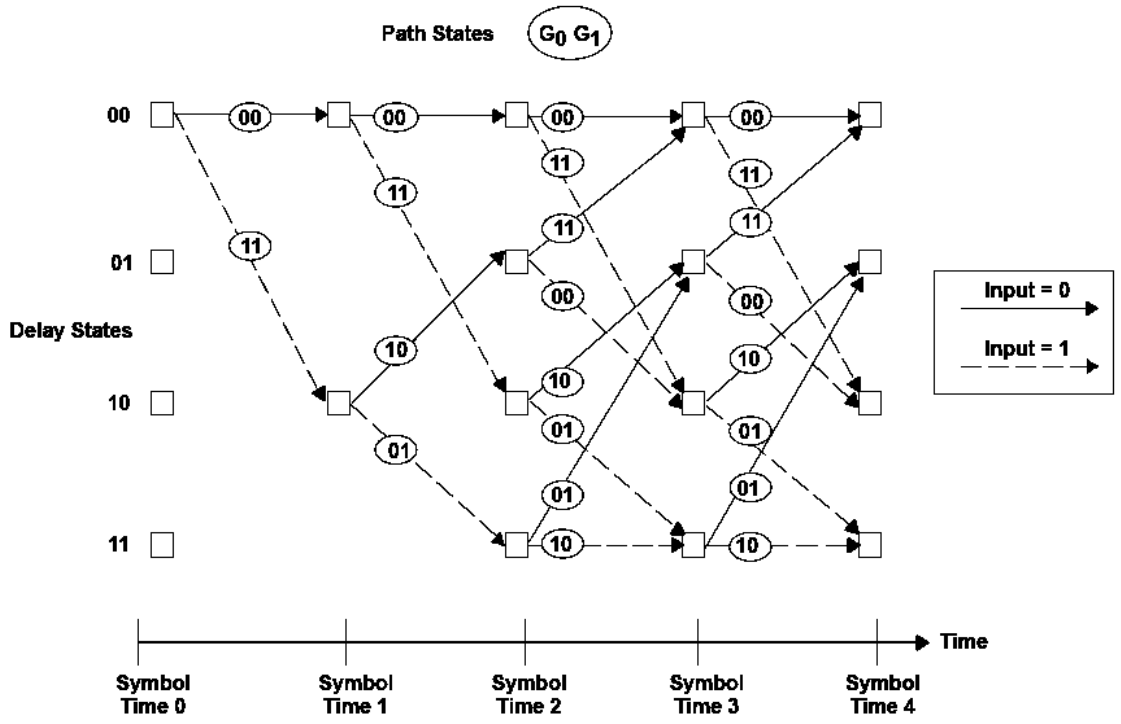


Figure 1.7 Trellis Diagram for $K = 3$, Rate $1/2$ Convolutional Encoder [4]

Systems of all constraint lengths use similar state mapping. The correspondence between data values and states allows easy data reconstruction once the path through the trellis is determined [4].

1.3.3 Turbo Codes

Recently, a near channel capacity error correcting code called turbo code was introduced. This error correcting code is able to transmit information across the channel with arbitrary low (approaching zero) bit error rate [8]. This code is a parallel concatenation of two component convolutional codes separated by a random interleaver.

It has been shown that a turbo code can achieve performance within 1 dB of channel capacity. Random coding of long block lengths may also perform close to channel capacity, but this code is very hard to decode due to the lack of code structure.

Without a doubt, the performance of a turbo code is partly due to the random interleaver used to give the turbo code a "random" appearance. However, one big advantage of a turbo code is that there is enough code structure (from the convolutional codes) to decode it efficiently [9].

There are two primary decoding strategies for turbo codes. They are based on a maximum a posteriori (MAP) algorithm and a soft output Viterbi algorithm (SOVA). Regardless of which algorithm is implemented, the turbo code decoder requires the use of two (same algorithm) component decoders that operate in an iterative manner.

1.3.3.1 Turbo Encoder and Decoder

The turbo encoder consists of two recursive systematic convolutional codes known as constituent codes (Figure 1.8). Since the turbo code is a type of linear block code, the encoding operation can be viewed as the modulo-2 matrix multiplication of an information matrix with a generator matrix. Encoder 1 encodes the input data sequence directly, while Encoder 2 encodes the data sequence permuted in time by a pseudorandom interleaver with length *N*. The encoder outputs are composed of the systematic bit d_k and parity bits $x_{p1,k}$ and $x_{p2,k}$ from the two constituent codes. In this encoding process, a very large effective constraint length is generated through interleaving and concatenation. For this reason, the conventional Viterbi decoding algorithm is not feasible for turbo decoding, and turbo decoding is typically performed in an iterative manner. In particular, each constituent code is separately decoded using the most recent decoding information from the other constituent code. Each constituent decoder computes the a posteriori log-likelihood ratio (LLR) of the systematic bits, which is given by

$$\Lambda_k = \log \frac{P(d_k = 1 | y_1, y_2, \cdots, y_N)}{P(d_k = 0 | y_1, y_2, \cdots, y_N)} \tag{1.25}$$

where $y_i (1 \le i \le N)$ is the decoder output and $p(d_k = i \mid y_1, y_2, \cdots, y_N)$ is the a posteriori probability of the bit value. This extrinsic information Λ_k can be regarded as a type of diversity in that it can refine the decoder outputs in each iteration. The decoding process continues until some stopping criterion is met. After each iteration, the data bit decision $\hat{d}_k$ is made based on the final decoder output [10]. The Figures 1.8 and 1.9 show the block diagrams of turbo encoder and turbo decoder.

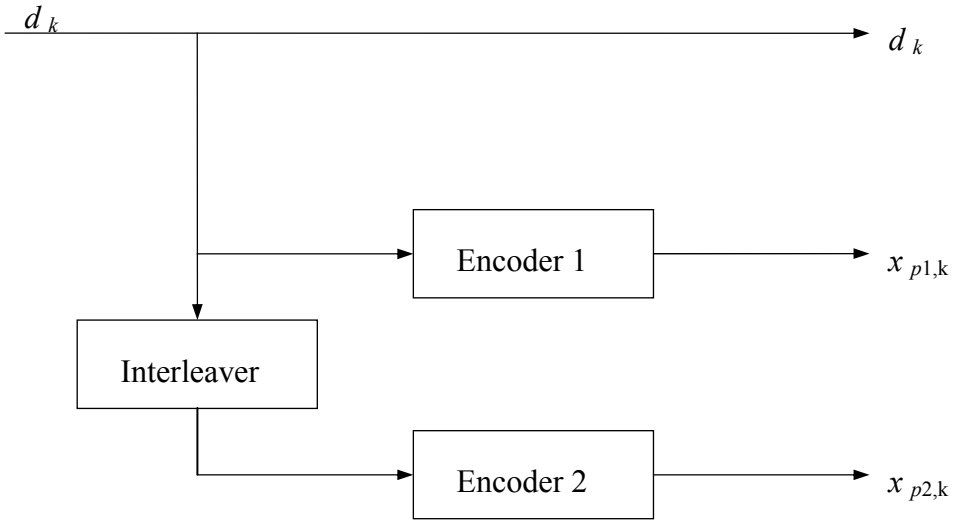


Figure 1.8 Block Diagram of the Turbo Encoder

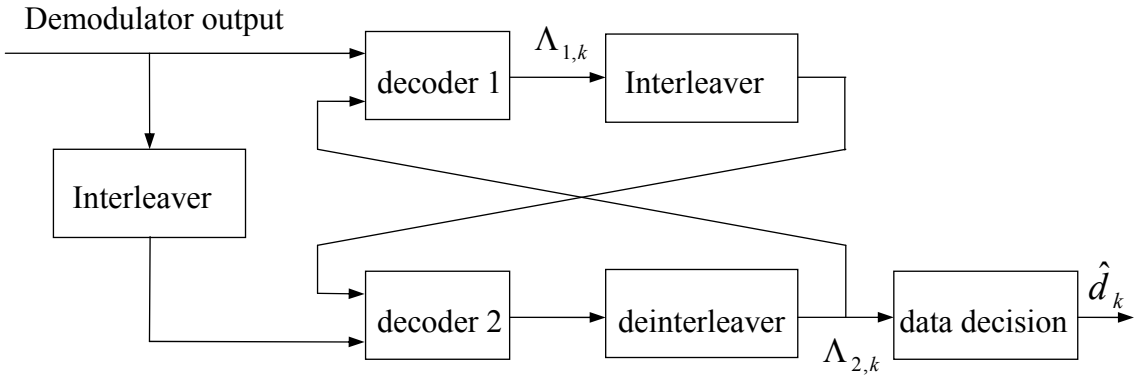


Figure 1.9 Block Diagram of the Turbo Decoder

The turbo code integrates code concatenation in a pseudorandom approach where the randomness and long block size are provided by the interleaver. The first decoder passes the extrinsic information [a part of the soft output provided by an a posteriori probability (APP)] to the next decoding stage. The term "turbo" stems from the fact that the decoder uses its processed output as a priori input in the next iteration, and thus reuses the information in the observations [10].

1.3.4 Cyclic Codes

Cyclic codes form an important sub class of linear block codes. Cyclic codes have well-defined mathematical structure, which lead to the development of very efficient

encoding and decoding circuits. Indeed, most block codes used in various applications are cyclic. This includes Hamming codes, BCH codes and Reed-Solomon codes.

A cyclic code is a linear code with the additional property that shifting a codeword cyclically produces another codeword. To restate the cyclic property formally, let vector $(x_0 \quad x_1 \quad x_2 \quad .... \quad x_{n-1})$ denote a codeword of a (n,k) linear block code.

The code is a cyclic code if the following vectors are all valid codewords as well:

$$(x_{n-1} \quad x_0 \quad x_1 \quad .... \quad x_{n-2}),\ (x_{n-2} \quad x_{n-1} \quad x_0 \quad .... \quad x_{n-3}),....,(x_1 \quad x_2 \quad x_3 \quad .... \quad x_0)$$

The cyclic property of a linear code suggests that the components of a codeword may be viewed as the coefficients of a binary polynomial of degree n- 1, i.e.

$$X = (x_0 \quad x_1 \quad x_2 \quad .... \quad x_{n-1});$$

$$X(D) = x_0 + x_1 D + x_2 D^2 + ...x_{n-1} D^{n-1};$$

For example, $X = (11001);\ X(D) = 1 + D + D^4$ where the power of D determines the position of the corresponding component in the codeword. Note that although the exponents of various terms of this polynomial are taken from the set of integers, the polynomial is essentially a binary polynomial. The coefficients are binary numbers.

This establishes a one-to-one relationship between the codeword of a cyclic code and binary polynomials of degree $n-1$ or less. These polynomials are called code polynomials. For a (n,k) cyclic code, all code polynomials contain a polynomial of a minimum degree as a factor. This polynomial, called the generator polynomial $G(D)$, possesses the following properties:

- The generator polynomial is unique in that it is the only nonzero code polynomial of minimum degree $n-k$.
- The polynomial is a factor of polynomial $D^n + 1$, i.e. $D^n + 1 = G(D)H(D)$.
- The code polynomial for each message block is formed by first forming the message polynomial, and then multiplying the message polynomial by the generator polynomial.
- The last property defines the encoding mechanism for cyclic codes. The second property stipulates a method for the construction of cyclic codes.

The algebraic structure of the cyclic codes leads to efficient encoding and decoding methods. Indeed, the encoder consists of a simple linear feedback shift register circuit with $n-k$ memory cells (flip flops). Similarly, one can exploit the elegant algebraic structure of cyclic codes to devise efficient decoding systems for error detection and correction [11].

1.4 Summary

This chapter presented an overview of a digital communication system. Linear block codes, convolutional codes, and other common error control coding are described.

The next chapter will present in detail the low density parity check (LDPC) codes, and iterative decoding algorithms that it is used for their encoding decoding process, which will be simulated by Matlab program within this thesis.

CHAPTER TWO

Low–Density Parity–Check Codes and Decoding Algorithms

2.1 Overview

This chapter presents low-density parity-check (LDPC) codes. First, a brief historical introduction of LDPC codes is given and basics of graph theory are explained. Then, LDPC code fundamentals are presented and several code design methods are discussed. Also, the details of LDPC encoding and decoding are given.

2.2 Introduction to LDPC Codes

LDPC codes were invented by Gallager [13] in 1962, but did not receive much attention until they were rediscovered independently by MacKay [16] following the invention of turbo codes. The work of Tanner [15] about 20 years after Gallager, in which he introduced a graphical representation of LDPC codes, they were rarely used until the rediscovery of MacKay [16].

An LDPC code is a linear block code which has a very sparse parity-check matrix. LDPC codes were initially constructed using regular graphs until the work of Luby *et al.* who proposed graphs with degrees that are not constant, called irregular codes [17].

2.3 Graph Theory

Graph theory is often useful for understanding the operation of codes and how well they perform. A graph $G = (V, E)$ (Figure 2.1(a)) consists of two sets, a finite set V of points called vertices or nodes and a finite set E of lines called edges. An edge is incident on a node if it is connected to it. Each edge connects two nodes and makes them adjacent. The set of all nodes that are adjacent to a particular node is its neighbors. The number of edges that are connected to a node is called the local degree of the node. A cycle of length v, called a v cycle, in a graph is a closed path consisting of v edges. The graph in Figure 2.1(a) has two three cycles, two four cycles and a five cycle. A tree (Figure 2.1(b)) is a connected graph with no cycles (a graph is connected if there is a path from v_i to v_j, for all v_i and v_j in V). Each node has one or more children below and at most one parent above it. Nodes with no children are called leaf nodes and the node with no parent is the root node.

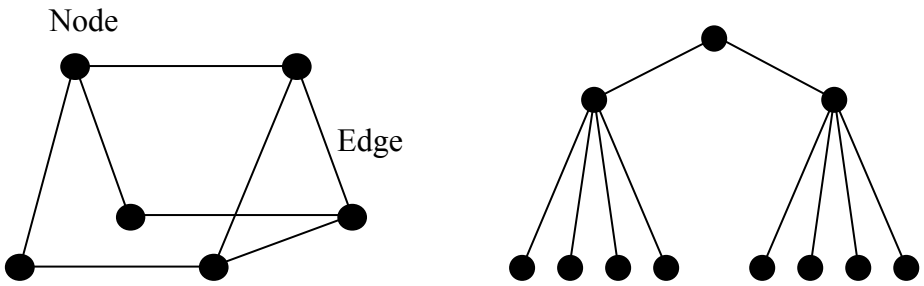


Figure 2.1 A Graph and a Tree

An LDPC code can be represented by a bipartite graph also called Tanner graph. In a Tanner graph the nodes of the graph are separated into two distinctive sets. The elements of these sets, called variable and check nodes, are connected by edges.

Variable nodes are also called symbol nodes (or bit nodes in binary case) and check nodes are also called function nodes. In some literature, variable and check nodes are referred to as left nodes and right nodes, respectively.

2.4 LDPC

LDPC code is a linear block code for which the $M \times N$ parity-check matrix, *H*, has a low density of ones. A regular LDPC code contains exactly w_c ones in each column (i.e. column weight $= w_c$) and exactly w_r ones in each row (i.e. row weight $= w_r$), where w_r is related to w_c by $w_r M = w_c N$. Also, $w_c << M$ and $w_r << N$. If *H* is full rank, the code rate *R* can also be written in terms of the row and column weights as $R = 1 - w_c / w_r$. If the number of ones in each column or row is not constant, then the code is an irregular LDPC code [14].

There is an edge connecting the check node j to variable node j if the value in row i and column i of matrix *H* is a one. There are a total of *N* variable nodes and *M* check nodes. For a particular code, degree distribution polynomials, $\lambda(x)$ and $\rho(x)$, given by

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1} \tag{2.1}$$

and

$$\rho(x) = \sum_{i=2}^{d_c} \rho_i x^{i-1} \tag{2.2}$$

represent the distributions of variable and check node degrees, respectively. Here, d_v is the maximum variable node degree and d_c is the maximum check node degree. The λ_i represents the fraction of edges emanating from variable nodes of degree i and ρ_i represents the fraction of edges emanating from check nodes of degree i. The λ_i and ρ_i can be converted into node perspective equivalents $\hat{\lambda}_i$ and $\hat{\rho}_i$ by defining

$$\hat{\lambda}_i = \frac{\lambda_i}{i\sum_{j=2}^{d_v} \lambda_j / j} \tag{2.3}$$

and

$$\hat{\rho}_i = \frac{\rho_i}{i\sum_{j=2}^{d_c} \rho_j / j} \tag{2.4}$$

Now $\hat{\lambda}_i$ is the fraction of variable nodes with degree i and $\hat{\rho}_i$ is the fraction of check nodes with degree i. Assuming check equations are linearly independent, the rate *R* of an irregular LDPC code becomes

$$R(\lambda, \rho) = \frac{N - M}{N} = 1 - \frac{\int_0^1 \rho(x)\ dx}{\int_0^1 \lambda(x)\ dx} \tag{2.5}$$

For regular codes the degree of each variable node is exactly d_v and the degree of each check node is exactly d_c. Therefore, the degree distribution pairs for regular codes are $\lambda(x) = x^{d_v - 1}$ and $\rho(x) = x^{d_c - 1}$.

The number of variable nodes of degree i is

$$N_v(i) = N \frac{\lambda_i / i}{\sum_{j=2}^{d_v} \lambda_j / j} = N \frac{\lambda_i / i}{\int_0^1 \lambda(x)\ dx} \tag{2.6}$$

And the number of check nodes of degree i is

$$N_c(i) = M \frac{\rho_i / i}{\sum_{j=2}^{d_c} \rho_j / j} = M \frac{\rho_i / i}{\int_0^1 \lambda(x)\ dx} \tag{2.7}$$

the total number of edges emerging from all variable nodes, *E*, is equal to

$$E = N \frac{1}{\int_0^1 \lambda(x)\ dx} \tag{2.8}$$

Consider a rate $1/2(K=5, N=10)$ regular LDPC code with the following H matrix:

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Here, $w_c = d_v = 2$, $w_r = d_c = 4$ and $\lambda(x) = x$ and $\rho(x) = x^3$. The bipartite graph representation of this code is given in Figure 2.2.

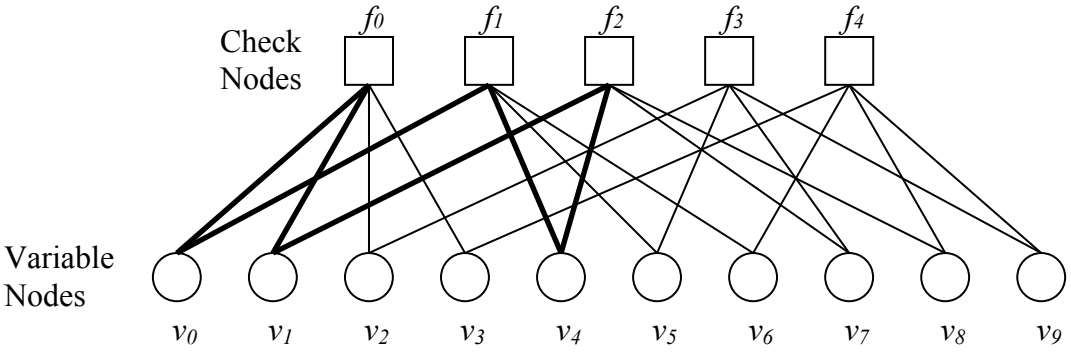


Figure 2.2 Bipartite Graph Representation of $(K=5, N=10)$ Regular LDPC Code

The bipartite graph in Figure 2.2 consists of a six cycle, as shown in bold edges. The shortest possible cycle in a bipartite graph is a four cycle. Short cycles degrade the performance of LDPC codes, so they should be avoided if possible [14].

2.5 Code Design

The code properties described in previous section determine the dimensions of the parity-check matrix as well as the column and row weights, but they do not indicate how the matrix should be constructed. There are various code design methods in the literature that address this issue. Depending on the application, the code is designed to meet certain criteria, including low error floors, efficient encoding and decoding, or

near capacity performance. Trade-offs exists among them and it is difficult to design LDPC codes that meet all of the criteria. Different code design approaches in the literature fall into two categories, random and deterministic, although there exists codes that carry the characteristics of both. The following sections describe some different LDPC codes, starting with Gallager's initial definition of the codes [13].

2.5.1 Gallager Codes

Gallager expressed the original LDPC codes as regular. The parity-check matrix has N columns with j ones in each column, k ones in each row and zeros elsewhere. This forms a matrix having Nj/k rows and a code having a rate $R \geq 1 - j/k$. The matrix is also divided into j submatrices of column weight one. For the first submatrix i the ith row contains ones in columns $(i-1)k+1$ to ik The other submatrices are simply column permutations of the first. This procedure does not guarantee the absence of four cycles but Gallager showed that the ensemble of such codes should have excellent distance properties if $j \geq 3$ and $k > j$. Figure 2.3 shows a $N = 20$, $j = 3$, $k = 4$ Gallager code [13].

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1

Figure 2.3 Example of a Parity-Check Matrix for a $N = 20$, $j = 3$, $k = 4$ Gallager Code.

2.5.2 MacKay Codes

MacKay rediscovered Gallager's work in [16] and showed that these codes could also achieve near capacity performance like the well known Turbo codes [9]. He extended the code construction method of Gallager and provided six methods of generating code ensembles.

1. Generate H by starting from all-zero matrix and randomly changing d_v not necessarily distinct values in each column.
2. Generate H as in 1 but with columns having exactly weight d_v.
3. Generate H as in 2 and also with uniform weight per row.
4. Generate H as in 3 and constrain it so that no two columns overlap in more than one position. Also named 'Construction 1A'.
5. Further constraint H so that its graph has large girth (shortest cycle length). This is achieved by eliminating the short cycles. Also named 'Construction 1B'.
6. Generate H as in 5 and partition H in the form $H = [H_1 H_2]$ so that H_2 is invertible or H is full rank.

MacKay suggests that encoding be performed by using the generator matrix G obtained through Gaussian elimination from H. This method is not efficient because even though the parity-check matrix is sparse the generator matrix is generally not. Therefore, the encoding complexity of long block length codes generated in this manner would be high.

2.6 Encoding

Even though the encoding process is defined by the generator matrix G, it is the parity-check matrix H that is used in constructing and decoding LDPC codes.

The encoder is defined by a $K \times N$ generator matrix, G, which maps each K symbol data block, u, into N symbol codeword, c, using the form

$$uG = c \tag{2.9}$$

The receiver (decoder) is defined by an $M \times N$ parity-check matrix, H, which is related to the generator matrix by

$$GH^T = 0 \tag{2.10}$$

The decoder verifies a received codeword is valid by performing the operation

$$cH^T = uGH^T$$

$$cH^T = 0 \tag{2.11}$$

The decoding is simplified due to the sparse nature of *H*. The encoding, on the other hand, is more complex since *G* is not necessarily sparse [14]. The following subsections describe a few ways of encoding using *H*, without directly computing *G*.

2.6.1 Generic Encoding

Let $H = [x_1 x_2]$ be the $M \times N$ parity-check matrix with the submatrices x_1 and x_2 having the dimensions $M \times K$ and $M \times M$, respectively. Assume, the codeword c is in the systematic form $c = [u\, p]$, where u is the *K* bit message and *p* is the *M* bit parity. As defined by Equation 2.11, a valid codeword should satisfy *M* parity checks of *H*. i.e.

$$cH^T = 0$$

$$[u\, p]\ \ [x_1^T\, x_2^T] = 0$$

$$ux_1^T + px_2^T = 0$$

$$px_2^T = -ux_1^T$$

$$p = -ux_1^T x_2^{-T} \tag{2.12}$$

Since the message u and the parity-check matrix *H* are binary, the negative sign in Equation 2.12 can be removed. This equation shows that *p* exists if and only if x_2 is invertible i.e. if *H* is full rank or the rank of *H* is equal to *M*. If it is not, then the linearly dependent rows of *H* can be deleted to obtain a full rank *H*, at the expense of obtaining a higher code rate. If x_2 is still not invertible, columns of *H* can also be reordered. Note that with this method a generator matrix is never used.

Another method also exists which does not necessitate a matrix inversion. Instead, x_2 submatrix is partitioned into a lower triangular matrix *L* and an upper triangular matrix *U* using LU decomposition.

$$p = ux_1^T x_2^{-T}$$

$$p = zx_2^{-T}$$

$$z = zx_2^T = pL^T U^T$$

$$z = yU^T \tag{2.13}$$

$$y = pL^T \tag{2.14}$$

First, y is obtained from Equation 2.13 by forward substitution. Then, y is used in Equation 2.14 to obtain p using backward substitution.

2.7 Decoding Algorithms

The algorithm used to decode LDPC codes was discovered independently several times and as a matter of fact comes under different names. The most common ones are the belief propagation algorithm, the message passing algorithm and the sum-product algorithm.

The following sections describe the probability version of the decoding algorithm.

2.7.1 Message-Passing Decoding Algorithms

This algorithm is iterative; in each iteration it calculates the a posteriori (extrinsic) probability that a given bit in the transmitted codeword $c = [c_0, c_1, ..., c_{N-1}]$ equals one, given the received word $y = [y_0, y_1, ..., y_{N-1}]$. i.e.

$$P_r(c_i = 1 | y_i) \tag{2.15}$$

Note that, as stated earlier, every codeword c must satisfy a set of equations defined by

$$cH^T = 0 \tag{2.16}$$

Therefore, the event S_i, that the code bit c_i satisfies the equations defined by the parity-check matrix H, should also be included in the probability

$$P_r(c_i = 1 | y_i, S_i) \tag{2.17}$$

Iterative computation of the probabilities can be best visualized by the Tanner graph drawn based on the parity-check matrix. The edges on the graph can be thought of as the paths for the extrinsic probabilities, called ‘messages’, to travel between each set of nodes of the graph. Figure 2.4 is a subgraph showing the variable node v_0 connected to three check nodes. The directional arrows connecting the four nodes indicate that the extrinsic information being passed from node v_0 to node f_2 comprises of the information node that v_0 receives from the channel with the received bit y_0 as well as the extrinsic information node v_0 received from check nodes f_0 and f_1 on the previous half iteration.

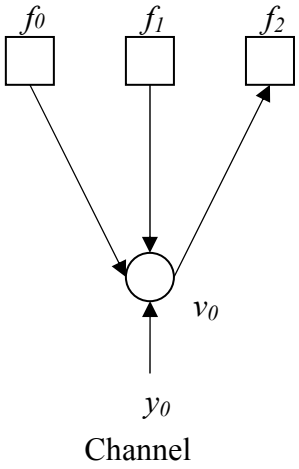


Figure 2.4 Graph Showing the Message Send From a Variable Node to a Check Node

Figure 2.4 Subgraph showing the message send from a variable node to a check node. Similarly, Figure 2.5 shows the subgraph for the node f_0 involving the information it is sending to variable node v_4. The information sent to variable node v_4 from check node f_0 is the information node f_0 had received on the previous half iteration from nodes v_0, v_1 and v_2.

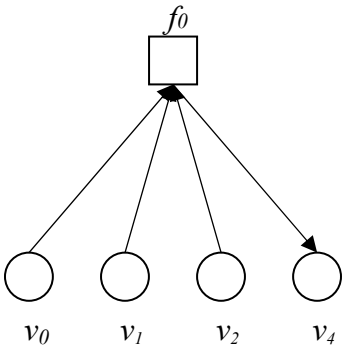


Figure 2.5 Graph Showing the Message Send From a Check Node to a Variable Node

The same procedure is repeated for all nodes connected to each check and variable node, where each node sends all extrinsic information it has received to its connecting nodes, excluding the information the receiving node already has. One iteration consists of traversal of information along all the edges from variable to check

nodes and back. Decoding is stopped after a specified number iterations is reached, or before that if a valid estimated codeword, $\hat{c}$, is found (for $\hat{c}$ to be valid, $\hat{c}H^T = 0$ should be satisfied) [14].

2.7.2 Probability Decoding Algorithm

Before deriving the probability decoding algorithm, the following notation is introduced:

- R_j : The set of column locations of the ones in the j th row.
- $R_{j\backslash i}$: The set of column locations of the ones in the j th row, excluding location i .
- C_i : The set of row locations of the ones in the i th column.
- $C_{i\backslash j}$: The set of row locations of the ones in the i th column, excluding location j .
- $P_i : P_r\left(c_i = 1 \middle| y_i\right)$
- $P_{ij} : P_r\left(c_{ij} = 1 \middle| y_{ij}\right)$
- S_i : Event that the check equations involving c_i are satisfied.
- $q_{ij}(b) : P_r\left(c_i = b \middle| S_i, y_i\right), b \in \{0,1\}$ = Probability that $c_i = b$ given extrinsic information from all neighboring nodes, except node f_j .
- $r_{ij}(b) : P_r\left(S_i \middle| c_i = b, y\right), b \in \{0,1\}$ = Probability of the j th check equation being satisfied given $c_i = b$ and extrinsic information from all variable nodes, except node v_i .
- $Q_{ij}(b)$ = Probability that $c_i = b$ given extrinsic information from all check nodes.

The algorithm is initialized with $q_{ij}(b)$, for all i and j for which $H_{ij} = 1$, where $q_{ij}(b)$ is given by

$$q_{ij}(b) = P_r\left(c_i = b \middle| y_i\right) \quad b \in \{0,1\} \tag{2.18}$$

Note that $P_r\left(c_i = 0 \middle| y_i\right) = 1 - P_i$ and $P_r\left(c_i = 1 \middle| y_i\right) = P_i$. The received symbol corresponding to c_i is y_i. For the binary input additive white Gaussian noise (AWGN) channel, $P_r\left(c_i = b \middle| y_i\right)$ is calculated by the technique discussed below.

Let $y_i = x_i + n_i$, where n_i is zero mean Gaussian noise with variance σ^2, $x_i = 1 - 2c_i$ and $P_r(x_i = +1) = P_r(x_i = -1) = 1/2$. Using Bayes' rule:

$$\begin{aligned} p(x_i = x|y) &= \frac{p(y|x_i = x)p(x_i = x)}{p(y)} \\ &= \frac{e^{-(y-x)^2/2\sigma^2}(1/2)}{e^{-(y-1)^2/2\sigma^2}(1/2) + e^{-(y+1)^2/2\sigma^2}(1/2)} \\ &= \frac{e^{xy}}{e^{y/\sigma^2} + e^{-y/\sigma^2}} \\ &= \frac{1}{e^{y(1-x)/\sigma^2} + e^{y(1+x)/\sigma^2}} \\ &= \frac{1}{1 + e^{-2yx/\sigma^2}} \end{aligned} \tag{2.19}$$

On the last step, one of the terms in the denominator is going to be one no matter what the value of x is. In order to derive $P_r(S_i|c_i = b, y)$, first note the following Lemma by Gallager [13].

For a set of M binary numbers $a = (a_1, ..., a_M)$, the probability that a contains an even number of ones is

$$\frac{1}{2} + \frac{1}{2}\prod_{k=1}^{M}(1 - 2p_k) \tag{2.20}$$

where $P_r(a_k = 1) = p_k$. The probability that it contains an odd number of ones is

$$\frac{1}{2} - \frac{1}{2}\prod_{k=1}^{M}(1 - 2p_k) \tag{2.21}$$

Using this Lemma, given $c_i = 1$, in order for event S_i to be valid, the probability that the other set of bits that are in the same check equation contain an odd number of ones is

$$\frac{1}{2} - \frac{1}{2}\prod_{i' \in R_{j \setminus i}}(1 - 2p_{i'j}) \tag{2.22}$$

If $c_i = 0$, then the probability that the other bits contain an even number of ones is

$$\frac{1}{2} + \frac{1}{2}\prod_{i' \in R_{j \setminus i}}(1 - 2p_{i'j}) \tag{2.23}$$

The probability that all the check equations involving c_i are satisfied is the product of the individual probabilities. Thus, using Equations 2.22 and 2.23, $P_r(S_i|c_i = b, y)$ corresponds to

$$P_r(S_i|c_i = 0, y) = \prod_{j \in c_i} \left(\frac{1}{2} + \frac{1}{2} \prod_{i' \in Rj \backslash i} (1 - 2p_{i'j}) \right) \tag{2.24}$$

and

$$P_r(S_i|c_i = 1, y) = \prod_{j \in c_i} \left(\frac{1}{2} - \frac{1}{2} \prod_{i' \in Rj \backslash i} (1 - 2p_{i'j}) \right) \tag{2.25}$$

The a posteriori probability (APP) for c_i, given the received codeword y and the event S_i is

$$\begin{aligned} P_r(c_i = b|y, S_i) &= \frac{P_r(y, S_i|c_i = b) P_r(c_i = b)}{P_r(y, S_i)} \\ &= \frac{P_r(S_i|c_i = b) P_r(c_i = b)}{P_r(S_i)} \end{aligned} \tag{2.26}$$

For $c_i = 0$ and $c_i = 1$, this simplifies to

$$\begin{aligned} P_r(c_i = 0|y, S_i) &= \frac{(1 - p_i) P_r(S_i|c_i = 0, y)}{P_r(S_i)} \\ &= (1 - p_i) \prod_{j \in c_i} \left(\frac{1}{2} + \frac{1}{2} \prod_{i \in R_{j \backslash i}} (1 - 2p_{i'j}) \right) \end{aligned} \tag{2.27}$$

and

$$\begin{aligned} P_r(c_i = 1|y, S_i) &= \frac{p_i P_r(S_i|c_i = 1, y)}{P_r(S_i)} \\ &= p_i \prod_{j \in c_i} \left(\frac{1}{2} - \frac{1}{2} \prod_{i \in R_{j \backslash i}} (1 - 2p_{i'j}) \right) \end{aligned} \tag{2.28}$$

With all the notation defined, the decoding algorithm can be summarized as follows:

For all i and j for which $H_{ij} = 1$:

1. Initialize

   $q_{ij}(0) = P_r(x_i = 1|y_i)$

   $q_{ij}(1) = P_r(x_i = -1|y_i)$

2. Set the number of iterations to 1. Loop through the denoted steps below until either the number of iterations equals a specified number or $\hat{c}H^T = 0$ ($\hat{c}$ defined below). Then go back to step 1 with another received codeword y.

   - $r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in R_{j \setminus i}} (1 - 2q_{ij'}(1))$

     $r_{ji}(1) = 1 - r_{ji}(0)$

   - $q_{ij}(0) = K_{ij}(1 - p_i)\prod_{j' c_{j \setminus i}} r_{j'i}(0)$

     $q_{ij}(1) = K_{ij} p_i \prod_{j' c_{i \setminus j}} r_{j'i}(1)$

     where $q_{ij}(0)$ and $q_{ij}(1)$ are scaled by K_{ij} so that $q_{ij}(0) + q_{ij}(1) = 1$.

     For all $i = 1,..., N$ do:

   - $Q_i(0) = K_i(1 - p_i)\prod_{j \in c_i} r_{ji}(0)$

     $Q_i(1) = K_i p_i \prod_{j \in c_i} r_{ji}(1)$

     where $Q_i(0)$ and $Q_i(1)$ are scaled by K_i so that $Q_i(0) + Q_i(1) = 1$.

   - If $Q_i(1)$ is greater than 0.5, then $\hat{c}_i = 1$. Otherwise, set $\hat{c}_i = 0$. Here $\hat{c}_i$ is the estimate of c_i.

2.7.3 Logarithmic Probability Decoding Algorithm

After many iterations of the probability decoding algorithm, the multiplications involved might produce precision errors. Therefore, a logarithmic version of this algorithm has been proposed [14].

The logarithmic decoding algorithm can be summarized as follows.

For all i and j for which $H_{ij} = 1$:

1. Initialize

   $L(q_{ij}) = L(c_i)$

2. Set the number of iterations to 1. Loop through the denoted steps below until either the number of iterations equals a specified number or $\hat{c}H^T = 0$ whichever comes first ($\hat{c}$ defined below). Then go back to step 1 with another received codeword y.

- $L(r_{ji}) = \left(\prod_{i' \in R_{j \setminus i}} \alpha_{j \setminus i} \right) \phi \left(\sum_{i' \in R_{j \setminus i}} \phi(\beta_{i'j}) \right)$

  Where

  $\alpha_{ij} = \text{sign}(L(q_{ij}))$

  $\beta_{ij} = |L(q_{ij})|$

  $\phi(x) = -\log(\tanh(x/2)) = \log \frac{e^x + 1}{e^x - 1}$

- $L(q_{ij}) = L(c_i) + \sum_{j' \in i \setminus j} L(r_{j'i})$

  For all $i = 1...N$ do:

- $L(Q_i) = L(c_i) + \sum_{j \in c_i} L(r_{ji})$

- If $L(Q_i)$ is smaller than 0, then set $\hat{c}_i = 1$. Otherwise, set $\hat{c}_i = 0$.

2.7.4 Other Decoding Algorithms

Besides the logarithmic decoding, a few other reduced complexity decoders have been designed. Although not the focus of this research, two are briefly summarized below.

Min-Sum Decoding Algorithm:

The min-sum decoding algorithm [18] is essentially same as the logarithmic decoding algorithm with only a small complexity reducing change. It uses the fact that $\phi(x)$ is maximum when x is minimum to make the following simplification

$$\phi \left(\sum_{i' \in R_{j \setminus i}} \phi(\beta_{i'j}) \right) = \phi \left(\phi \left(\min_{i' \in R_{j \setminus i}} \beta_{i'j} \right) \right) = \min_{i' \in R_{j \setminus i}} \beta_{i'j} \tag{2.29}$$

Note that $\phi(\phi(x)) = x$. Incorporating this into the logarithmic decoding algorithm, set 2 can be simplified as

$$L(r_{ji}) = \left(\prod_{i' \in R_{j \setminus i}} \alpha_{i'j} \right) \min_{i' \in_{j \setminus i}} \beta_{i'j}$$

Normalized Min-Sum Algorithm:

In the normalized min-sum algorithm, all the log- likelihood ratios in the min-sum algorithm are normalized by $2\sigma^2$. This eliminates the need to know the noise variance σ^2.

2.8 LDPC Code Performance in AWGN Channel Model

The channel, which may include other effects besides those of the physical medium, invariably distorts the transmitted modulation signals in random and non-random ways. It is often characterized by the way it distorts the transmitted signals.

The codeword is passed through a communication channel, which introduces random errors into the codeword. Even though the errors are random, statistical properties of the channel are known, which enables the receiver to predict the original input to the channel. In order to test the LDPC system used in this thesis, the additive white Gaussian channel was used. For each bit in the codeword, the Gaussian channel outputs a Gaussian random variable of a certain known standard deviation. If the bit going in the channel is one, the output has mean negative one, and if the bit going in is zero, the output has mean one. The receiver knows the standard deviation of the channel used. Therefore, for each bit received, the receiver is able to deduce a probability for the original bit to be one given the output from the channel for that bit. In the additive white Gaussian noise channel model, zero mean noise having a Gaussian distribution is added to the signal, as shown in Figure 2.6.

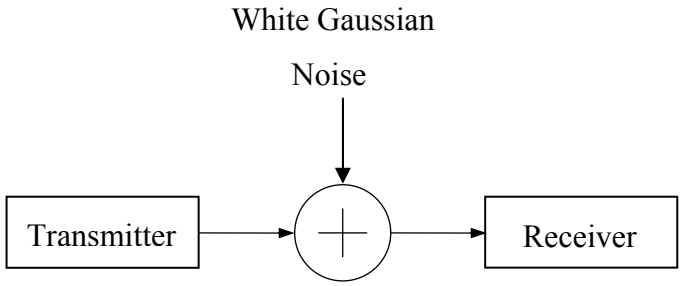


Figure 2.6 The Gaussian Noise Channel

Figure 2.7 shows the performance of the LDPC system given in chapter four of a rate 1/2 LDPC code in AWGN channel. Binary phase-shift keying (BPSK) modulation is assumed and the block size is 512 bits. It is compared to the probability of error for a binary antipodal signal. It is shown that adding LDPC codes to the system significantly improves the performance. For LDPC codes this improvement is affected by the block size.

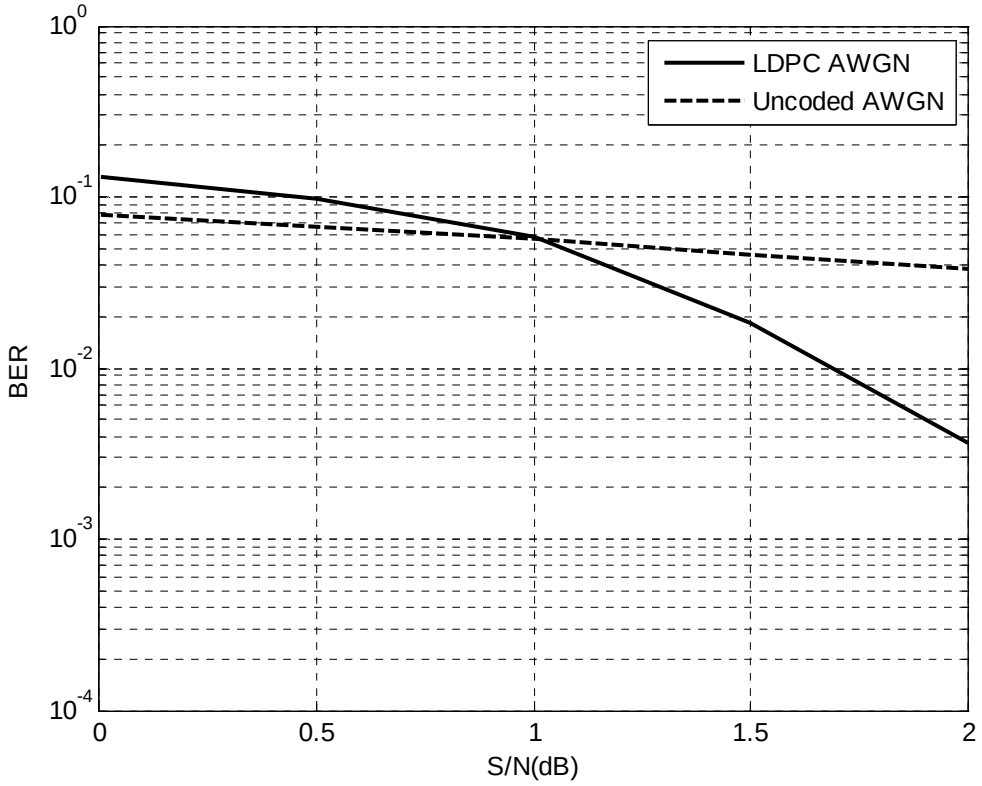


Figure 2.7 Performance of LDPC Code in AWGN Channel

The standard way to report the standard deviation of the white Gaussian channel is through the signal-to-noise ratio (SNR). Symbolicaly, this is written E_b/N_0, and if ± 1 signaling is used, it is defined to be

$$\frac{E_b}{N_0} = \frac{1}{2R\sigma^2} \tag{2.30}$$

where R is the rate of the encoder used.

SNR is usualy recorded in decibels. That is $10\log_{10}(E_b/N_0)$ dB. For exampel assume 1.5 dB, that means should set E_b/N_0 so that $10\log_{10}(E_b/N_0)=1.5$ that is

$$\frac{E_b}{N_0} = 10^{1.5/10} = 1.4125.$$

Then, by reversing this formula again to solve for σ, wich is that case is $\sigma = 0.8414$. So, when E_b/N_0 goes up, σ goes down.

2.9 Summary

This chapter presented low-density parity-check codes. After giving a brief background on LDPC code graph theory, a few code design methods were discussed, along with different methods of encoding and iterative decoding. The next chapter gives the details of image restoration techniques in spatial domain, and chapter four shows their results to be compared with LDPC codes.

CHAPTER THREE

Image Restoration Techniques

3.1 Overview

The principal objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application.

Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term spatial domain refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. Frequency domain processing techniques are based on modifying the Fourier transform of an image [20].

As in image enhancement, the ultimate goal of restoration techniques is to improve an image in some predefined sense. Although there are areas of overlap, image enhancement is largely a subjective process, while image restoration is for the most part an objective process. Restoration attempts to reconstruct or recover an image that has been degraded by using a priori knowledge of the degradation phenomenon. Thus restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image [19].

The material developed in this chapter is strictly introductory, considering the restoration problem only from the point where a degraded, digital image is given. Some restoration techniques are best formulated in the spatial domain, while others are better suited for the frequency domain. For example, spatial processing is applicable when the only degradation is additive noise. On the other hand, degradations such as image blur are difficult to approach in the spatial domain using small masks. In this case, frequency domain filters based on various criteria of optimality are the approaches of choice [19].

Since the only degradation considered in this thesis is additive white Gaussian noise, this chapter focuses only on spatial domain filtering processes.

3.2 Background

As indicated previously, the term spatial domain refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on these pixels. Spatial domain processes will be denoted by the expression

$$g(x,y) = T[f(x,y)] \tag{3.1}$$

where $f(x, y)$ is the input image, $g(x, y)$ is the processed image, and T is an operator on f, defined over some neighborhood of (x, y). In addition, T can operate on a set of input images, such as performing the pixel-by-pixel sum of K images for noise reduction. The principal approach in defining a neighborhood about a point (x, y) is to use a square or rectangular subimage area centered at (x, y), as Figure 3.1 shows. The center of the subimage is moved from pixel to pixel starting at the top left corner. The operator T is applied at each location (x, y) to yield the output, g, at that location. The process utilizes only the pixels in the area of the image spanned by the neighborhood. Although other neighborhood shapes, such as approximations to a circle, sometimes are used, square and rectangular arrays are by far the most predominant because of their ease of implementation [20].

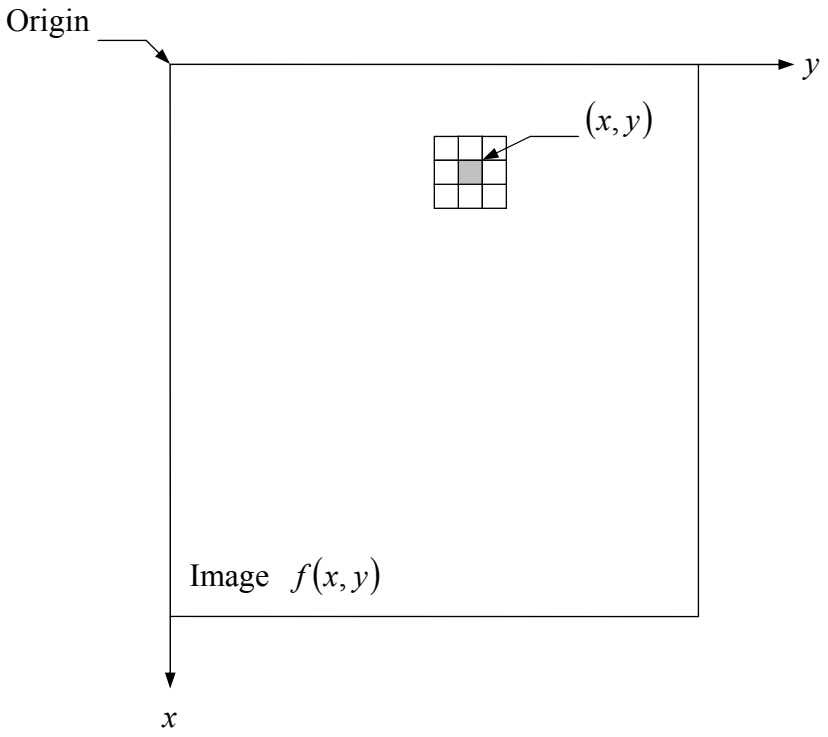


Figure 3.1 A 3×3 Neighborhood About a Point (x, y) in an Image

The simplest form of T is when the neighborhood is of size 1×1 (that is, a single pixel). In this case, g depends only on the value of f at (x, y), and T becomes a gray-level (also called an intensity or mapping) transformation function of the form

$$s = T(r) \tag{3.2}$$

where, for simplicity in notation, r and s are variables denoting, respectively, the gray level of $f(x, y)$ and $g(x, y)$ at any point (x, y). For example, if $T(r)$ has the form shown in Figure 3.2(a), the effect of this transformation would be to produce an image of higher contrast than the original by darkening the levels below m and brightening the levels above m in the original image. In this technique, known as contrast stretching, the values of r below m are compressed by the transformation function into a narrow range of s, toward black. The opposite effect takes place for values of r above m. In the limiting case shown in Figure 3.2(b), $T(r)$ produces a two-level (binary) image. A mapping of this form is called a threshold function. Some fairly simple, yet powerful, processing approaches can be formulated with gray-level transformations. Because enhancement at any point in an image depends only on the gray level at that point, techniques in this category often are referred to as point processing.

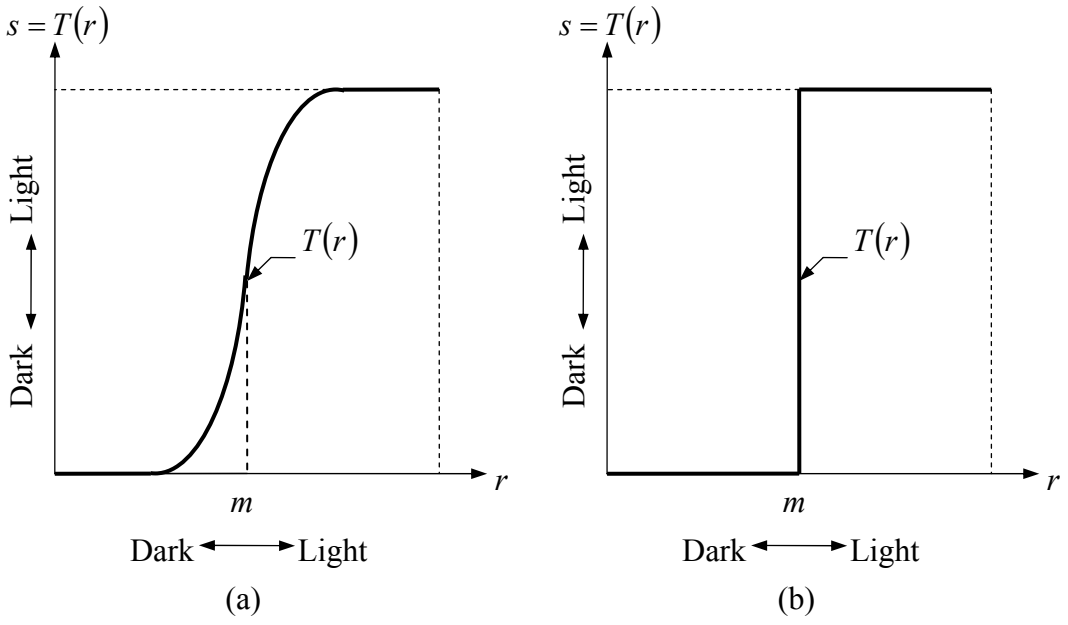


Figure 3.2 Gray level Transformation Functions for Contrast Enhancement
(a) Contrast Stretching and (b) Threshold Function

Larger neighborhoods allow considerably more flexibility. The general approach is to use a function of the values of f in a predefined neighborhood of (x, y) to determine the value of g at (x, y). One of the principal approaches in this formulation is based on

the use of so-called masks (also referred to as filters, kernels, templates, or windows). Basically, a mask is a small (say, 3×3) 2-D array, such as the one shown in Figure 3.1, in which the values of the mask coefficients determine the nature of the process, such as image sharpening. Enhancement techniques based on this type of approach often are referred to as mask processing or filtering [20].

3.3 Spatial Filtering

As mentioned in Section 3.2, some neighborhood operations work with the values of the image pixels in the neighborhood and the corresponding values of a subimage that has the same dimensions as the neighborhood. The subimage is called a filter, mask, kernel, template, or window, with the first three terms being the most prevalent terminology. The values in a filter subimage are referred to as coefficients, rather than pixels.

The concept of filtering has its roots in the use of the Fourier transform for signal processing in the so-called frequency domain. The term spatial filtering is used to differentiate this type of process from the more traditional frequency domain filtering.

The mechanics of spatial filtering are illustrated in Figure 3.3. The process consists simply of moving the filter mask from point to point in an image. At each point (x, y), the response of the filter at that point is calculated using a predefined relationship. For linear spatial filtering, the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask. For the 3×3 mask shown in Figure 3.3, the result (or response), R, of linear filtering with the filter mask at a point (x, y) in the image is

$$R = w(-1,-1)f(x-1, y-1) + w(-1,0)f(x-1, y) + \cdots$$
$$+ w(0,0)f(x, y) + \cdots + w(1,0)f(x+1, y) + w(1,1)f(x+1, y+1),$$

which we see is the sum of products of the mask coefficients with the corresponding pixels directly under the mask. Note in particular that the coefficient w (0, 0) coincides with image value $f(x, y)$, indicating that the mask is centered at (x, y) when the computation of the sum of products takes place. For a mask of size $m \times n$, it is assumed that $m = 2a + 1$ and $n = 2b + 1$, where a and b are nonnegative integers. The following discussion will be on masks of odd sizes, with the smallest meaningful size being 3×3.

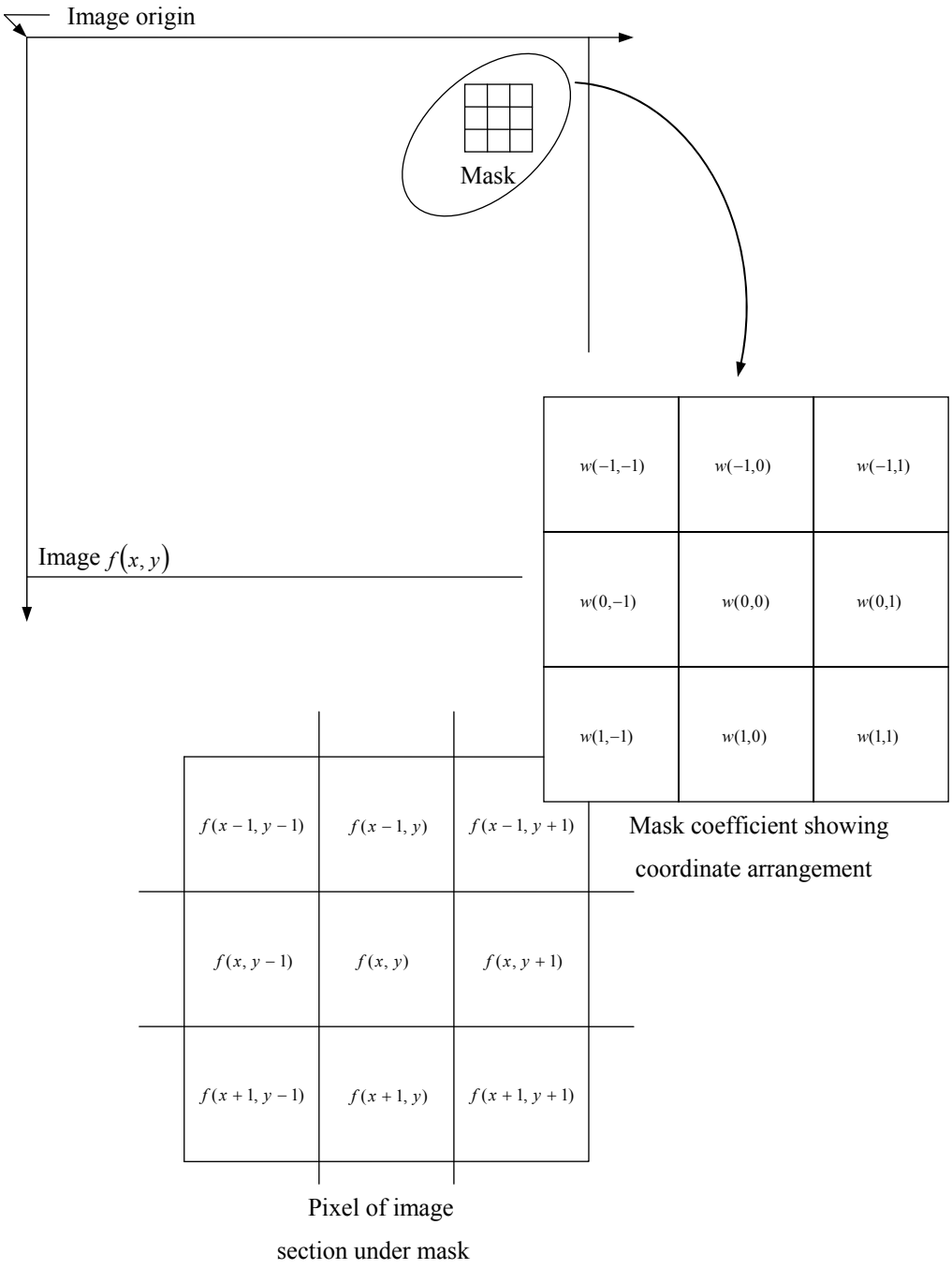


Figure 3.3 The Mechanics of Spatial Filtering. The Magnified Drawing Shows a 3×3 Mask and the Image Section Directly Under it

In general, linear filtering of an image f of size $M \times N$ with a filter mask of size $m \times n$ is given by the expression:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{3.3}$$

where $a=(m-1)/2$ and $b=(n-1)/2$. To generate a complete filtered image this equation must be applied for $x=0,1,2,\ldots,M-1$ and $y=0,1,2,\ldots,N-1$. In this way, it is assured that the mask processes all pixels in the image. It is easily verified when $m=n=3$ that this expression reduces to the example given in the previous paragraph [19].

The process of linear filtering given in Equation (3.3) is similar to a frequency domain concept called convolution. For this reason, linear spatial filtering often is referred to as "convolving a mask with an image". Similarly, filter masks are sometimes called convolution masks. The term convolution kernel also is in common use.

When interest lies on the response, *R*, of an $m\times n$ mask at any point (*x*, *y*), and not on the mechanics of implementing mask convolution, it is common practice to simplify the notation by using the following expression:

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} \tag{3.4}$$

$$= \sum_{i=1}^{mn} w_i z_i .$$

where the *w*'s are mask coefficients, the *z*'s are the values of the image gray levels corresponding to those coefficients, and mn is the total number of coefficients in the mask. For the 3×3 general mask shown in Figure 3.3 the response at any point (*x*, *y*) in the image is given by

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 \tag{3.5}$$

$$= \sum_{i=1}^{9} w_i z_i .$$

Nonlinear spatial filters also operate on neighborhoods, and the mechanics of sliding a mask past an image are the same as was just outlined. In general, however, the filtering operation is based conditionally on the values of the pixels in the neighborhood under consideration, and they do not explicitly use coefficients in the sum-of-products manner described in Equations (3.3) and (3.4). As shown in Section 3.4.2, for example, noise reduction can be achieved effectively with a nonlinear filter whose basic function is to compute the median gray-level value in the neighborhood in which the filter is located. Computation of the median is a nonlinear operation, as is computation of the variance. The Figure 3.4 gives another representation of a general 3×3 spatial filter mask [19].

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 3.4 Another Representation of a General 3×3 Spatial Filter Mask

An important consideration in implementing neighborhood operations for spatial filtering is the issue of what happens when the center of the filter approaches the border of the image. Consider for simplicity a square mask of size $n \times n$. At least one edge of such a mask will coincide with the border of the image when the center of the mask is at a distance of $(n-1)/2$ pixels away from the border of the image. If the center of the mask moves any closer to the border, one or more rows or columns of the mask will be located outside the image plane. There are several ways to handle this situation. The simplest is to limit the excursions of the center of the mask to be at a distance no less than $(n-1)/2$ pixels from the border. The resulting filtered image will be smaller than the original, but all the pixels in the filtered imaged will have been processed with the full mask. If the result is required to be the same size as the original, then the approach typically employed is to filter all pixels only with the section of the mask that is fully contained in the image. With this approach, there will be bands of pixels near the border that will have been processed with a partial filter mask. Other approaches include "padding" the image by adding rows and columns of 0's (or other constant gray level), or padding by replicating rows or columns. The padding is then stripped off at the end of the process. This keeps the size of the filtered image the same as the original, but the values of the padding will have an effect near the edges that becomes more prevalent as the size of the mask increases. The only way to obtain a perfectly filtered result is to accept a somewhat smaller filtered image by limiting the excursions of the center of the filter mask to a distance no less than $(n-1)/2$ pixels from the border of the original image [20].

3.4 Smoothing Spatial Filters

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering [19].

3.4.1 Smoothing Linear Filters

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called averaging filters. They also are referred to a lowpass filters.

The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in gray levels. Because random noise typically consists of sharp transitions in gray levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable features of an image) also are characterized by sharp transitions in gray levels, so averaging filters have the undesirable side effect that they blur edges. Another application of this type of process includes the smoothing of false contours that result from using an insufficient number of gray levels. A major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant" we mean pixel regions that are small with respect to the size of the filter mask [20].

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

(a)

$\frac{1}{16}\times$

1	2	1
2	4	2
1	2	1

(b)

Figure 3.5 Two 3×3 Smoothing (Averaging) Filter Masks

Figure 3.5 shows two 3×3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask. The constant multiplier in front of each mask is equal to the sum of the values of its coefficients, as is required to compute an average. This can best be seen by substituting the coefficients of the mask into Equation. (3.5):

$$R=\frac{1}{9}\sum_{i=1}^{9}z_i,$$

which is the average of the gray levels of the pixels in the 3×3 neighborhood defined by the mask. Note that, instead of being $1/9$, the coefficients of the filter are all 1's. The idea here is that it is computationally more efficient to have coefficients valued 1. At the end of the filtering process the entire image is divided by 9. An $m\times n$ mask would have a normalizing constant equal to $1/mn$. A spatial averaging filter in which all coefficients are equal is sometimes called a box filter [19].

The second mask shown in Figure 3.5 is a little more interesting. This mask yields a so-called weighted average, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown in Figure 3.5(b) the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask. The diagonal terms are further away from the center than the orthogonal neighbors (by a factor of $\sqrt{2}$) and, thus, are weighed less than these immediate neighbors of the center pixel. The basic strategy behind weighing the center point the highest and then reducing the value of the coefficients as a function of increasing distance from the origin is simply an attempt to reduce blurring in the smoothing process. It is possible to pick other weights to accomplish the same general objective. However, the sum of all the coefficients in the mask of Figure 3.5(b) is equal to 16, an attractive feature for computer implementation because it has an integer power of 2. In practice, it is difficult in general to see differences between images smoothed by using either of the masks in Figure 3.5, or similar arrangements, because the area these masks span at any one location in an image is so small.

With reference to Equation (3.3), the general implementation for filtering an $M \times N$ image with a weighted averaging filter of size $m \times n$ (m and n odd) is given by the expression

$$g(x,y) = \frac{\sum_{s=-a}^{a} \sum_{s=-b}^{b} w(s,t) f(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)} \tag{3.6}$$

The parameters in this equation are as defined in Equation (3.3). As before, it is understood that the complete filtered image is obtained by applying Equation (3.6) for $x = 0,1,2,\ldots,M-1$ and $y = 0,1,2,\ldots,N-1$. The denominator in Equation (3.6) is simply the sum of the mask coefficients and, therefore, it is a constant that needs to be computed only once. Typically, this scale factor is applied to all the pixels of the output image after the filtering process is completed [19].

3.4.2 Order-Statistics Filters

Order-statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result. The best-known example in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median). Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.

The median, ξ, of a set of values is such that half the values in the set are less than or equal to ξ, and half are greater than or equal to ξ. In order to perform median filtering at a point in an image, we first sort the values of the pixel in question and its neighbors, determine their median, and assign this value to that pixel. For example, in a 3×3 neighborhood the median is the 5th largest value, in a 5×5 neighborhood the 13th largest value, and so on. When several values in a neighborhood are the same, all equal values are grouped. For example, suppose that a 3×3 neighborhood has values (10, 20,

20, 20, 15, 20, 20, 25, 100). These values are sorted as (10, 15, 20, 20, 20, 20, 20, 25, 100), which results in a median of 20. Thus, the principal function of median filters is to force points with distinct gray levels to be more like their neighbors. In fact, isolated clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than $n^2/2$ (one-half the filter area), are eliminated by an $n \times n$ median filter. In this case "eliminated" means forced to the median intensity of the neighbors. Larger clusters are affected considerably less [19].

Although the median filter is by far the most useful order-statistics filter in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called max filter, which is useful in finding the brightest points in an image. The response of a 3×3 max filter is given by $R = \max\{z_k | k = 1, 2, \ldots, 9\}$. The 0th percentile filter is the min filter, used for the opposite purpose. Median, max, and mean filters are considered in more detail in section 3.7.2 [20].

3.5 A Model of the Image Degradation/Restoration Process

As Figure 3.6 shows, the degradation process is modeled in this chapter as a degradation function that, together with an additive noise term, operates on an input image $f(x, y)$ to produce a degraded image $g(x, y)$. Given $g(x, y)$, some knowledge about the degradation function H, and some knowledge about the additive noise term $q(x, y)$, the objective of restoration is to obtain an estimate $f(x, y)$ of the original image. We want the estimate to be as close as possible to the original input image and, in general the more we know about H and η, the closer $\hat{f}(x, y)$ will be to $f(x, y)$. The approach used throughout most of this chapter is based on various types of image restoration filters.

If H is a linear, position-invariant process, then the degraded image is given in the spatial domain by

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y) \tag{3.7}$$

where $h(x, y)$ is the spatial representation of the degradation function and the symbol * indicates convolution. The convolution in the spatial domain is equal to multiplication in the frequency domain, so the model in Equation (3.7) might be written in an equivalent frequency domain representation:

$$G(u,v) = H(u,v)F(u,v) + N(u,v) \tag{3.8}$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in Equation (3.7).

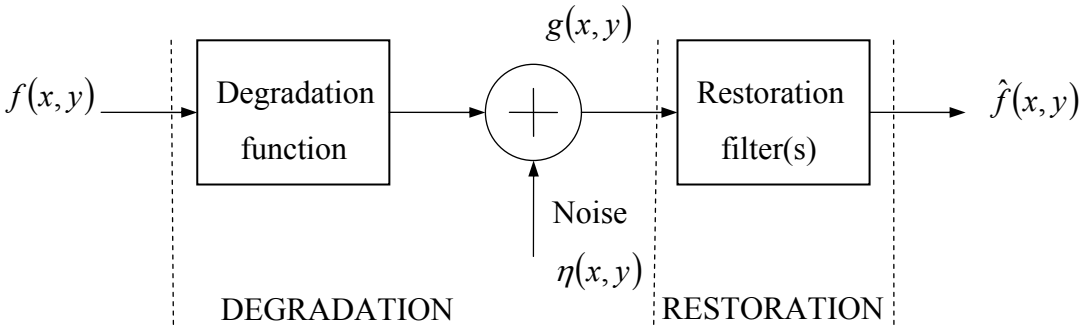


Figure 3.6 A model of the Image Degradation/ Restoration Process.

In the following two sections, we assume that H is the identity operator, and we deal only with degradations due to noise [20].

3.6 Noise Models

The principal sources of noise in digital images arise, during image acquisition (digitization) and/or transmission. The performance of imaging sensors is affected by a variety of factors, such as environmental conditions during image acquisition, and by the quality of the sensing elements themselves. For instance, in acquiring images with an camera, light levels and sensor temperature are major factors affecting the amount of noise in the resulting image. Images are corrupted during transmission principally due to interference in the channel used for transmission. For example, an image transmitted using a wireless network might be corrupted as a result of lightning or other atmospheric disturbance [19].

3.6.1 Spatial and Frequency Properties of Noise

Relevant to this discussion are parameters that define the spatial characteristics of noise, and whether the noise is correlated with the image, Frequency properties refer to the frequency content of noise in the Fourier sense (i.e., as opposed to the electromagnetic spectrum). For example, when the Fourier spectrum of noise is constant, the noise usually is called white noise. This terminology is a carry over from the physical

properties of white light, which contains nearly all frequencies in the visible spectrum in equal proportions. It is not difficult to show that the Fourier spectrum of a function containing all frequencies in equal proportions is a constant. With the exception of spatially periodic noise, noise is independent of spatial coordinates, and that it is uncorrelated with respect to the image itself (that is, there is no correlation between pixel values and the values of noise components) [19].

3.6.2 Gaussian Noise

Gaussian noise is a very good approximation of noise that occurs in many practical cases. Probability density of the random variable is given by the Gaussian curve. Based on the assumptions in the previous section, the spatial noise descriptor with which it shall be concerned is the statistical behavior of the gray-level values in the noise component of the model in Figure 3.6. These may be considered random variables, characterized by a probability density function (PDF).

Because of its mathematical tractability in both the spatial and frequency domains, Gaussian (also called normal) noise models are used frequently in practice. In fact, this tractability is so convenient that it often results in Gaussian models being used in situations in which they are marginally applicable at best. The Figure 3.7 showing the Gaussian probability density function.

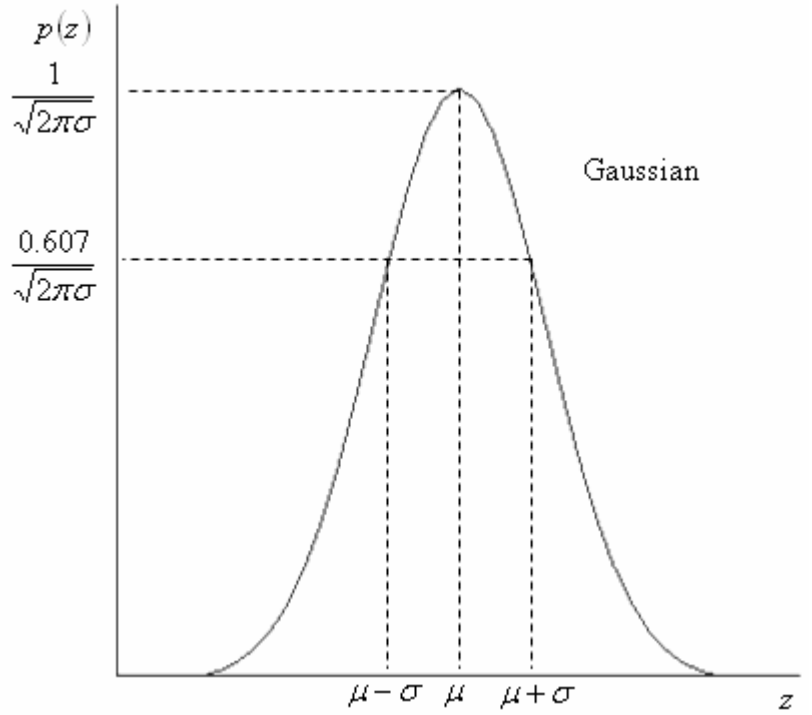


Figure 3.7 Gaussian Probability Density Function

The PDF of a Gaussian random variable, z is given by

$$p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\mu)^2/2\sigma^2} \tag{3.9}$$

where z represents gray level, μ is the mean of average value of z, and σ is its standard deviation. The standard deviation squared, σ^2, is called the variance of z. A plot of this function is shown in Figure 3.7. When z is described by Equation (3.9), approximately 70% of its values will be in the range $[(\mu-\sigma),(\mu+\sigma)]$, and about 95% will be in the range $[(\mu-2\sigma),(\mu+2\sigma)]$ [20].

3.7 Restoration in the Presence of Noise Only-Spatial Filtering

When the only degradation present in an image is noise, Equations (3.7) and (3.8) become

$$g(x,y) = f(x,y) + \eta(x,y) \tag{3.10}$$

and

$$G(u,v) = F(u,v) + N(u,v) \tag{3.11}$$

The noise terms are unknown, so subtracting them from $g(x, y)$ or $G(u, v)$ is not a realistic option. In the case of periodic noise, it usually is possible to estimate $N(u, v)$ from the spectrum of $G(u, v)$. In this case $N(u, v)$ can be subtracted from $G(u, v)$ to obtain an estimate of the original image. In general, however, this type of knowledge is the exception, rather than the rule.

Spatial filtering is the method of choice in situations when only additive noise is present. This topic was discussed in detail in Sections 3.3 and 3.4 in connection with image enhancement. In fact, enhancement and restoration become almost indistinguishable disciplines in this particular case [19]. With the exception of the nature of the computation performed by a specific filter, the mechanics for implementing all the filters that follow are exactly as discussed in Section 3.3.

3.7.1 Mean Filters

In this section, the noise-reduction spatial filters introduced in Section 3.4 are discussed briefly including several other filters whose performances are in many cases superior to the filters discussed in that section.

3.7.1.1 Arithmetic Mean Filter

This is the simplest of the mean filters. Let S_{xy} represent the set of coordinates in a rectangular subimage window of size $m \times n$, centered at point (x, y). The arithmetic mean filtering process computes the average value of the corrupted image $g(x, y)$ in the area defined by S_{xy}. The value of the restored image $\hat{f}$ at any point (x, y) is simply the arithmetic mean computed using the pixels in the region defined by S_{xy},. In other words,

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s,t) \tag{3.12}$$

This operation can be implemented using a convolution mask in which all coefficients have value $1/mn$. As discussed in Section 3.4.1, a mean filter simply smoothes local variations in an image. Noise is reduced as a result of blurring.

3.7.1.2 Geometric Mean Filter

An image restored using a geometric mean filter is given by the expression

$$\hat{f}(x,y) = \left[\prod_{(s,t) \in S_{xy}} g(s,t) \right]^{\frac{1}{mn}} \tag{3.13}$$

Here, each restored pixel is given by the product of the pixels in the subimage window, raised to the power $1/mn$. A geometric mean filter achieves smoothing comparable to the arithmetic mean filter, but it tends to lose less image detail in the process [19].

3.7.1.3 Harmonic Mean Filter

The harmonic mean filtering operation is given by the expression

$$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s,t)}} \tag{3.14}$$

The harmonic mean filter works well for salt noise, but fails for pepper noise. It does well also with other types of noise like Gaussian noise [19].

3.7.1.4 Contraharmonic Mean Filter

The contraharmonic mean filtering operation yields a restored image based on the expression

$$\hat{f}(x,y)=\frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}} \tag{3.15}$$

where Q is called the order of the filter. This filter is well suited for reducing or virtually eliminating the effects of salt-and-pepper noise. For positive values of Q, the filter eliminates pepper noise. For negative values of Q it eliminates salt noise. It cannot do both simultaneously. Note that the contraharmonic filter reduces to the arithmetic mean filter if $Q=0$, and to the harmonic mean filter if $Q=-1$ [19].

3.7.2 Order-Statistics Filters

Order-statistics filters were introduced in Section 3.4.2. As noted in Section 3.4.2, order-statistics filters are spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter. The response of the filter at any point is determined by the ranking result [19].

3.7.2.1 Median Filter

The best-known order-statistics filter is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel

$$\hat{f}(x,y)=\underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\} \tag{3.16}$$

The original value of the pixel is included in the computation of the median. Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of both bipolar and unipolar impulse noise [19]. In fact, the median filter yields excellent results for images corrupted by this type of noise. Computation of the median and implementation of this filter are discussed in detail in Section 3.4.2.

3.7.2.2 Max and Min Filters

Although the median filter is by far the order-statistics filter most used in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but the reader will recall from basic statistics that ranking lends

itself to many other possibilities. For example, using the 100th percentile results in the so-called max filter, given by

$$\hat{f}(x,y) = \max_{(s,t) \in S_{xy}} \{g(s,t)\} \tag{3.17}$$

This filter is useful for finding the brightest points in an image. Also, because pepper noise has very low values, it is reduced by this filter as a result of the max selection process in the subimage area S_{xy}.

The 0th percentile filter is the min filter

$$\hat{f}(x,y) = \min_{(s,t) \in S_{xy}} \{g(s,t)\} \tag{3.18}$$

This filter is useful for finding the darkest points in an image. Also, it reduces salt noise as a result of the min operation [19].

3.7.2.3 Midpoint Filter

The midpoint filter simply computes the midpoint between the maximum and minimum values in the area encompassed by the filter

$$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t) \in S_{xy}} \{g(s,t)\} + \min_{(s,t) \in S_{xy}} \{g(s,t)\}\right] \tag{3.19}$$

Note that this filter combines order statistics and averaging. This filter works best for randomly distributed noise, like Gaussian or uniform noise [19].

3.7.2.4 Alpha-trimmed Mean Filter

Suppose that the $d/2$ lowest and the $d/2$ highest gray-level values of $g(s, t)$ in the neighborhood S_{xy} are deleted. Let $g_r(s,t)$ represent the remaining $mn - d$ pixels. A filter formed by averaging these remaining pixels is called an alpha-trimmed mean filter

$$\hat{f}(x,y) = \frac{1}{mn - d} \sum_{(s,t) \in S_{xy}} g_r(s,t) \tag{3.20}$$

where the value of d can range from 0 to $mn - 1$. When $d = 0$, the alpha-trimmed filter reduces to the arithmetic mean filter discussed in the previous section. If we choose $d = mn - 1$, the filter becomes a median filter. For other values of d, the alpha-trimmed filter is useful in situations involving multiple types of noise, such as a combination of salt-and-pepper and Gaussian noise [19].

3.7.3 Adaptive Filters

Once selected, the filters discussed thus far are applied to an image without regard for how image characteristics vary from one point to another. In this section, two simple adaptive filters whose behavior changes based on statistical characteristics of the image inside the filter region defined by the $m \times n$ rectangular window S_{xy}. As shown in the following discussion, adaptive filters are capable of performance superior to that of the filters discussed thus far. The price paid for improved filtering power is an increase in filter complexity [20].

3.7.3.1 Adaptive, Local Noise Reduction Filter

The simplest statistical measures of a random variable are its mean and variance. These are reasonable parameters on which to base an adaptive filter because they are quantities closely related to the appearance of an image. The mean gives a measure of average gray level in the region over which the mean is computed, and the variance gives a measure of average contrast in that region.

The filter is to operate on a local region, S_{xy}. The response of the filter at any point (x, y) on which the region is centered is to be based on four quantities: (a) $g(x, y)$, the value of the noisy image at (x, y) ; (b) σ_η^2, the variance of the noise corrupting $f(x, y)$ to form $g(x, y)$; (c) m_L, the local mean of the pixels in S_{xy} and (*d*) σ_L^2. The local variance of the pixels in S_{xy}. The behavior of the filter should be as follows:

1. If σ_η^2 is zero, the filter should return simply the value of $g(x, y)$. This is the trivial, zero-noise case in which $g(x, y)$ is equal to $f(x, y)$.
2. If the local variance is high relative to σ_η^2 the filter should return a value close to $g(x, y)$. A high local variance typically is associated with edges, and these should be preserved.
3. If the two variances are equal, we want the filter to return the arithmetic mean value of the pixels in S_{xy}. This condition occurs when the local area has the same properties as the overall image, and local noise is to be reduced simply by averaging.

An adaptive expression for obtaining $\hat{f}(x,y)$ based on these assumptions maybe written as

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L] \tag{3.21}$$

The only quantity that needs to be known or estimated is the variance of the overall noise, σ_η^2. The other parameters are computed from the pixels in S_{xy}, at each location (x, y) on which the filter window is centered. A tacit assumption in Equation (3.20) is that $\sigma_\eta^2 \leq \sigma_L^2$,. The noise in our model is additive and position independent, so this is a reasonable assumption to make because S_{xy} is a subset of $g(x, y)$. Therefore, it is possible for this condition to be violated in practice. For that reason, a test should be built into an implementation of Equation (3.21) so that the ratio is set to 1 if the condition $\sigma_\eta^2 > \sigma_L^2$ occurs. This makes this filter nonlinear. However, it prevents nonsensical results (i.e., negative gray levels, depending on the value of m_L) due to a potential lack of knowledge about the variance of the image noise. Another approach is to allow the negative values to occur, and then rescale the gray level values at the end. The result then would be a loss of dynamic range in the image [19].

3.7.3.2 Adaptive Median Filter

The median filter discussed in Section 3.7.2.1 performs well as long as the spatial density of the impulse noise is not large (as a. It is shown in this section that adaptive median filtering can handle impulse noise with probabilities even larger than these. An additional benefit of the adaptive median filter is that it seeks to preserve detail while smoothing nonimpulse noise, something that the traditional median filter does not do. The adaptive median filter also works in a rectangular window area S_{xy}. Unlike those filters, however, the adaptive median filter changes (increases) the size of S_{xy} during filter operation, depending on certain conditions listed in this section. Keep in mind that the output of the filter is a single value used to replace the value of the pixel at (x, y), the particular point on which the window S_{xy}, is centered at a given time.

Consider the following notation:

- $z_{\min}$ = minimum gray level value in S_{xy}

- $z_{\max}$ = maximum gray level value in S_{xy}
- z_{med} = median gray levels in S_{xy}
- z_{xy} = gray level at coordinates (x, y)
- $S_{\max}$ = maximum allowed size of S_{xy}.

The adaptive median filtering algorithm works in two levels, denoted level A and level B, as follows:

- Level A: $A1 = z_{\text{med}} - z_{\min}$

  $A2 = z_{\text{med}} - z_{\text{man}}$

  If $A1 > 0$ AND $A2 < 0$, Go to level B

  Else increase the window size

  If window size $\leq S_{\max}$ repeat level A

  Else output z_{med}.
- Level B: $B1 = z_{\text{xy}} - z_{\min}$

  $B1 = z_{\text{xy}} - z_{\max}$

  If $B1 > 0$ AND $B2 < 0$, output z_{xy}

  Else output z_{med}.

The key to understanding the mechanics of this algorithm is to keep in mind that it has three main purposes: to remove salt-and-pepper (impulse) noise, to provide smoothing of other noise that may not be impulsive, and to reduce distortion, such as excessive thinning or thickening of object boundaries. The values $z_{\min}$, and $z_{\max}$ are considered statistically by the algorithm to be ‘‘impulselike’’ noise components, even if these are not the lowest and highest possible pixel values in the image.

With these observations in mind, the purpose of level A is to determine if the median filter output, z_{med}, is an impulse (black or white) or not. If the condition $z_{\min} < z_{\text{med}} < z_{\max}$ holds, then z_{med} cannot be an impulse for the reason mentioned in the previous paragraph. In this case, go to level B and test to see if the point in the center of the window, z_{xy}, is itself an impulse (recall that z_{xy} is the point being processed). If the condition $B1 > 0$ AND $B2 < 0$ is true, then $z_{\min} < z_{xy} < z_{\max}$, and z_{xy} cannot be an impulse for the same reason that z_{med} was not. In this case, the algorithm outputs the

unchanged pixel value, z_{xy}. By not changing these ‘‘intermediate-level’’ points, distortion is reduced in the image. If the condition $B1 > 0$ AND $B2 < 0$ is false, then either $z_{xy} = z_{\min}$ or $z_{xy} = z_{\max}$. In either case, the value of the pixel is an extreme value and the algorithm outputs the median value z_{med}, which we know from level A is not a noise impulse. The last step is what the standard median filter does. The problem is that the standard median filter replaces every point in the image by the median of the corresponding neighborhood. This causes unnecessary loss of detail.

Continuing with the explanation, suppose that level A does find an impulse (i.e., it fails the test that would cause it to branch to level B). The algorithm then increases the size of the window and repeats level A. This looping continues until the algorithm either finds a median value that is not an impulse (and branches to level B), or the maximum window size is reached. If the maximum window size is reached, the algorithm returns the value of z_{med}. Note that there is no guarantee that this value is not an impulse. The smaller the noise probabilities p_a and/or p_b are, or the larger $S_{\max}$ is allowed to be, the less likely it is that a premature exit condition will occur. This is plausible. As the density of the impulses increases, it stands to reason that we would need a larger window to ‘‘clean up’’ the noise spikes.

Every time the algorithm outputs a value, the window S_{xy} is moved to the next location in the image. The algorithm then is reinitialized and applied to the pixels in the new location. The median value can be updated iteratively using only the new pixels, thus reducing computational overhead [19].

3.8 Summary

As mentioned before, this chapter discussed most important image restoration techniques in spatial domain and types of filters that is used for this case (deal with only additive noise, when only degradation function is noise).

Next chapter will present the selected filters from spatial domain that described as a good noise removal for Gaussian noise, and comparison criteria will be used to compare between previous mentioned filters and LDPC system.

CHAPTER FOUR

Methodology

4.1 Overview

This chapter presents a proposed system of digital image transmission and restoration. It also discusses the details of criteria for the comparison of the proposed LDPC system and other restoration methods.

4.2 Comparison Criteria

In order to compare between LDPC codes and image restoration techniques in spatial domain, comparison criteria should first be defined. Visual inspection of reconstructed images and analysis using the comparison criteria will help in deciding which restoration method is superior.

The comparison criteria suggested include:

- PSNR Values
- Contrast
- Brightness
- Processing Time
- Visual Inspection

4.2.1 PSNR Values

One of the simple ways to measure the difference between an original image and a reconstructed image is to measure the PSNR (Peak Signal to Noise Ratio) value [22]. A higher value of PSNR means higher quality of reconstructed images. The PSNR value is calculated according to the following form

$$PSNR = 20\log_{10}^{(b/rms)}$$

where b is the largest possible value of the signal (typically 255), and RMS is the root mean square difference between two images. The PSNR is given in decibel units (dB), which measures the ratio of the peak signal and the difference between two images. An increase of 20 *dB* corresponds to a ten-fold decrease in the rms difference between two images. There are many versions of signal-to-noise ratios, but the PSNR is very

common in image processing, probably because it gives better-sounding numbers than other measures. It is defined using the following formula

$$PSNR = 20\log_{10}\left[\frac{255}{\sqrt{\left(\frac{1}{255^2}\sum_{i,j}(x(i,j) - x_{REC}(i,j))^2\right)}}\right] \quad (4.1)$$

4.2.2 Contrast

Contrast adjustment increases or decreases the apparent difference in lightness between lighter and darker pixels. Increasing or decreasing contrast is applicable to each pixel in image. Increasing contrast, if gray value of pixel is greater 128 contrast changes the value up to 255, if the pixel value is less than or equal to 128 contrast change the value down to 0. In decreasing contrast, if gray value of pixel grater than 128 contrast change the value down to 128, if the pixel value is less than or equal to 128 contrast changes the value up to 128 according to the contrast adjustment level as shown in Figure 4.1.

It is better in the experiment if the total change in contrast between the original image and the restored image is minimum [23].

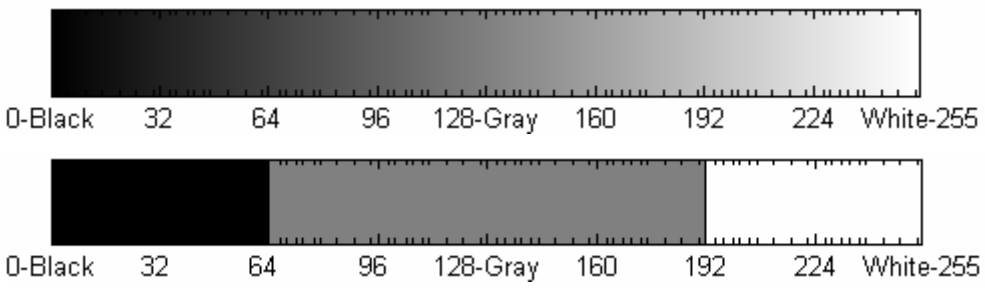


Figure 4.1 Grayscale Palette

Image contrast is defined as the difference between the average of the '*N*' brightest pixels present in the image and the average of the '*M*' darkest pixels in the image, as follow form.

Average of the '*N*' Whitest Pixels - Average of the '*M*' Blackest Pixels

Total change in contrast of an image will be calculated and compared with the total change in contrast of a reconstructed image using the following equation.

$$\text{Contrast of an image} = \sum_{i,j}^{256} Org(x_{ij} y_{ij}) - \text{Re}c(x_{ij} y_{ij}) \quad (4.2)$$

4.2.3 Brightness

Brightness is used to increase or decrease the brightness of pixel. Low brightness will result in dark tones while high brightness will result in higher, pastel tones.

Increasing or decreasing the brightness is applicable in same level to each pixel in image. Increasing brightness changes the pixel values up to 255 while decreasing brightness changes the pixel values down to 0 according to brightness adjustment level.

In experiments, brightness of original images will be compared with brightness of the reconstructed images [23]. Reconstructed images that have a brightness level most near to brightness level of an original image will be better. Brightness of an image is calculated according to the following formula:

$$\text{Brightness of an image} = \frac{Total\ Gray\ Values\ of\ each\ pixel\ in\ image}{256 \times 256} \quad (4.3)$$

4.2.4 Processing Time

Processing time is the period of time that the Matlab program takes to restore an image for each method.

4.2.5 Visual Inspection

This is based on visual inspection and observation of humans. The results of this visual inspection form part of the comparison criteria which is then combined with the results of the computed analysis in order to decide upon the ideal restoration technique [23].

4.3 System Structure and Design

As mentioned in chapter three, the system is dealing with additive white Gaussian noise only. As the additive noise added to the original image, the system will de-noise the original image according to the LDPC decoder and the other chosen filters, then comparing the result of LDPC system with the image restoration techniques according the comparison criteria listed in the previous section. Figure 4.2 shows the block diagram of image restoration techniques.

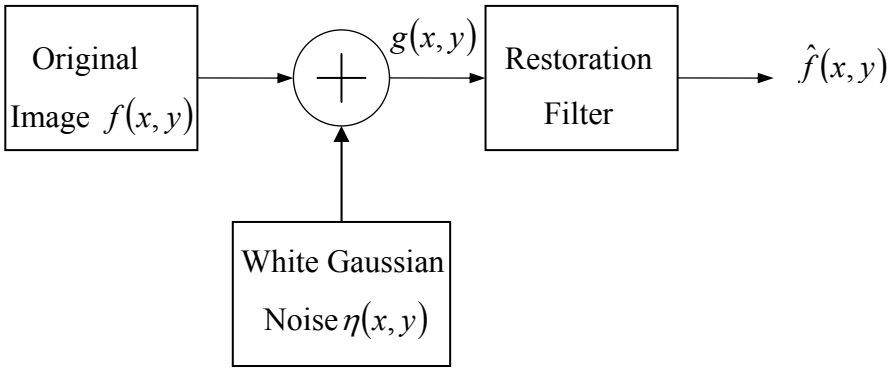


Figure 4.2 Block Diagram of the Image Restoration

Image restoration techniques used in the experiment will include three various filters; namely Harmonic Mean filter, Alpha-trimmed Mean filter and Adaptive Median filter. The only added noise is Gaussian noise. Figure 4.3 explains the stages of the system for those filters and prerequisites steps to filtering an captured image.

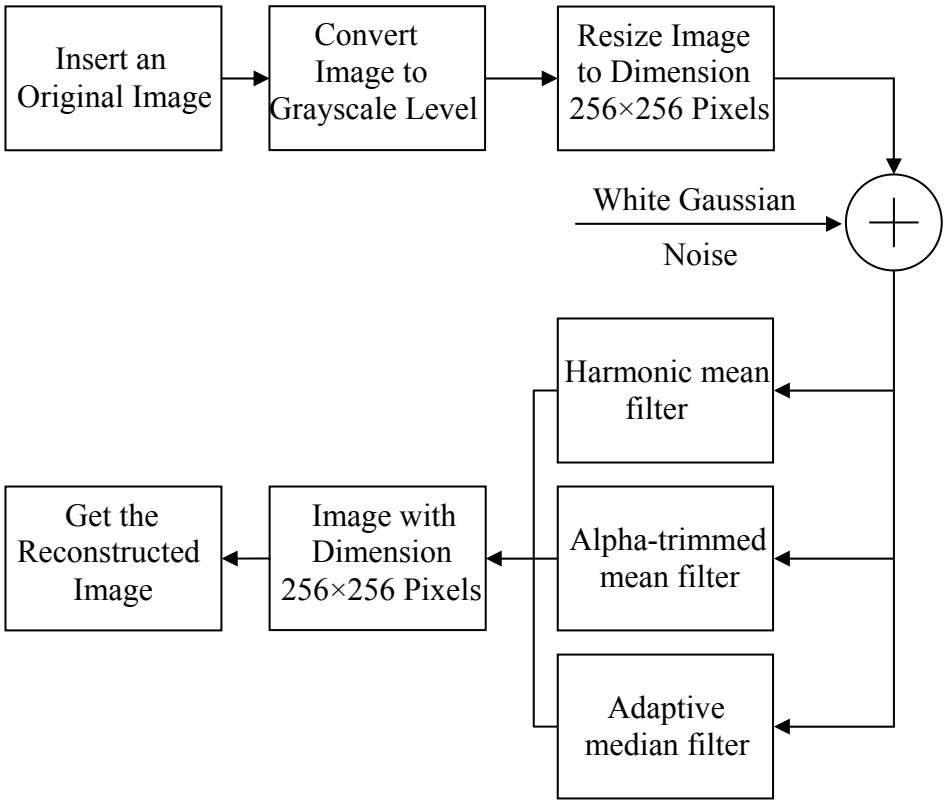


Figure 4.3 Block Diagram of Image Restoration Filters

Figure 4.4 shows the block diagram of LDPC system including the steps of the captured image that consist the grayscaling and resizing the dimension of the image and preparing step of the codeword blocks (data blocks).

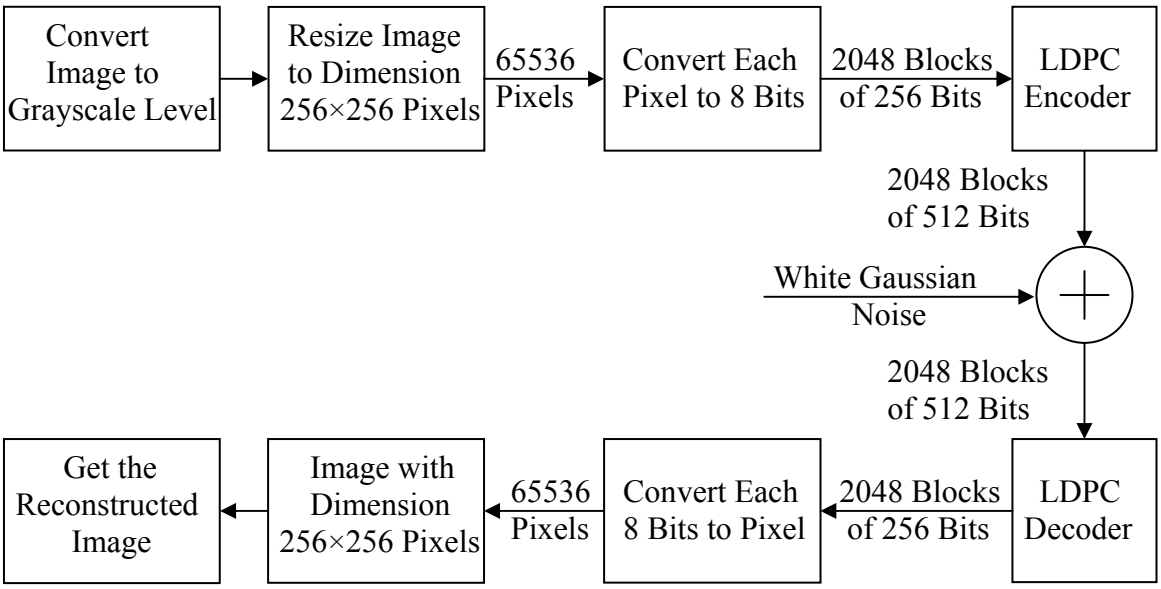


Figure 4.4 Block Diagram of the LDPC System

4.3.1 Data Representation

The third step in block diagram of LDPC system shows how the image data (pixels) is coded to pass through the LDPC encoder. The data is prepared according to the following steps:

1. An image of size $[256 \times 256]$ contains 65536 pixels. This number is obtained by multiplying the number of pixels in *X* axis (row) and the number of pixels in *Y* axis (column) of the image.
2. According to the grayscale image format each pixel should have a value between 0 to 255, so eight digits are needed to implement these values in binary number.
3. The matrix passing through the LDPC encoder should have a size of $[2048 \times 256]$. This matrix is obtained by dividing the result of multiplying number of pixels by the number of bits for each pixel to 256. Therefore, size of matrix representing the number of blocks is 2048 and the block length is 256.

4. According to the block length is equal $[1\times256]$ each block contain 32 pixels, this number is obtained by dividing the block length (256) to number of bit for each pixel (8).
5. The step after this data be noised after passing the encoder and passed form the decoder it will be with same steps but with opposite direction.

4.3.2 Data Encoding

Considering that *u* is representing the data words or according the system is applied in this thesis is the first block (matrix) of size [1×256], the encoding process of LDPC encoder is performable according to the general steps of LDPC encoder as follow:

- $\overline{C} = \overline{u}G = [\overline{u} : \overline{u}P]$ where *G*, as mentioned in chapter two is the generator matrix, and $\overline{C}$ is the code words matrix.
- Since $\overline{u}P = \overline{x}$ then $\overline{C} = [\overline{u} : \overline{x}]$.
- $H = \underbrace{\left[{}_{M}[\overbrace{A}^{M} \ \vdots \ \overbrace{B}^{K}]_{M} \right]}_{N}$ the $G = K \times N$ and $H = M \times N$ so they are related to each other, where *K* is data word length and *N* is a code word length, $N > K$ and *H* is representing the low density parity check matrix.
- Since $CH^T = HC^T = 0$ then $[A : B]\begin{bmatrix}\overline{u}\\ \overline{x}\end{bmatrix} = A\overline{u} + B\overline{x} = 0$, and $x = AB^{-1}\overline{u}$. Therefore, the data is coded according to the equation $\overline{C} = [\overline{u} : AB^{-1}\overline{u}]$ (with dimension $[N \times N]$).

The above steps of encoding process of LDPC encoder are performed in this thesis from the following

1. Data word matrix or in this case pixels matrix with size $u = [1\times256]$ then $\overline{u} = [2048\times256]$ and AB^{-1} should equal the size of $[256\times256]$ to perform the code word matrix $\overline{C} = [\overline{u} : AB^{-1}\overline{u}]$.
2. The matrix of $AB^{-1}\overline{u}$ should equal to size of $[2048\times256]$ so it could be performed by multiplication of AB^{-1} matrix by transpose of $\overline{u}$ matrix ($\overline{u}^T$) matrix to be as in form $[AB^{-1}\times\overline{u}^T]$.the resulting matrix equal to size $[256\times2048]$.

3. In this step taking of the transpose of the resulting matrix and putting it in the code word order as in form $\overline{C} = \left[\left[\overline{u}\right]:\left[AB^{-1}\overline{u}\right]^{T}\right]$ this arrangement gives request code word matrix of size $\left[2048 \times 512\right]$.
4. Sending this size of matrix row by row to the decoder with adding the Gaussian noise.

The Figure 4.5 shows the flowchart for encoding process of LDPC encoder in the mentioned system.

4.3.3 Data Decoding

According to the explanations given in the previous section and the first and second chapters *H* matrix is too important for decoding process and in this case it should have the dimension of $\left[256 \times 512\right]$.

The *H* matrix consists mostly of ones as explained previously in first and second chapters.

As will be mentioned later in experimental results, the experiment will be performed using four different values of variance (σ) of Gaussian noise in each case.

The decoding process of LDPC decoder is applied by the message passing - algorithm in probability domain perform looping below $\forall i, j$ for which $h_{ij} = 1$ according the following steps.

1. Initialize:

$$q_{ij}(0) = 1 - p_i = \frac{1}{1 + e^{-2Y_i/\sigma^2}}$$

$$q_{ij}(1) = p_i = \frac{1}{1 - e^{2Y_i/\sigma^2}}$$

2. Calculate:

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in R_j \setminus i}\left(1 - 2q_{i'j}(1)\right)$$

$$r_{ji}(1) = 1 - r_{ji}(0)$$

3. Update the values of $q_{ij}(0)$ and $q_{ij}(1)$ according to the following equations to perform the iterations until finding the correct message word:

$$q_{ij}(0) = K_{ij}(1 - p_i)\prod_{j' \in C_i \setminus j} r_{j'i}(0)$$

$$q_{ij}(1) = K_{ij}\ p_i \prod_{j' \in C_i \setminus j} r_{j'i}(1)$$

where the constants K_{ij} are chosen to ensure $q_{ij}(0) + q_{ij}(1) = 1$.

4. Compute $\forall i$:

$$Q_i(0) = K_i(1 - p_i)\prod_{j \in C_i} r_{ji}(0)$$

$$Q_i(1) = K_i\ p_i \prod_{j \in C_i} r_{ji}(1)$$

where the constants K_{ij} are chosen to ensure $Q_i(0) + Q_i(1) = 1$.

5. Apply following two conditions:

- $\forall i$,

$$\hat{C}_i = \begin{cases} 1 & if \quad Q_i(1) > 0.5 \\ 0 & else \end{cases}$$

- if $\left(\hat{\overline{C}}\, H^T\right) = \overline{0}$ or

$$\left(\#\text{of iterations} = \text{maximum iterations}\right)$$

then stop, else go to step 2.

Figure 4.6 shows the flowchart decoding process of LDPC decodes in the system.

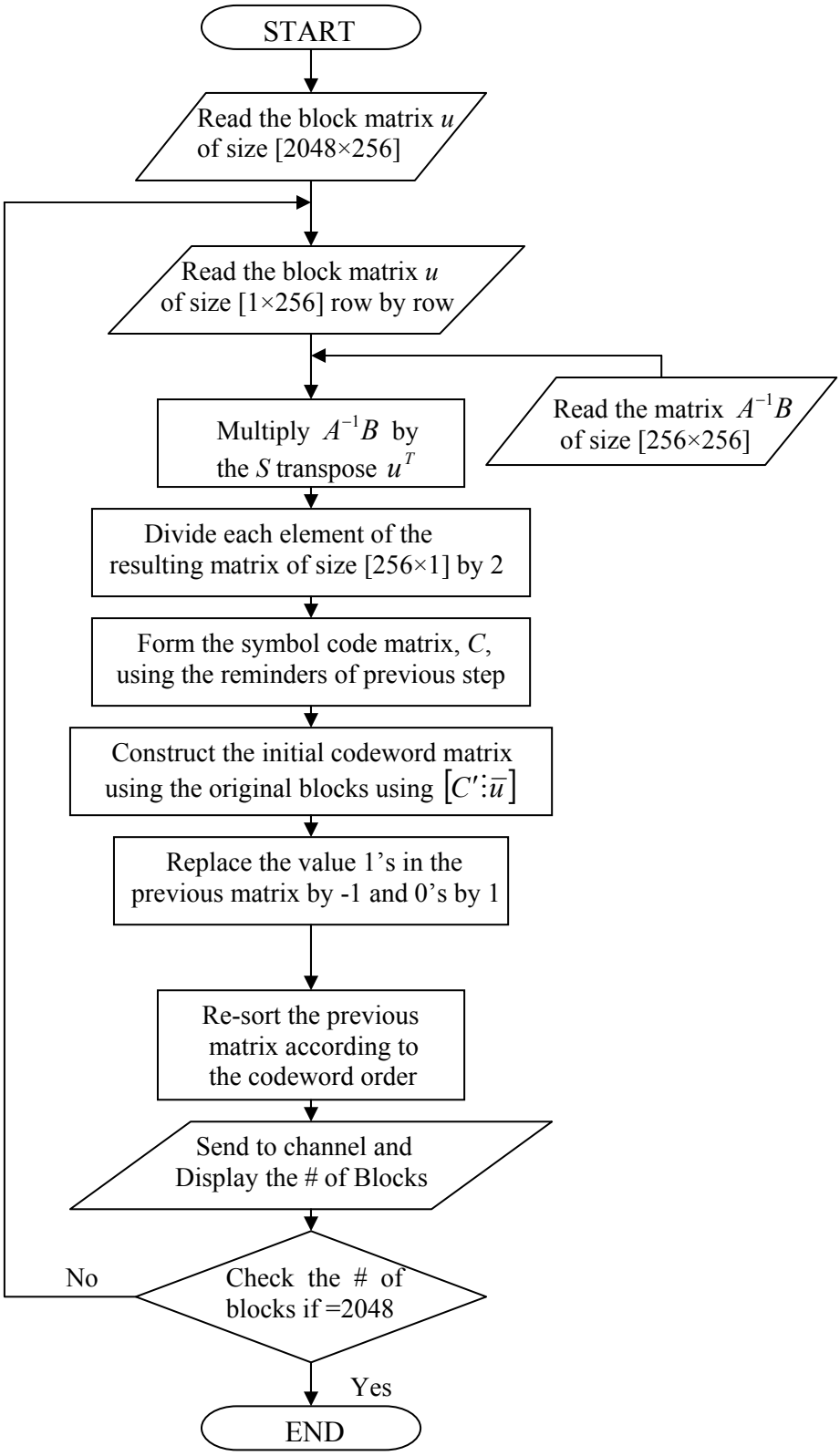


Figure 4.5 Flowchart of the LDPC Encoder

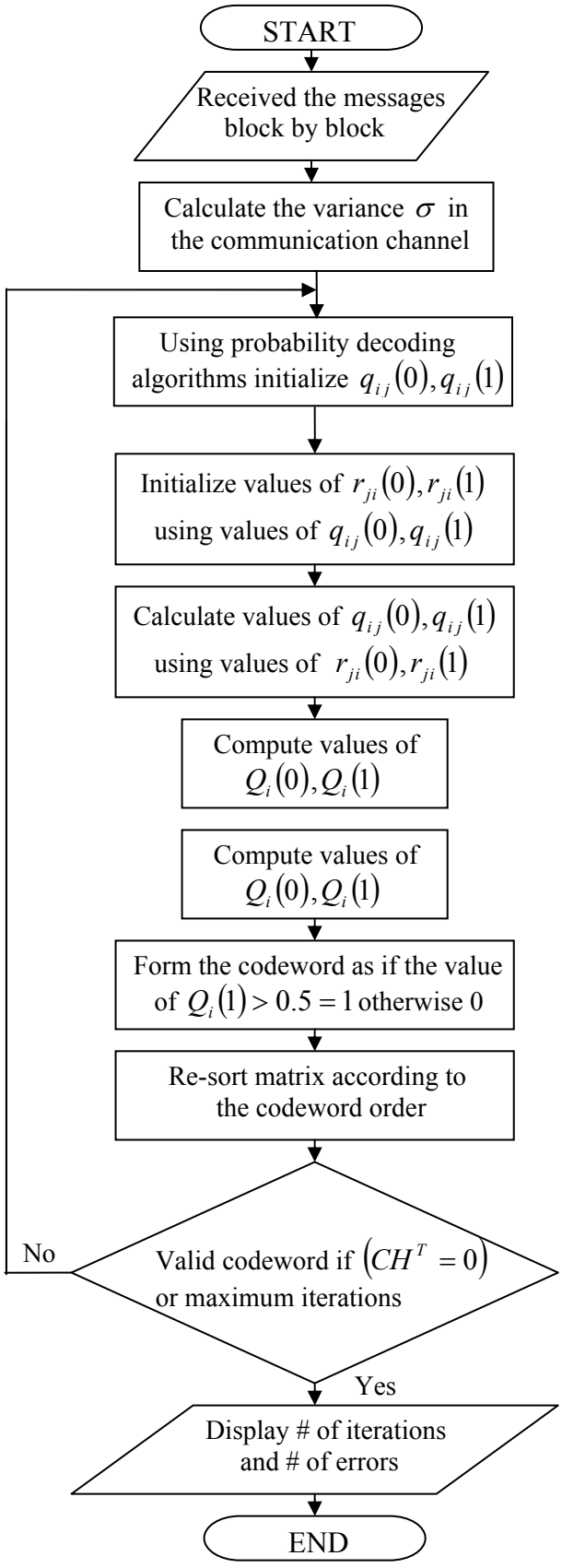


Figure 4.6 Flowchart of the LDPC Decoder

4.4 Prerequisites of the System

The system mentioned in this research needs some prerequisites to work properly. The prerequisites have many effects on the algorithm that is used in the assembled system. If they are not insuring satisfactory conditions, then the system will be not usable.

These conditions are used in different processing steps and are considered as insured by the following sections.

4.4.1 Image Acquisition

This stage does the capturing of the images using peripheral capturing devices. The image can be obtained from many sources. There are obvious ways to obtain digitized images: scanning a photo camera made picture, saving web-cam generated static or dynamic images, capturing with high quality video camera, or using other sources (e.g. manual scratches, painted images and results of other digital image processing procedures).

4.4.2 Initial Parameters

The initial parameters of the system are set as follows:

1. This system is accomplished to denoise the images noised with Gaussian noise with mean equal to zero, and the value of variance randomly selected, or as selected in the section of analysis the system.
2. The accomplished system set to process the images in grayscale level with dimensions of pixels equal $[256 \times 256]$.
3. The dimension of AB^{-1} matrix should be $[256 \times 256]$.
4. The dimension of H matrix should be $[256 \times 512]$.
5. In order to increase the dimensions of the input images the values of the other matrices respectively should be changed according to the explanations of data encoding and data decoding. Number of iteration in decoding process is set to be 100 iterations.
6. The size of window for Harmonic mean filter and Alpha-trimmed filter is set to be with size $[3 \times 3]$. For Adaptive median filter maximum allowable size $S_{\max} = 7$.

4.4.3 Database Collection

The database of images which are used in the experiments only contains images of dimensions $[256 \times 256]$. The database contains 5 normal images in the same resolution but with different backgrounds. The developed system is implemented using miscellaneous famous pictures database [21].

4.5 Software Tools (MATLAB)

This section contains a simple description of the tools that were used. The software implementation has been done using Matlab version 7.0, Image Processing Toolbox.

Matlab is a simulation environment for doing numerical computations with matrices and vectors. It handles a wide range of computing tasks in engineering and science, and has several built-in interfaces. In addition there are several toolboxes available to expand the capabilities of Matlab one such toolbox is image processing toolbox, which extends the Matlab computing environment to provide functions and interactive tools for enhancing and analyzing digital images and developing image processing algorithms.

4.6 Summary

A comparison criterion has been created to help select the ideal restoration method. The criteria includes: PSNR Values, Contrast, Brightness and Processing Time.

To make a decision upon which method is ideal, various images and criteria must be considered. An overall sufficient and good quality reconstructed image, with the highest value of variance and lowest PSNR is sought after.

This chapter explained in detail developed image restoration system and the experimental results will be discussed in the next chapter.

CHAPTER FIVE

Results and Analysis

5.1 Overview

This chapter will present a real life application of digital image restoration. Different restoration methods as well as low-density parity-check LDPC coding will be applied to different images and all the resulting reconstructed images will be analyzed using the comparison criteria in order to decide upon the optimum restoration method.

Additionally, the application software using Matlab to restore an image by different methods which are, LDPC encoding and decoding system, Harmonic mean filter, Alpha-trimmed mean filter, and Adaptive median filter, will be covered.

5.2 Experimental Results

In this section, various amounts of white Gaussian noise will be added to the selected images and comparison tables will be made based on some chosen criteria and visual inspection of the reconstructed images. All results will then be analyzed.

5.2.1 Images Database

To provide a real-life application, five images are chosen for analysis. These images are from a database of experimental work, and contain images of various contrasts and patterns. The five images are: Lena, Moon Surface, Clock and Moon, as shown in Figure 5.1.

The following sections of visual inspection could be dividing to three types of show.

The first block figures shows the original test images; and the second block figures will contain four block figures of images, every one consist from five images, the original image with four value of white Gaussian noise added to original image, shown with each other to help the viewer to understanding how the selected values of noise effect the images.

The Third section will contain sixteen block figures of images, every one consist from six images, the original one and the filtering results of that image.

As well as it is easy to observe that figures shows the best restored image is obtained by LDPC coding system.

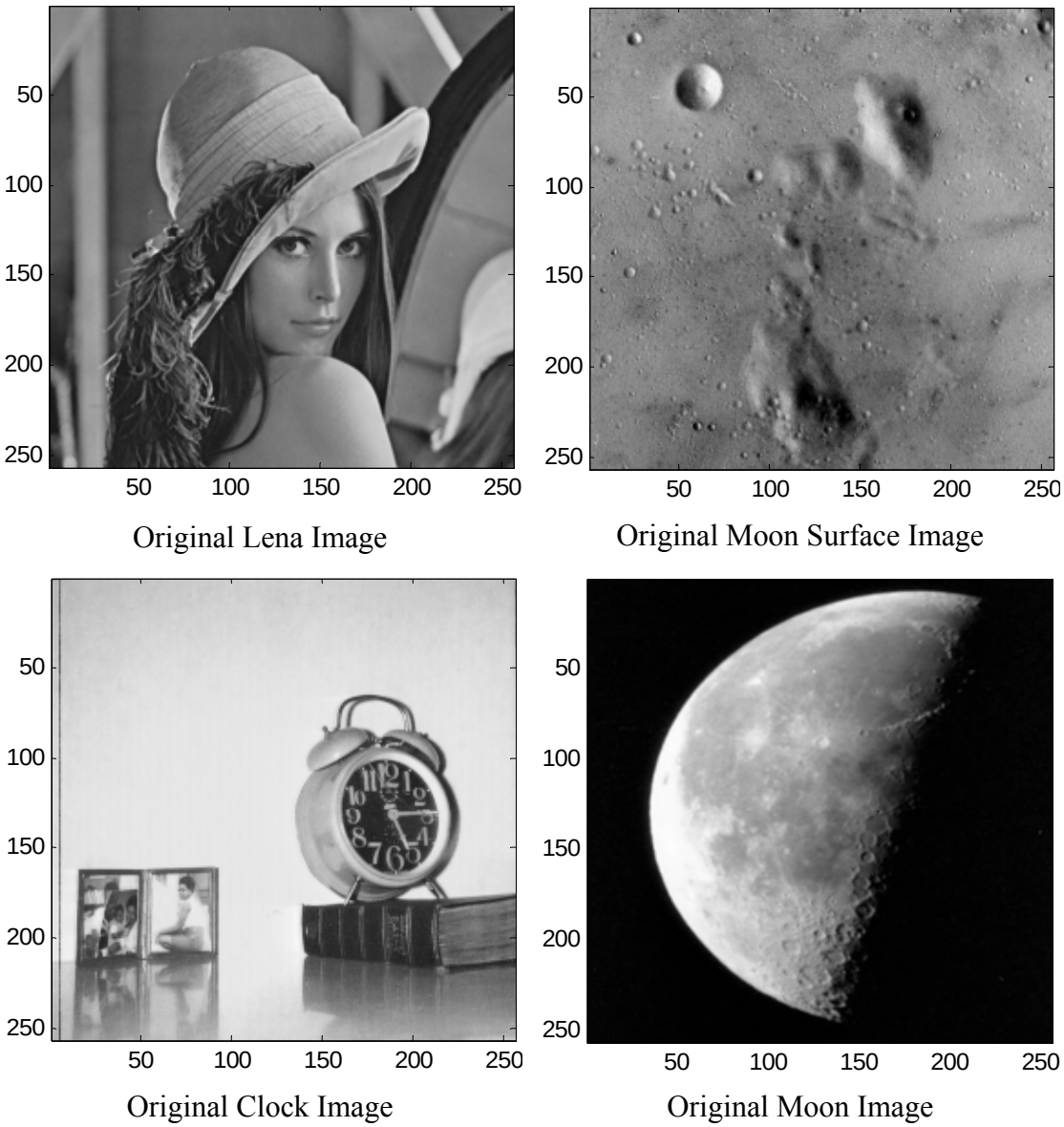


Figure 5.1 Original Test Images

From the images that are chosen, it could be easily observed that two of them opposite to each other one is too bright and the other is almost darkly image. The other two are chosen to symbolize the humans' life and there need to photo capturing, and the other one is symbolize the modern communications and science investigations and researches.

Figure 5.2 shows the original Lena image and four noise added Lena images with different values of Gaussian noise.

Original Lena Image

Image with 20 dB SNR

Image with 15 dB SNR

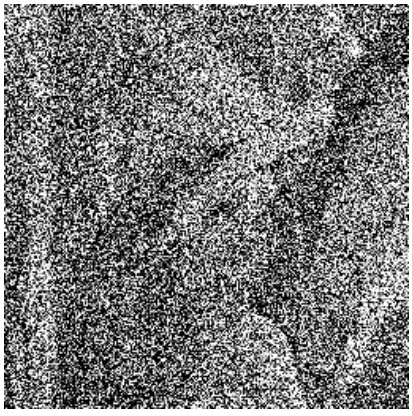


Image with 5 dB SNR

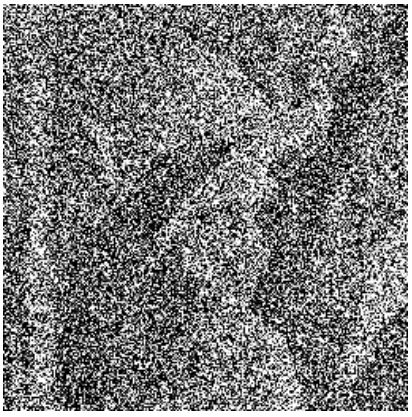
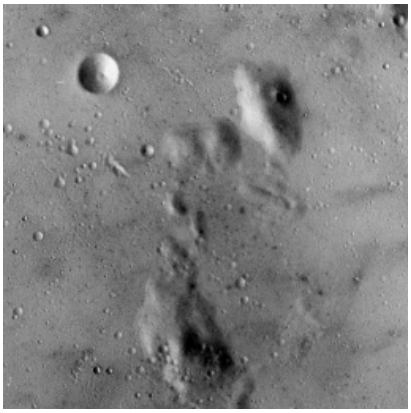


Image with 2 dB SNR

Figure 5.2 Lena Image and Noise Added Images with SNR = 2, 5, 15 and 20 dB

Figure 5.3 shows the original Moon surface image and four noise added Moon surface images with different values of Gaussian noise.



Original Moon Surface Image

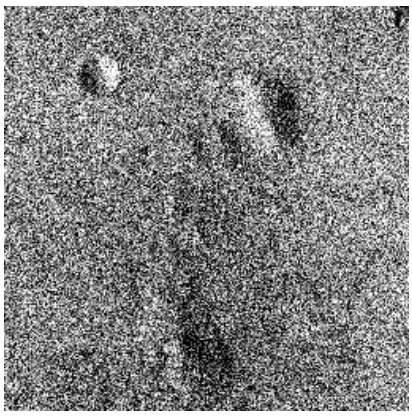


Image with 20 dB SNR

Image with 15 dB SNR

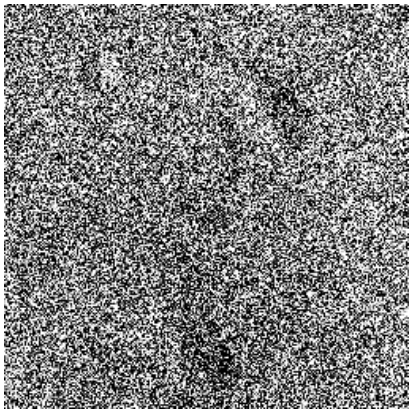


Image with 5 dB SNR

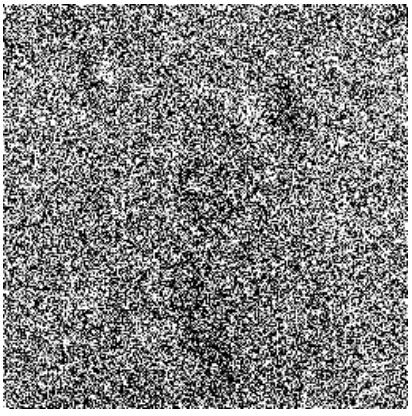


Image with 2 dB SNR

Figure 5.3 Moon Surface Image and Noise Added Images with SNR = 2, 5, 15 and 20 dB

Figure 5.4 shows the original Clock image and four noise added Clock images with different values of Gaussian noise.

Original Clock Image

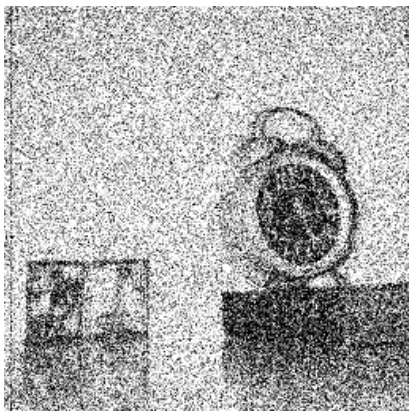


Image with 20 dB SNR

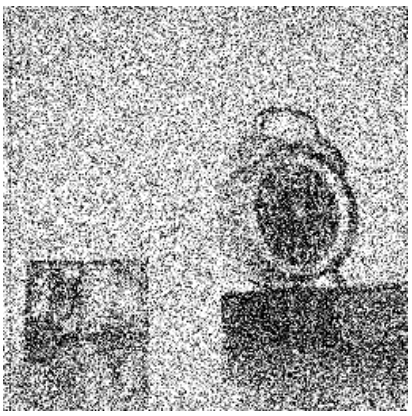


Image with 15 dB SNR

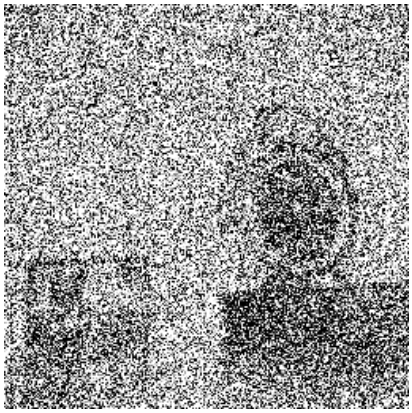


Image with 5 dB SNR

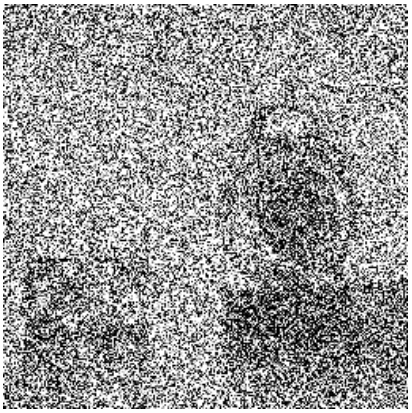


Image with 2 dB SNR

Figure 5.4 Clock Image and Noise Added Images with SNR = 2, 5, 15 and 20 dB

Figure 5.5 shows the original Moon image and four noise added Moon images with different values of Gaussian noise.

Original Moon Image

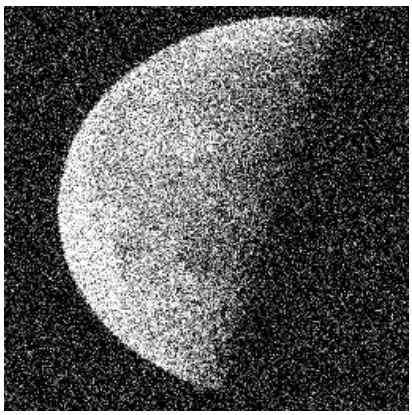


Image with 20 dB SNR

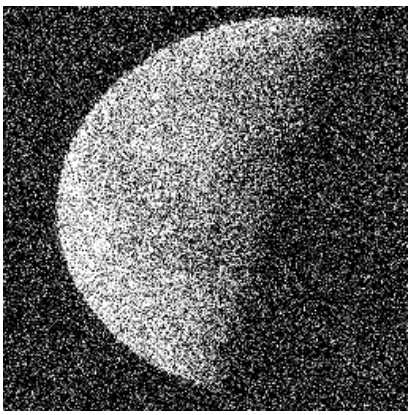


Image with 15 dB SNR

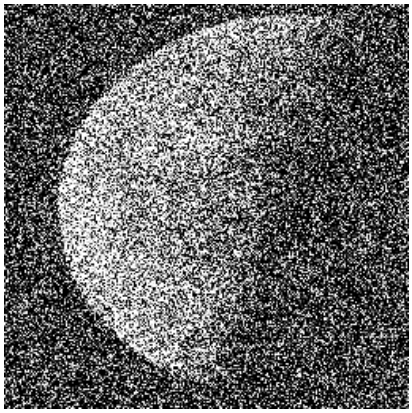


Image with 5 dB SNR

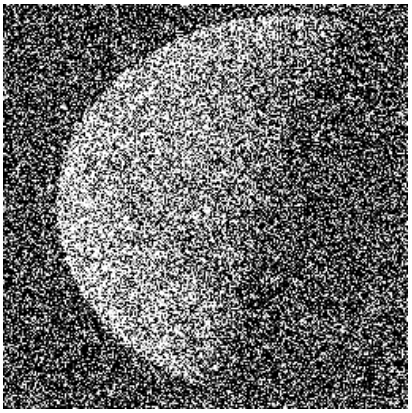


Image with 2 dB SNR

Figure 5.5 Moon Image and Noise Added Images with SNR = 2, 5, 15 and 20 dB

Figure 5.6 shows the original Lena image and the noise added Lena image with 2 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Lena Image

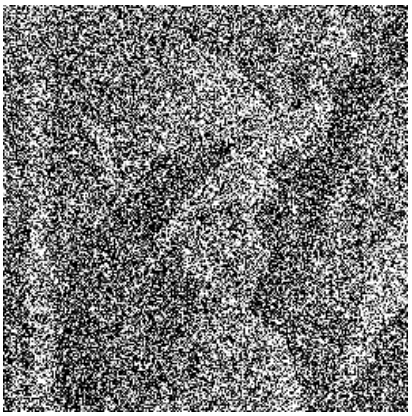


Image with 2 dB SNR

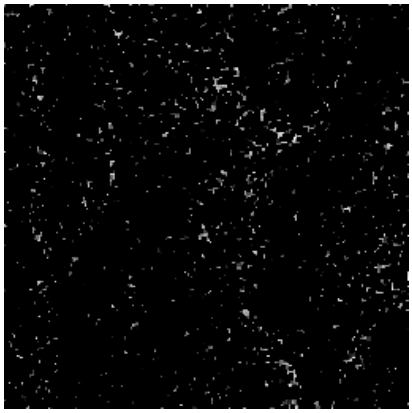


Image Restored By Harmonic Mean Filter

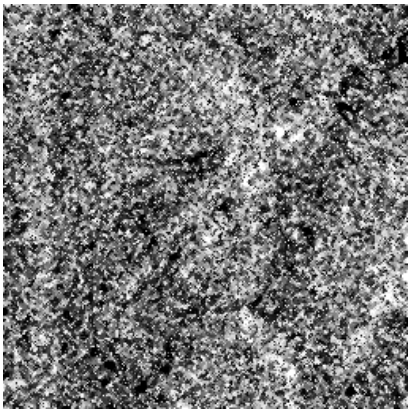


Image Restored By Adapt. Median Filter

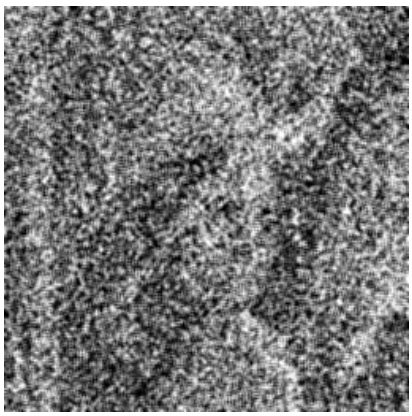


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.6 Lena Image, Noise Added Image with 2 dB SNR and Restored Images

Figure 5.7 shows the original Lena image and the noise added Lena image with 5 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Lena Image

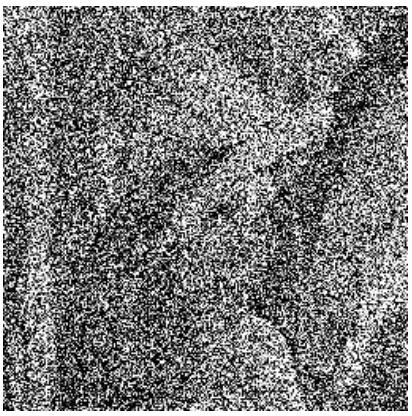


Image with 5 dB SNR

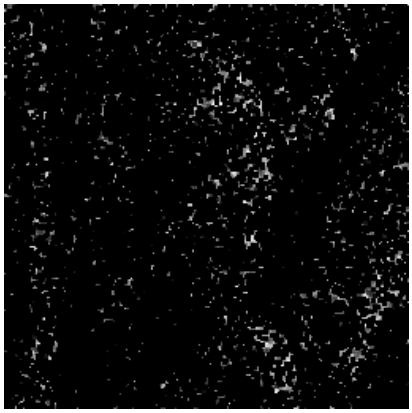


Image Restored By Harmonic Mean Filter

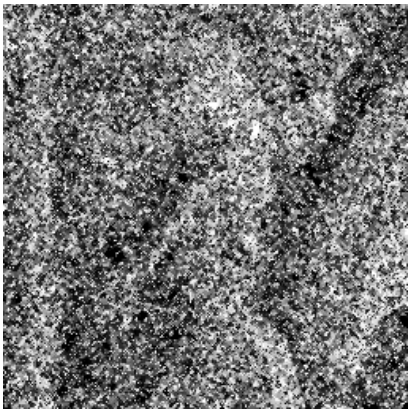


Image Restored By Adapt. Median Filter

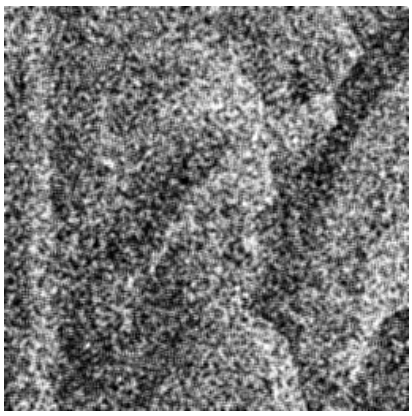


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.7 Lena Image, Noise Added Image with 5 dB SNR and Restored Images

Figure 5.8 shows the original Lena image and the noise added Lena image with 15 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Lena Image

Image with 15 dB SNR

Image Restored By Harmonic Mean Filter

Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.8 Lena Image, Noise Added Image with 15 dB SNR and Restored Images

Figure 5.9 shows the original Lena image and the noise added Lena image with 20 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Lena Image

Image with 20 dB SNR

Image Restored By Harmonic Mean Filter

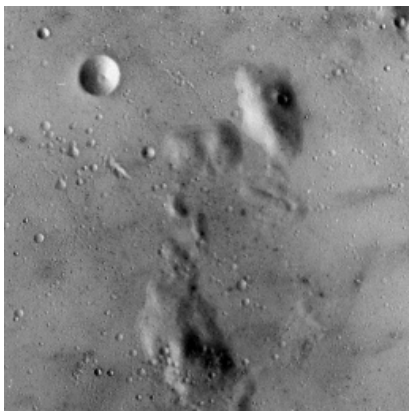
Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.9 Lena Image, Noise Added Image with 20 dB SNR and Restored Images

Figure 5.10 shows the original Moon surface image and the noise added Moon surface image with 2 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.



Original Moon Surface Image

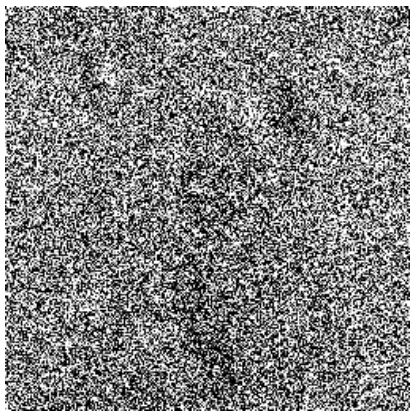


Image with 2 dB SNR

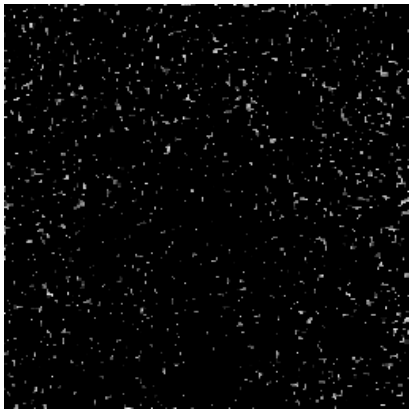


Image Restored By Harmonic Mean Filter

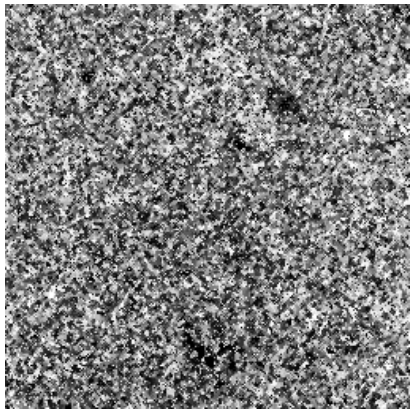


Image Restored By Adapt. Median Filter

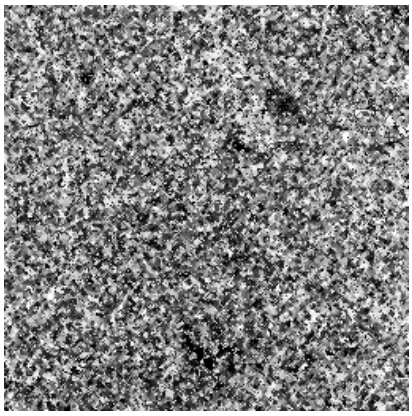
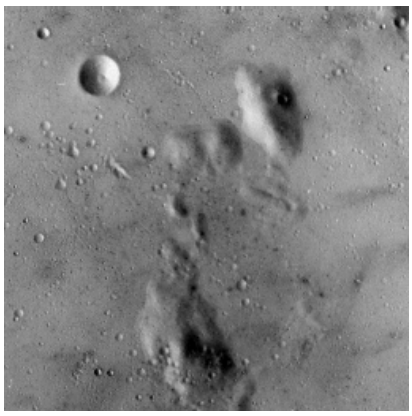


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.10 Moon Surface Image, Noise Added Image with 2 dB SNR and Restored Images

Figure 5.11 shows the original Moon surface image and the noise added Moon surface image with 5 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.



Original Moon Surface Image

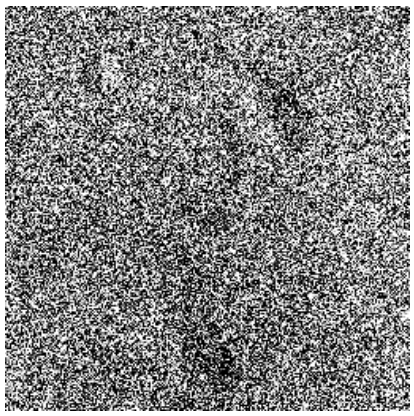


Image with 5 dB SNR

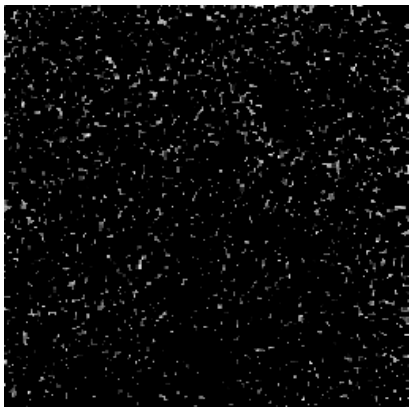


Image Restored By Harmonic Mean Filter

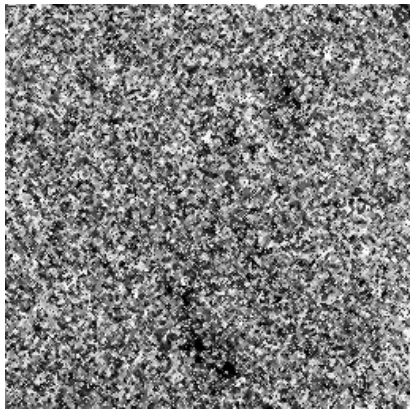


Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

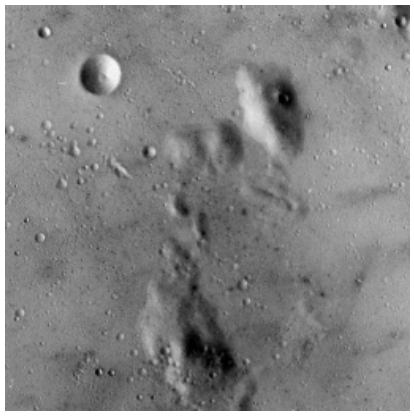
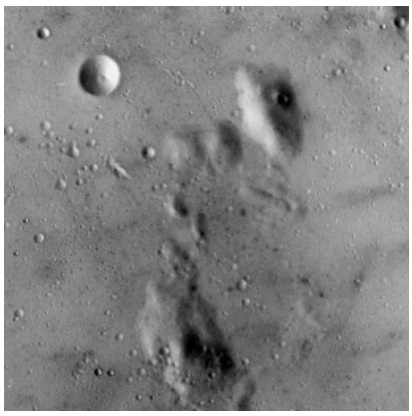


Image Restored By LDPC Coding

Figure 5.11 Moon Surface Image, Noise Added Image with 5 dB SNR and Restored Images

Figure 5.12 shows the original Moon surface image and the noise added Moon surface image with 15 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.



Original Moon Surface Image

Image with 15 dB SNR

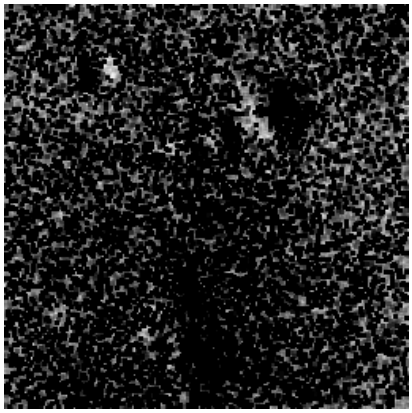


Image Restored By Harmonic Mean Filter

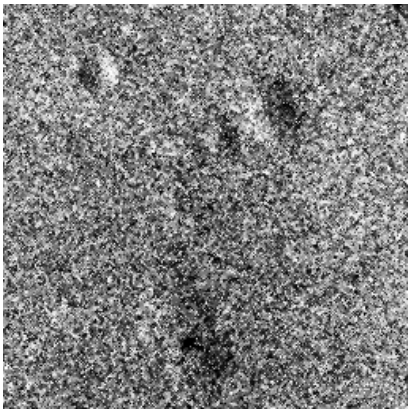


Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

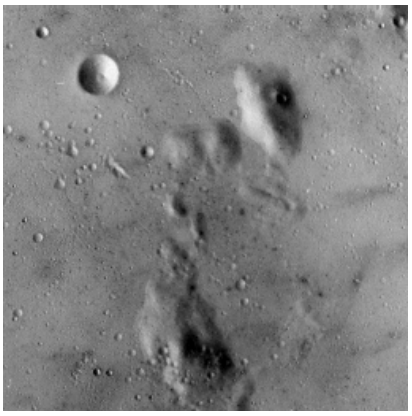
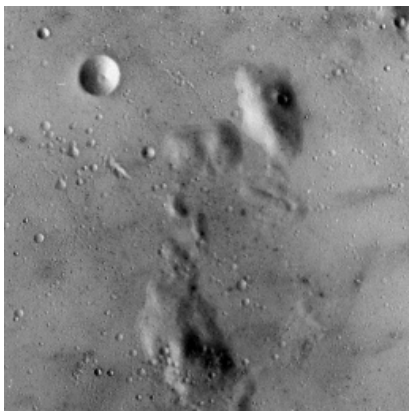


Image Restored By LDPC Coding

Figure 5.12 Moon Surface Image, Noise Added Image with 15 dB SNR and Restored Images

Figure 5.13 shows the original Moon surface image and the noise added Moon surface image with 20 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.



Original Moon Surface Image

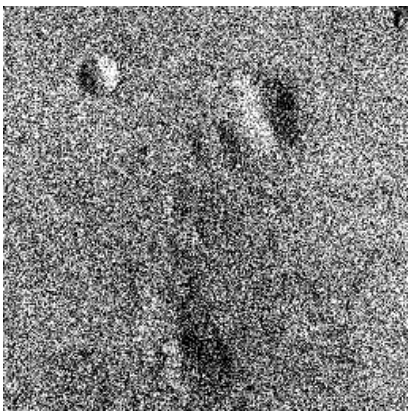


Image with 20 dB SNR

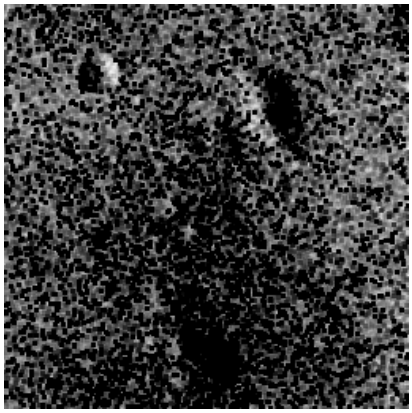


Image Restored By Harmonic Mean Filter

Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

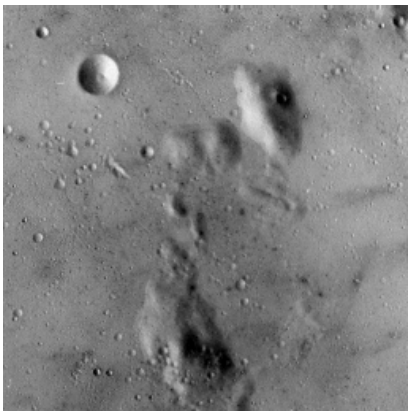


Image Restored By LDPC Coding

Figure 5.13 Moon Surface Image, Noise Added Image with 20 dB SNR and Restored Images

Figure 5.14 shows the original Clock image and the noise added Clock image with 2 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Clock Image

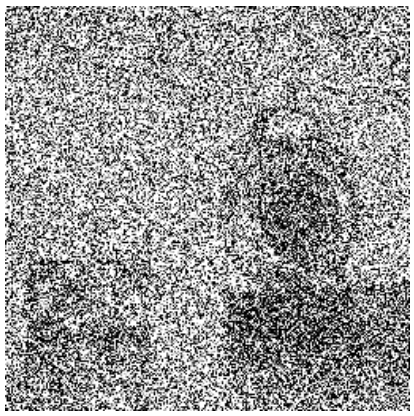


Image with 2 dB SNR

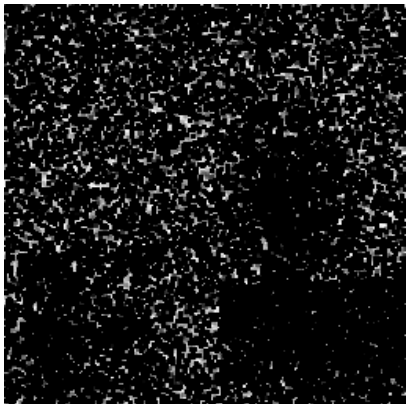


Image Restored By Harmonic Mean Filter

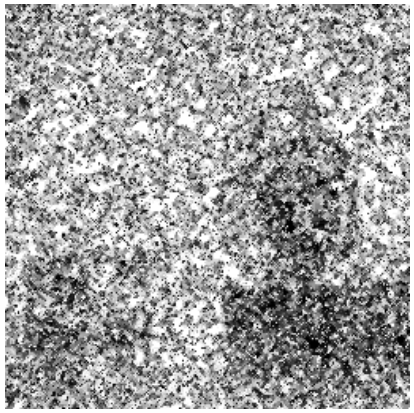


Image Restored By Adapt. Median Filter

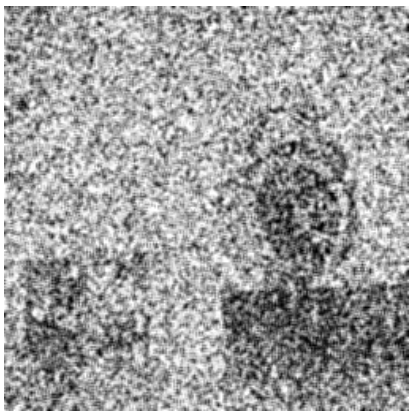


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.14 Clock Image, Noise Added Image with 2 dB SNR and Restored Images

Figure 5.15 shows the original Clock image and the noise added Clock image with 5 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Clock Image

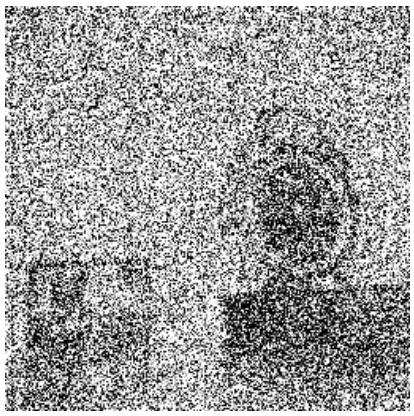


Image with 5 dB SNR

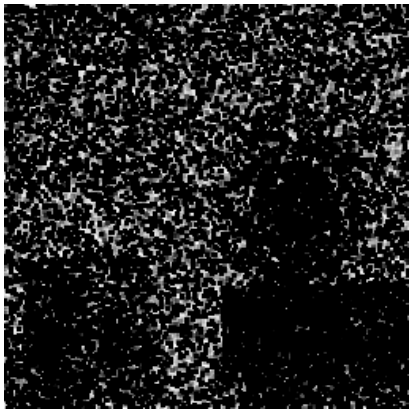


Image Restored By Harmonic Mean Filter

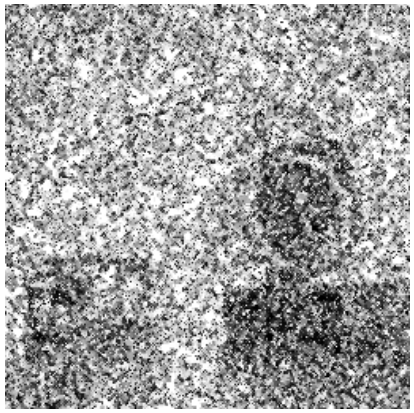


Image Restored By Adapt. Median Filter

Image Restored By Alpha-trimmed Filter

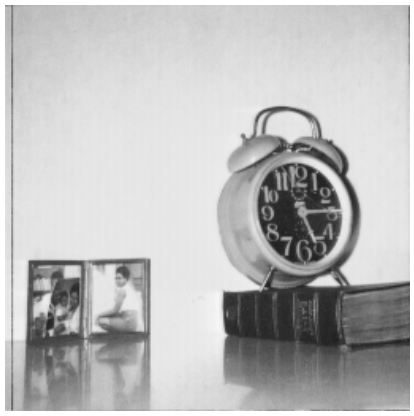


Image Restored By LDPC Coding

Figure 5.15 Clock Image, Noise Added Image with 5 dB SNR and Restored Images

Figure 5.16 shows the original Clock image and the noise added Clock image with 15 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Clock Image

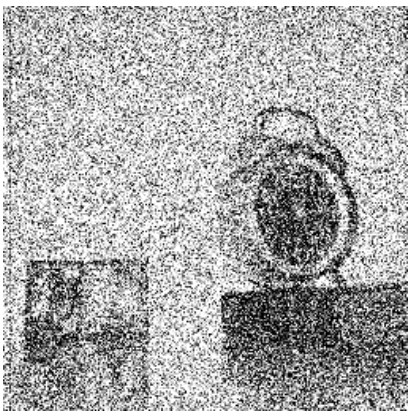


Image with 15 dB SNR

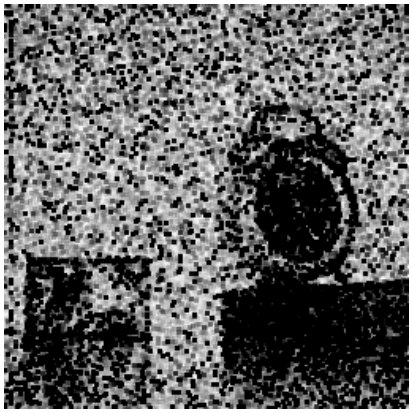


Image Restored By Harmonic Mean Filter

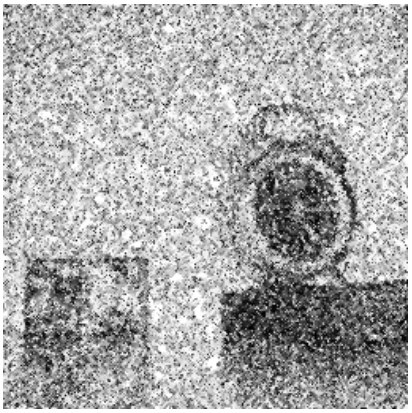


Image Restored By Adapt. Median Filter

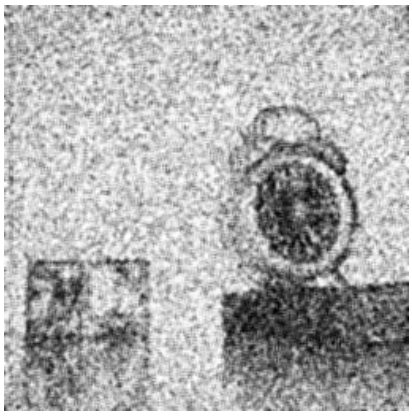


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.16 Clock Image, Noise Added Image with 15 dB SNR and Restored Images

Figure 5.17 shows the original Clock image and the noise added Clock image with 20 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Clock Image

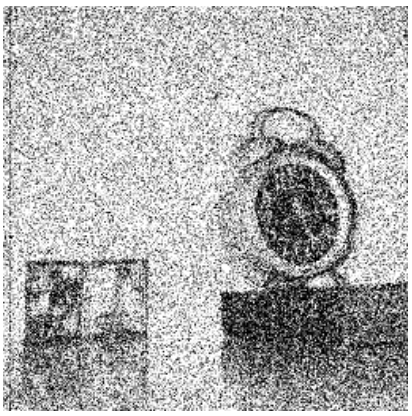


Image with 20 dB SNR

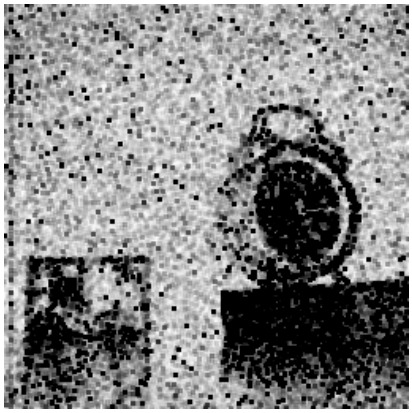


Image Restored By Harmonic Mean Filter

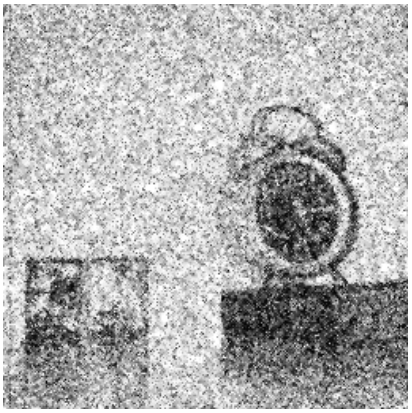


Image Restored By Adapt. Median Filter

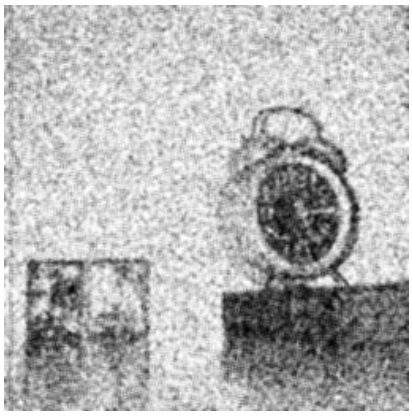


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.17 Clock Image, Noise Added Image with 20 dB SNR and Restored Images

Figure 5.18 shows the original Moon image and the noise added Moon image with 2 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Moon Image

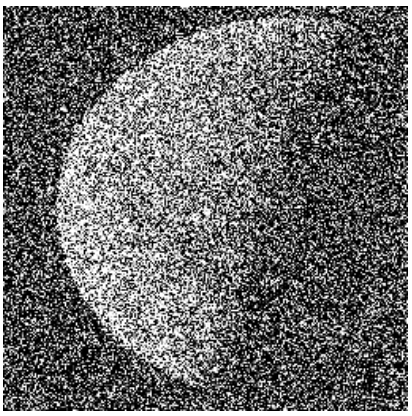


Image with 2 dB SNR

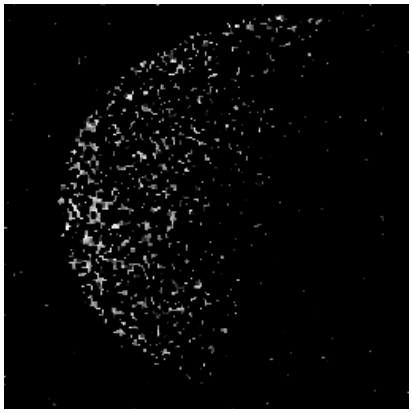


Image Restored By Harmonic Mean Filter

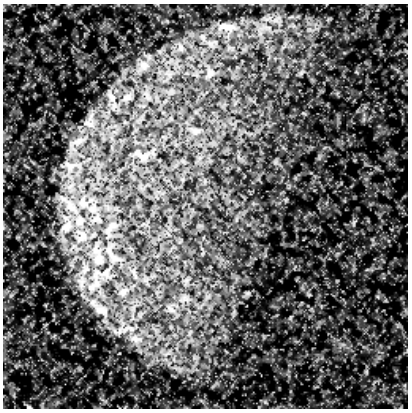


Image Restored By Adapt. Median Filter

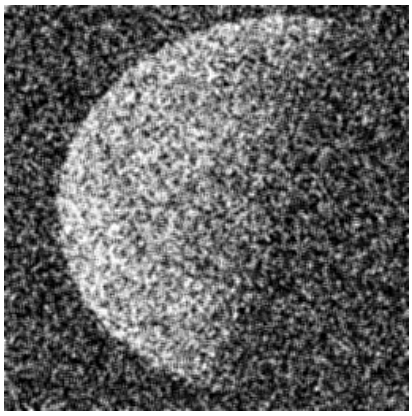


Image Restored By Alpha-trimmed Filter

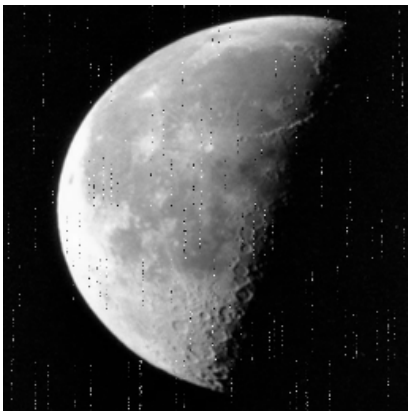


Image Restored By LDPC Coding

Figure 5.18 Moon Image, Noise Added Image with 2 dB SNR and Restored Images

Figure 5.19 shows the original Moon image and the noise added Moon image with 5 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Moon Image

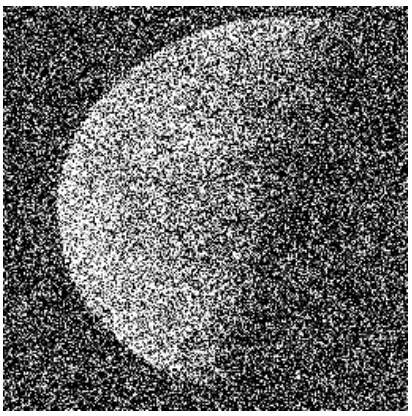


Image with 5 dB SNR

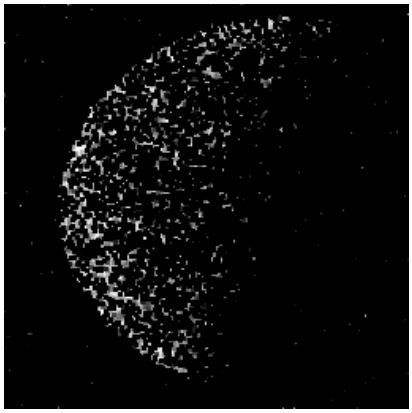


Image Restored By Harmonic Mean Filter

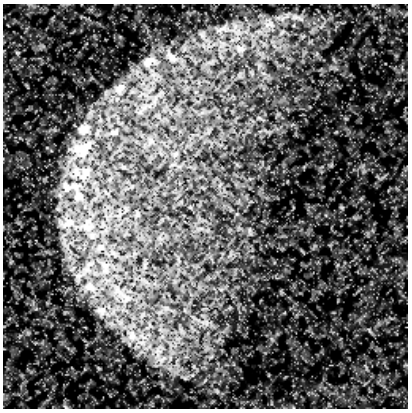


Image Restored By Adapt. Median Filter

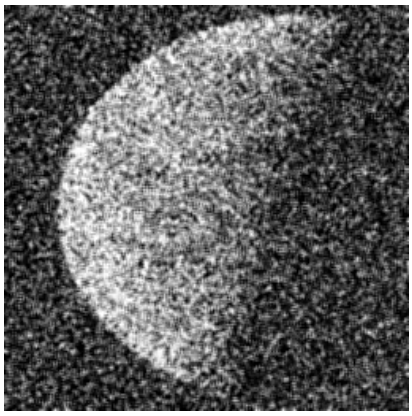


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.19 Moon Image, Noise Added Image with 5 dB SNR and Restored Images

Figure 5.20 shows the original Moon image and the noise added Moon image with 15 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Moon Image

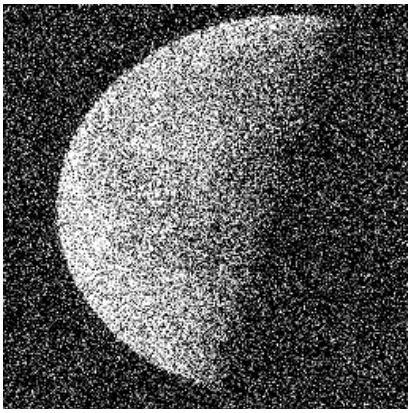


Image with 15 dB SNR

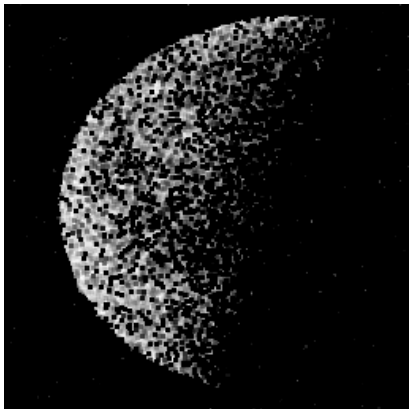


Image Restored By Harmonic Mean Filter

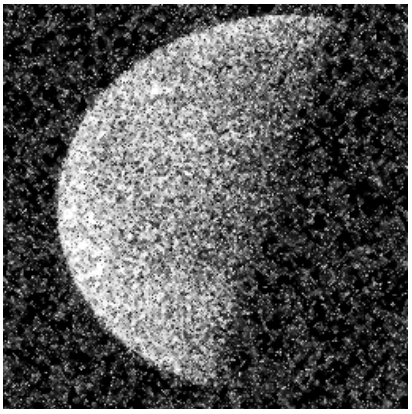


Image Restored By Adapt. Median Filter

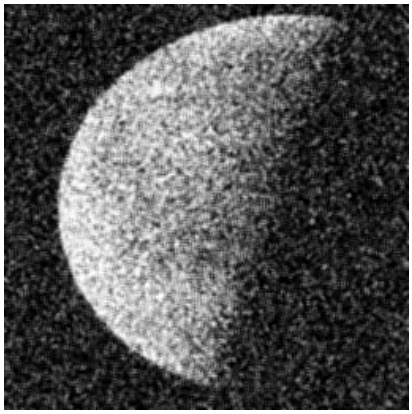


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.20 Moon Image, Noise Added Image with 15 dB SNR and Restored Images

Figure 5.21 shows the original Moon image and the noise added Moon image with 20 dB SNR. Subsequent images are the results of image restoration by various methods including LDPC coding.

Original Moon Image

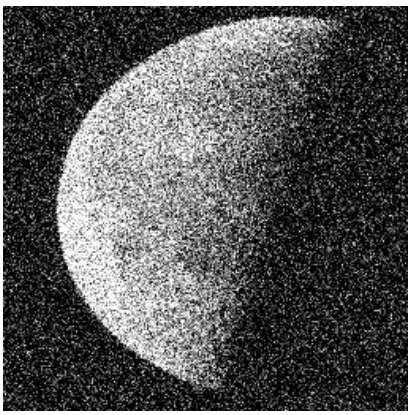


Image with 20 dB SNR

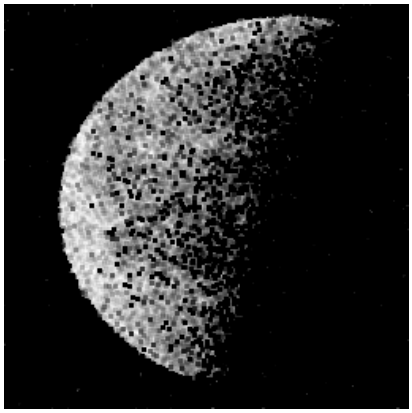


Image Restored By Harmonic Mean Filter

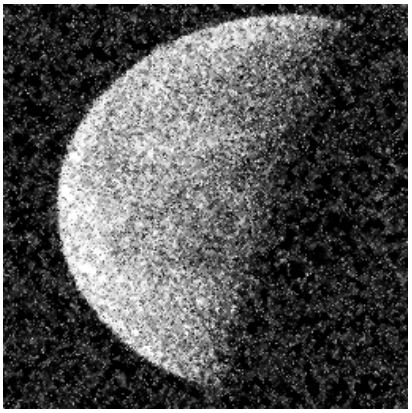


Image Restored By Adapt. Median Filter

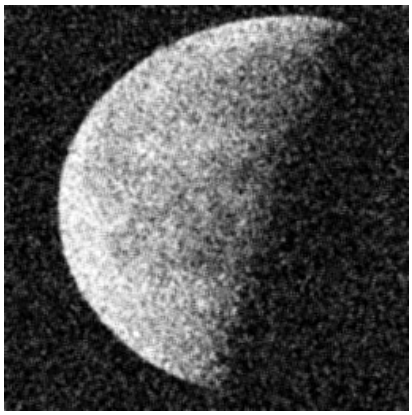


Image Restored By Alpha-trimmed Filter

Image Restored By LDPC Coding

Figure 5.21 Moon Image, Noise Added Image with 20 dB SNR and Restored Images

5.2.2 Comparison of Methods Using PSNR Values

The PSNR results were classified as Super Quality, Ultra Low Loss, Low Loss, Medium Loss and High Loss as shown in Table 5.1.

Table 5.1 Classification Values for PSNR Comparison

PSNR Value	Classification
0↔20	High Loss
20↔30	Medium Loss
30↔40	Low Loss
40↔60	Ultra Low Loss
>60	Super Quality

The aim of this classification is to help in determining the performance of the restoration methods.

When analyzing the PSNR values of each restoration method some of the following should be noted:

- The amount of noise added in this analysis was high because of good performance of LDPC error correction codes.
- While setting the noise amount, the SNR is calculated in two different ways depending on if coding has been used.

$$\text{coding } SNR = \frac{1}{2 \times R \times \sigma^2}$$

$$\text{no coding } SNR = \frac{1}{2 \times \sigma^2}$$

where σ^2 is the noise variance, and R is the coding rate.

Table 5.2 shows the standard deviation of noise for various SNR values which are used in the simulation.

Table 5.2 Standard Deviation of Noise for Various SNR Values

SNR Values	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
σ	0.79435	0.56237	0.17783	0.1

The PSNR values for four restoration methods with different noise values presented in Table 5.3.

Table 5.3 PSNR Result in (dB) Using 2, 5, 15 and 20 dB SNR

Harmonic Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	10.2177	8.6169	7.2291	7.1209
Moon Surface	9.2662	7.4223	6.1334	6.0151
Clock	10.8885	7.1732	3.4579	3.0543
Moon	14.0616	11.3902	8.5559	8.2528

Alpha-trimmed Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	19.4802	17.5582	14.6097	13.9625
Moon Surface	19.3203	17.5313	15.1494	14.6104
Clock	19.6093	17.6169	13.6254	12.7253
Moon	19.3792	17.0033	12.7517	11.7322

Adaptive Median Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	14.4235	12.8148	11.1175	10.8159
Moon Surface	14.1549	12.8543	11.4681	11.2681
Clock	14.9256	13.0376	10.5405	10.1413
Moon	15.2513	13.1911	10.3217	9.8438

LDPC Coding

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	No Noise	No Noise	No Noise	44.8750
Moon Surface	No Noise	No Noise	No Noise	44.8580
Clock	No Noise	No Noise	No Noise	44.9227
Moon	No Noise	No Noise	No Noise	48.2415

Noise ratio and the quality of the reconstructed image are important factors. When an image data is being sent through the communication channel, the quality of the restored

image is as important as the rate of transmission. In this section, the PSNR values of the reconstructed images are compared with each other. The results show that, LDPC codes should be preferred when restorating images corrupted by Gaussian noise. Table 5.4 shows the quality comparison of PSNR values given in Table 5.1.

Table 5.4 Quality Comparison of PSNR Results

Harmonic Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	High Loss	High Loss	High Loss	High Loss
Moon Surface	High Loss	High Loss	High Loss	High Loss
Clock	High Loss	High Loss	High Loss	High Loss
Moon	High Loss	High Loss	High Loss	High Loss

Alpha-trimmed Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	High Loss	High Loss	High Loss	High Loss
Moon Surface	High Loss	High Loss	High Loss	High Loss
Clock	High Loss	High Loss	High Loss	High Loss
Moon	High Loss	High Loss	High Loss	High Loss

Adaptive Median Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	High Loss	High Loss	High Loss	High Loss
Moon Surface	High Loss	High Loss	High Loss	High Loss
Clock	High Loss	High Loss	High Loss	High Loss
Moon	High Loss	High Loss	High Loss	High Loss

LDPC Coding

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	Super Quality	Super Quality	Super Quality	U. Low Loss
Moon Surface	Super Quality	Super Quality	Super Quality	U. Low Loss
Clock	Super Quality	Super Quality	Super Quality	U. Low Loss
Moon	Super Quality	Super Quality	Super Quality	U. Low Loss

5.2.3 Comparison of Methods Using Contrast Criteria

Contrast is widely used in image processing when analyzing images. It could also be used to see if a restored image has lost too much detail with respect to the original image. If the contrast of the restored image is very different, then the method of restoration is not satisfactory. Table 5.5 presents the results of the reconstruction method according to the contrast criteria. It can be observed that the results are too perfect for LDPC codes. The second best results are for Alpha-trimmed filter but they are still not good enough.

5.2.4 Comparison of Methods Using Brightness

Brightness, just like contrast, can be used to analyze the restored images. Table 5.5 presents the results of the reconstruction methods according to the brightness criteria. The brightness of the images restored using LDPC codes are almost the same as the brightness of the original images. The results were too perfect for LDPC codes. Table 5.7 shows anlysis of brightness values given in Table 5.6 which represent the differences between the brightness of original images and brightness of reconstructed images.

5.2.5 Comparison of Methods Using Processing Time

Many factors affect the processing time in communication system such as data processing software and hardware structure. The time of processing should be as minimum as possible. Table 5.8 presents a comparison of restoration methods according to the processing time. It is observed that the highest processing time is for the LDPC codes. The reasons for this are discussed in the analysis and discussion (section 5.3).

Table 5.5 Comparison of Contrast Criteria (Original Image – Reconstructed Image)

Harmonic Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	-55.2126	-67.3664	-87.455	-94.8218
Moon Surface	-68.6651	-92.0029	-122.5304	-127.6989
Clock	-17.4452	-21.717	-42.6385	-49.0335
Moon	5.7861	3.6764	-2.0364	-9.684

Alpha-trimmed Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	-3.9068	-4.9526	-4.5595	-3.8746
Moon Surface	-16.3054	-22.1165	-33.0279	-35.2813
Clock	11.519	17.2594	33.6314	36.8755
Moon	22.3022	30.6731	56.2493	62.6678

Adaptive Median Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	-24.618	-32.4542	-43.4593	-45.7405
Moon Surface	-47.7852	-58.3736	-74.979	-76.7486
Clock	4.9307	3.6022	-3.0668	-5.4142
Moon	16.7232	19.7787	22.2429	20.9369

LDPC Coding

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	0	0	0	-1.4750
Moon Surface	0	0	0	-1.4943
Clock	0	0	0	-0.1635
Moon	0	0	0	0.6932

Table 5.6 Brightness Results of HMF, AMF, AdMF and LDPC codes under different level of noise (2, 5, 15 and 20 dB SNR)

Harmonic Mean Filter

	Original Brightness	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	105.7428	41.5529	24.5973	6.3508	5.0125
Moon Surface	127.7600	57.6814	32.2889	8.6162	5.8828
Clock	185.9803	131.7579	96.9836	31.5073	21.3196
Moon	64.5697	37.8661	26.4056	7.7747	5.2794

Alpha-trimmed Mean Filter

	Original Brightness	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	105.7428	107.5943	109.8575	113.5447	116.7933
Moon Surface	127.7600	128.5522	127.5453	127.4515	127.8204
Clock	185.9803	178.6808	174.3141	162.1475	157.5082
Moon	64.5697	76.7516	80.4172	92.2404	96.6608

Adaptive Median Filter

	Original Brightness	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	105.7428	109.4716	112.3919	114.9577	117.1010
Moon Surface	127.7600	128.4747	127.4939	127.5040	127.0630
Clock	185.9803	174.4797	169.0919	160.3359	158.0781
Moon	64.5697	77.2850	80.6674	90.4823	93.2432

LDPC Codes

	Original Brightness	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	105.7428	105.7428	105.7428	105.7428	105.7586
Moon Surface	127.7600	127.7600	127.7600	127.7600	127.7770
Clock	185.9803	185.9803	185.9803	185.9803	185.6151
Moon	64.5697	64.5697	64.5697	64.5697	65.0228

Table 5.7 Brightness Analysis of HMF, AMF, AdMF and LDPC codes under different level of noise (Original Image – Reconstructed Image)

Harmonic Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	64.1899	81.1455	99.392	100.73
Moon Surface	70.0786	95.4711	119.144	121.877
Clock	54.2224	88.9967	154.473	164.661
Moon	26.7036	38.1641	56.795	59.2903

Alpha-trimmed Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	-1.8515	-4.1147	-7.8019	-11.051
Moon Surface	-0.7922	0.2147	0.3085	-0.0604
Clock	7.2995	11.6662	23.8328	28.4721
Moon	-12.182	-15.848	-27.671	-32.091

Adaptive Median Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	-3.7288	-6.6491	-9.2149	-11.358
Moon Surface	-0.7147	0.2661	0.256	0.697
Clock	11.5006	16.8884	25.6444	27.9022
Moon	-12.715	-16.098	-25.913	-28.674

LDPC Coding

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	0	0	0	-0.0158
Moon Surface	0	0	0	-0.017
Clock	0	0	0	0.3652
Moon	0	0	0	-0.4531

Table 5.8 Processing Time (in seconds)

Harmonic Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	1.2969	1.2344	1.2344	1.2969
Moon Surface	1.2813	1.2500	1.2188	1.2031
Clock	1.2500	1.2500	1.2344	1.2031
Moon	1.2500	1.2500	1.2344	1.2344

Alpha-trimmed Mean Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	1.2344	1.2656	1.2344	1.2656
Moon Surface	1.2500	1.2656	1.2188	1.2500
Clock	1.2656	1.2656	1.2813	1.2500
Moon	1.2813	1.2656	1.2969	1.2656

Adaptive Median Filter

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	1.4688	1.3906	1.4063	1.4375
Moon Surface	1.4688	1.4531	1.4063	1.4375
Clock	1.5000	1.4375	1.4063	1.4375
Moon	1.4375	1.4531	1.4531	1.4375

LDPC Coding

	20 *dB* SNR	15 *dB* SNR	5 *dB* SNR	2 *dB* SNR
Lena	13728.5166	13672.4119	13992.4844	19982.5469
Moon Surface	13699.4176	13534.6314	12541.5466	19708.6719
Clock	13427.2348	13536.7516	13708.6729	17541.1563
Moon	13285.6543	13525.9719	13982.1521	18436.1719

Time results were obtained using Matlab 7 on a 3.01 GHz PC with 512 Mb of RAM, running Windows XP

5.3 Analysis and Discussion

It is shown that LDPC coding removes the additive white Gaussian noise successfully in most of the images.

In image restoration high accuracy of restoration is very important. In coding, this is determined by various parameters such as parity check matrix *H, G* matrix, and number of iterations, and for other restoration techniques by the window size, and maximum allowable value $S_{\max}$, and also by the amount of noise.

The PSNR criteria shows that the highest value is obtained when LDPC coding is used, (classified as super quality) while the result of the Alpha-trimmed filter is the second best among the three filters.

The contrast criteria shows that there is almost no change in contrast value between the restored images and the original images for LDPC coding while some change has been occurred for the restoration methods. The change in contrast is increased by increasing the amount of noise but this increase is the last for LDPC coding.

The brightness criteria also shows that LDPC coding is successful in the removal of white Gaussian noise as for all set test images in all amounts of noise, the brightness of restored images and the original images are almost the same. The second best result is again using Alpha-trimmed filter in most cases. Note that sometimes the result of the other two methods are too close to the brightness of the original image, not because they have good quality of filtering but because the number of black pixels are too many as in the moon image.

LDPC codes are slow compared to the other restoration methods. The reason is the number of steps that the data has to go through for restoration, such as converting the image data from decimal to binary numbers and then restoring them to the suitable form that the encoder can use and restoring the restored data. It is believed those simulation could be done in C language in much less time compared to Matlab language. The efficiency of the written program could be improved for faster processing time. (Note that no Matlab toolbox commands are used for the LDPC encoder and decoder). A final factor affecting the LDPC codes is the amount of iterations needed for decoding.

Experimental results have shown that for LDPC coding, the presence of details in the images never effected from white Gaussian noise and the accuracy of restoration was 100% in the selected values of noise even when the other methods in most cases did not perform well to the LDPC codes. The second best result in most comparison criteria was for the Alpha-trimmed filter.

The performance of the system has been illustrated by the implementation using the other database such as Lena image and Moon image observing that they are famous for the researchers on topic of image processing field, which contains images with small variations in illumination and orientation.

The efficiency of the method suggested in this thesis has been shown to have 100% filtering accuracy with high value of noise added.

5.4 Summary

The software application which was developed has been presented and demonstration of applying the restoration methods has been shown.

Different restoration methods are applied to different images and all the reconstructed images are analyzed using the comparison criteria in order to decide upon the optimum restoration method.

Based on the work and analysis, the LDPC decoder is found to be the ideal restoration method in this work.

The experimental results were also discussed in this chapter, which demonstrated the successful implementation of the developed method.

CONCLUSION

LDPC codes provide coding gains when used in communication systems. Due to their iterative decoding structure, coding gains could be increased by increasing the number of iterations of each block of data in the decoder, although the amount of noise decreases by each of iterations. The LDPC decoder performance is also affected by the size of the G and H matrices. Larger H sizes provide larger coding gains, but also increase the latency of the overall system. For power-limited communication systems where latency is a critical issue, it is desirable to have moderate good H and G size for the encoder and decoder of LDPC codes with a reasonable number of decoder iterations. LDPC codes are also known to provide significant gains for AWGN channels. LDPC codes are currently being considered as part of a standard for future communication systems such as in medical imaging, deep-space and multimedia.

Removing or eliminating the noise from image data is one of the difficult problems. This thesis presents a system that corrects data in spatial domain using LDPC error correction codes. The white Gaussian noise is removed from images without affecting the pixel values. In order to analyze the accuracy and efficiency of this system, the results are compared using various criteria to some of the well-known filters used for image restoration.

The analysis criteria introduced can be used for the application of LDPC codes to image transmission and restoration. The criteria consist of visual inspection as well as theoretical computation. The results show better restoration of images using LDPC codes. However, the delay introduced is substantially larger as compared to other filtering techniques utilized.

The idea of future work for the presented work is to make the system dealing with the 3 D images (colored) not only with images in grayscale dimension. Also, for more flexibility, make the system work with bigger sizes of images.

The idea of future work also on LDPC error control codes is to give the flexibility of the structure algorithms of LDPC codes to can be usable or dealing with all kind of noise. Because they show efficient performance to correct the data corrupted as the results of this work is shown.

REFERENCES

[1] Couch II Leon W. (2007). Digital and Analog Communication Systems. Seventh Edition. New Jersey, Prentice-Hall International.

[2] Barry, J. Lee, E. and Messerschmitt, D. (2003). Digital Communication. Third Edition. Boston Kluwer Academic Publishers.

[3] Proakis J. and Salehi M. (2002). Communication Systems Engineering. New Jersey, Prentice-Hall.

[4] Hendrix H. (2002). Viterbi Decoding Techniques. Application Report, Texas Instruments.

[5] Pietrobon S.S. and Giles T. (1998). "Improving the constituent codes of turbo encoders". IEEE Global Telecommunications Conference, Sydney, Australia, Vol. 6, pp. 3525-3529.

[6] Fan J. L. (2001). Constrained Coding and Soft Iterative Decoding. Boston Kluwer Academic Publishers.

[7] Wicker S. B. (1995). Error Control Systems for Digital communication and Storage. New Jersey, Prentice-Hall.

[8] Proakis J. Digital Communications. Third Edition. New York, McGraw-Hill.

[9] Berrou C., Glavieux A. and Thitimajshima P. (May 1993) "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," Proceedings of ICC 1993, Geneva, Switzerland, pp. 1064-1070.

[10] Barry J. R., Lee E. A., and Messerschmitt D. G. (2003). Digital Communication. Third Edition. Boston Kluwer Academic Publishers.

[11] Kim Jin Young, Poor H. Vincent (2001). "Turbo-Coded Optical Direct-Detection CDMA System with PPM Modulation". Journal of LightWave Technology. VOL. 19, NO. 3, pp.312-323.

[12] Haykin S. (2001). Communication Systems. Fourth Edition. New York, John Wiley & Sons.

[13] Gallager R. (1963). Low-Density Parity-Check Codes. Cambridge, Massachusetts, MIT Press.

[14] Ryan W. (April 2001). An introduction to Low-Density Parity Check codes, Lecture notes. University of Arizona.

[15] Tanner R. M. (1981). “A recursive approach to low complexity codes”. IEEE Trans. Inform. Theory, pp. 533-547.

[16] MacKay D. J. C. (1999). “Good error-correcting codes based on very sparse matrices”. IEEE Trans. Inform. Theory. Vol. 45, pp. 399-431.

[17] Luby M. G., Mitzenmacher M., Shokrollahi M. A., and Spielman D. A. (1998). “Analysis of low density codes and improved designs using irregular graphs”. In Proc. of the 30th Annual ACM Symposium on the Theory of Computing, pp. 249-258.

[18] Wiberg N. (1996). “Codes and decoding on general graphs”. Ph.D. dissertation, University of Linköping, Sweden.

[19] Gonzalez Rafael C. and Woods Richard E. (2002). Digital Image Processing. Second Edition. New Jersey, Prentice-Hall.

[20] Gonzalez Rafael C., Woods Richard E. and Eddins Steven L. (2004). Digital Image Processing Using MATLAB. New Jersey, Prentice-Hall.

[21] University of Southern California, Signal and Image Processing Institute, “Volume 3: Miscellaneous”, “http://sipi.usc.edu/database”, Retrieved September 3, 2006.

[22] Christophe E., Leger D. And Mailhes C. (2005). “Comparison and Evaluation of Quality Criteria for Hyperspectral Imagery”. In Proc. of the SPIE 17th Annual Symposium of Electronic Imaging, pp.1-9.

[23] Khashman A. and Dimililer K. (2005), “Comparison Criteria for Optimum Image Compression”, In Proc. of IEEE International Conference on 'Computer as a Tool' (EUROCON'05), Serbia & Montenegro

APPENDICES

Appendix I Matlab Source Code of AWGN Channel Simulation

```
    %%%%%%%%% SIMULATION OF LDPC CODES OVER AWGN CHANNEL %%%%%%%%%%

% VARIABLES

clear all;clc;                %Clean the previous results
m=256;                        %Number of rows
n=512;                        %Number of columns
max_iterations=100;           %Maximum number of iterations
rate=(n-m)/n;                 %Rate of decoder

%LOADING OF CODING MATRICES

load ldpc256_512_gen.txt;          %Load of A inverse B
load ldpc256_512_pchk.txt;         %Load of H matrix
AinvB_256_256=ldpc256_512_gen;     %
H_256_512=ldpc256_512_pchk;        %
load codeword_order.txt;           %Load of codeword order
cnt=1;
for i=1:26
    for j=1:20;
    new_codeword_order(cnt)=codeword_order(i,j);%Sorting of codeword
        cnt=cnt+1;
    end
end
codeword_order_1_512=new_codeword_order(1:512);

%RANGE OF THE SIMULATION FOR NOISE VALUES

 SNR_db=0:0.5:2;                %SNR range
for o=1:length(SNR_db)          %Calculating of SNR
  disp('SNR');SNR_db(o)         %
  SNR=10^(SNR_db(o)/10);        %
  variance=1/(2*SNR*rate)       %Calculating of Variance
    err=0;                      %
    DATA_SUM=0;                 %Initial value of errors
    while (err<1000)            %Number of errors

        for i=1:(n-m)           %
            original_data(i)=rand;        %Random generate of data

            if (original_data(i)>0.5)     %Normalizing the data
              original_data(i)=1;         %
            else                          %
              original_data(i)=0;         %
            end                           %
        end                               %

%ACTIVATION OF THE ENCODER

yi=ldpc_encoder(m,n,AinvB_256_256,codeword_order_1_512,original_data);

%CONSIDERATION OF AWGN CHANNEL
        DATA_SUM=DATA_SUM+256;
   for i=1:n
```

```
     yi(i)=yi(i)+(sqrt(variance)*randn);%Function of AWGN channel
    end

%ACTIVATION OF THE DECODER

est_data=ldpc_decoder1(m,n,max_iterations,variance,yi,codeword_order_1
_512,H_256_512);

%ELIMINATING THE EXTRA MESSAGE BITS ADDED BY ENCODER

    for i=1:(n-m)
     if(original_data(i)~=est_data(codeword_order_1_512(m+i)+1))
      err=err+1;
     end
    end

   err
   end

 %CALCULATING OF BER

   ber(o)=err/DATA_SUM
   q(o)=0.5*erfc(sqrt(SNR));
end

%VARIABLES FOR PLOTING GRAPH

semilogy(SNR_db,ber,'-',SNR_db,q,'--')
title('AWGN channel')
xlabel('S/N(dB)'),ylabel('BER')
legend('AWGN','LDPC AWGN')
grid
set(gca,'XTick',[0;0.5;1;1.5;2])
axis([0 2 10e-5 1])

        %%%%%%%%%%%%%%%%%%%% LDPC ENCODER %%%%%%%%%%%%%%%%%%%%

%FUNCTION TO CALL THE ENCODER

function codeword=ldpc_encoder(m,n,AinvB,codeword_order,s_tx)
%MULTIBLICATION OF ORIGINAL DATA BY AINV B
c_tx=rem(AinvB*s_tx',2);        % Applying the binary mult.for c_tx
x_tx=[c_tx' s_tx];              % putting the x_tx in a new form

%NORMALIZING OF NEW MATRIX

for (i=1:n)                      %
  if (x_tx(i)==1)                %
    x_tx(i)=-1;                  % converting the c_tx to 1 or -1
  else                           %
    x_tx(i)=1;                   %
  end                            %
end                              %

%ARRANGING THE DATA ACCORDING TO CODEWORD

for i=1:n                         %
  for j=1:n                       %
    if (codeword_order(j)==(i-1)) % Arranging the data
      codeword(i)=x_tx(j);        %
```

```
    end                 %
  end                   %
end                     %

        %%%%%%%%%%%%%%%%%%%%%% LDPC DECODER %%%%%%%%%%%%%%%%%%%

%FUNCTION FOR CALLING LDPC DECODER

Function c_est=ldpc_decoder1(n,z,max_iterations,d,yi,codeword_order,
H)
% LDPC matrix
  H1= H';              % transpose of LDPC matrix

%VARIABLES

  qr=1;   %initial value for qij(1)needed to calculate rji(0)
  rji=1;   %initial value for rji(0)needed to calculate qij(0)
  excl=1;
  coun=0;  %initial value for a counter needed to verify that c*H'=0
  c1=[];
  k=[];
  L=[];
  qs=[];
  iteration=0;

% FIRST STEP: TO FIND INITIALIZED CODE qij(0)

  x=1+exp(2*yi./d);
  qij=x.^(-1);          %initialized code
  for i=1:z
    qs=[qs;qij];        %loop to put qij(1)in zxn matrix so that
rji(0)can be obtained
  end
  qu=qs;                %initial value for qo1(Ki)
  for i=1:n
    c1=[c1,1];          %initial value of c*H'
  end
while iteration~=max_iterations   %program's loop

% SECOND STEP: TO FIND rji(0)

  for i=1:n
    a=[];
    x=0;
     for j=1:z
        if H(i,j)==1
 m=[i,j];
 k=[k;m];%finding how many ones and their coordinates in H matrix
 a=[a,j];%columns number of each element equals to one in a certain row
         x=x+1;     %number of elements equal to one in a certain row
       end
     end
     excl=1;         %excluding operation
     for t=1:x                    %
       s=zeros(1,x);              %
       index=1;                   %
       for m=1:x                  %
         qr=1;
         if (m~=excl)             %
           s(index)=m;            %
```

```
          index=index+1;                  %
         end                              %
         if index==x
          for g=1:(x-1)
            qr=qr*(1-2*qu(i,a(s(g))));    %qij(1)used in an iteration
          end
                r(i,a(t))=0.5+0.5*qr;     %rji(0)
       wes=isinf (r(i,a(t)));
       mur=isnan (r(i,a(t)));
       if r(i,a(t))<1e-15
       r(i,a(t));
        %disp zero1
       else if (mur==1)
          r(i,a(t));
           %disp Not-a-Number1
       else if (wes==1)
          r(i,a(t))=0.9999;
           %disp infinty1
       else if r(i,a(t))==1
           r(i,a(t))=0.9999;
            %disp one1
       end
       end
       end
       end
          if r(i,a(t))<=(1e-15)
          r1(i,a(t))=0.9999;
          else
           r1(i,a(t))=1-r(i,a(t));   %rji(1)
          end
   end
       end

       s;
       excl=excl+1;
     end
  end
  k=[];                               %re initializing k

% THIRD STEP: TO FIND  qij(0) BY USING rji(0)

  for i=1:z
    a=[];
    x=0;
    for j=1:n
     if H1(i,j)==1
 m= [i,j];
 L=[L;m];%finding how many ones and their coordinates in H' matrix
 a=[a,j];%rows number of each element equals to one in a certain column
       x=x+1;          %number of elements equal to one in a certain row
     end
    end
    excl=1;                       %excluding operation
    for t=1:x                     %
      s=zeros(1,x);               %
      index=1;                    %
      for m=1:x                   %
        rji=1;
        rji1=1;
        if (m~=excl)              %
```

```
  s(index)=m;                              %
  index=index+1;                           %
end                                        %

      if index==x
      for g=1:(x-1)
       rji=rji*r(a(s(g)),i);            %rji(0)
       rji1=rji1*r1(a(s(g)),i);         %rji(1)
      end
        if rji==1
         q(i,a(t))=(1-qs(a(t),i));      %qij(0)
        else
        q(i,a(t))=(1-qs(a(t),i))*(rji);%qij(0)
        end
          wes=isinf (q(i,a(t)));
          mur=isnan (q(i,a(t)));
          if q(i,a(t))<1e-15
         q(i,a(t));
          %disp zero2
          %q(i,a(t))
          else if (mur==1)
         q(i,a(t));
          %disp Not-a-Number2
          else if (wes==1)
         q(i,a(t))=0.9999;
          %disp infinty2
          else if q(i,a(t))==1
         q(i,a(t))=0.9999;
          %disp one2
            end
            end
            end
            end
              if rji1==1
                q1(i,a(t))=qs(a(t),i);         %qij(1)
              else
                q1(i,a(t))=qs(a(t),i)*(rji1);  %qij(1)
              end
                wes=isinf (q1(i,a(t)));
                mur=isnan (q1(i,a(t)));
                if q1(i,a(t))<1e-15
               q1(i,a(t));
                %disp zero3
                %q1(i,a(t))
                else if (mur==1)
               q1(i,a(t));
                %disp Not-a-Number3
                else if (wes==1)
               q1(i,a(t))=0.9999;
                %disp infinty3
                else if q1(i,a(t))==1
               q1(i,a(t))=0.9999;
                %disp one3
                  end
                  end
                  end
                  end
qo0(i,a(t))=q(i,a(t))/(q(i,a(t))+q1(i,a(t)));   %Ki for qij(0)
qo1(i,a(t))=q1(i,a(t))/(q(i,a(t))+q1(i,a(t)));  %Ki for qij(1)
        end
```

```
      end
  s;
  excl=excl+1;
  end
end
L=[];                                %re initializing L

%FORTH STEP DETERMINING Qi(0)

  for i=1:z
    rji=1;
    rji1=1;
    for j=1:n
      if r(j,i)~=0
        rji=rji*r(j,i);              %rji(0)
        rji1=rji1*r1(j,i);           %rji(1)
      end
    end
    Q(i)=(1-qs(j,i))*rji;            %Qi(0)
    Q1(i)=qs(j,i)*rji1;              %Qi(1)
    Qo0(i)=Q(i)/(Q(i)+Q1(i));        %Ki of Qi(0)
    Qo1(i)=Q1(i)/(Q(i)+Q1(i));       %Ki of Qi(1)
  end

%FIFTH STEP THE ORIGINAL CODE

  for i=1:z
    if Qo1(i)>0.5
      c(i)=1;
    else
      c(i)=0;
    end
  end
   qu=qo1';
  iteration=iteration+1;
  c1=rem(c*H1,2);             %cxH'
  coun=[0];
  for j=1:n                   %
    if c1(j)==0               %check if c*H'=0
     coun=coun+1;             %
    end                       %
  end                         %
  if coun==n                  %
    iteration=max_iterations; %
  end
end
c_est=c;
```

Appendix II Matlab Source Code of Image Filtering Simulation

```
  %%%%%% PROGARM FOR PRESENTING THE IMAGE DATA TO LDPC ENCODER %%%%%%

function TOTAL_BLOCKS=PreEncodingProcess(A)
TOTAL_BLOCKS=[];     %INITIAL VALUE OF BLOCKS MATRIX
x=0;                 %
y=0;                 %INITIAL VALUES
z=32;                %
Data =de2bi(double(A),8);%CONVERSION IMAGE DATA FROM DECIMAL TO BINARY
for i=1:2048       %NUMBER OF BLOCKS
  Block=[];           %
  x=y+1;              %
  y=y+z;              %
  for j=x:y           %
    for n=1:8         %
    B=Data(j,(9-n));  %
    Block=[Block B];  %CONSTRUCTED BLOCK
    end               %
  end                 %
  TOTAL_BLOCKS=[TOTAL_BLOCKS; Block];%ALL BLOCKS MATRIX
end                   %ENDING PROCESS

  %%%%%% % PROGRAM TO APPLAYING THE WGN OVER PIXELS OF IMAGES %%%%%%%

%VARIABLES

clear all,clc;
starttime = cputime;
m=256;                  %number of rows
n=512;                  %number of columns
max_iterations=100;    %maximum number of iterations
rate=(n-m)/n;
variance=1;

%LOADING THE MATRICES

load ldpc256_512_gen.txt;
load ldpc256_512_pchk.txt;
AinvB_256_256=ldpc256_512_gen;
H_256_512=ldpc256_512_pchk;
%SORTING OF CODE WORD
load codeword_order.txt;
cnt=1;
for i=1:26
  for j=1:20;
    new_codeword_order(cnt)=codeword_order(i,j);
    cnt=cnt+1;
  end
end

codeword_order_1_512=new_codeword_order(1:512);

%CALCULATION VALUE OF SNR

SNR_db=5;
  disp('SNR');
  SNR=10^(SNR_db/10);
  variance=1/(2*SNR*rate)
```

```
load ldpc256_512_gen.txt;
load ldpc256_512_pchk.txt;

AinvB_256_256=ldpc256_512_gen;
H_256_512=ldpc256_512_pchk;

%INSERTING THE IMAGE

A = imread('Lena_','tiff'); %IMAGE AQUISITION
%CALLING DATA PREPERING PROGRAM
Blocks=PreEncodingProcess(A);
for i=1:2048
  original_data = Blocks(i,:);

%CALLING THE LDPC ENCODER

yi=ldpc_encoder(m,n,AinvB_256_256,codeword_order_1_512,original_data);
%APPLAYING WGN NOISE
    for j=1:n
      yi(j)=yi(j) + (sqrt(variance)*randn);
    end

%CALLING THE LDPC DECODER

est_data=ldpc_decoder1(m,n,max_iterations,variance,yi,
codeword_order_1_512,H_256_512);
final_est=est_data(codeword_order_1_512(m+(1:(n-m)))+1);
TOTAL_BLOCKS(i,:)=final_est;
end

%CALLING THE PROGRAM FOR CONVERTING THE FILTERED DATA TO PIXEL VALUES

Reconstructed_Image=AfterDecodingProcess(TOTAL_BLOCKS)
%SHOWING THE RECONSTRUCTED IMAGE BY LDPC SYSTEM
figure,imshow(Reconstructed_Image),title('Reconstructed Lena Image')%
Test_time =cputime - starttime;
disp(sprintf('CPU PROCESSING TIME = %5.4f Second',Test_time))

     %%%%%%%%%%%% PROGRAM TO RECONSTRUCT DECODED IMAGE %%%%%%%%%%%

function Reconstructed_Image= AfterDecodingProcess(TOTAL_BLOCKS)
%VARIABLES
decoded_matrix=[];         %
for i=1:2048               %
x=1;                       %
y=8;                       %
z=8;                       %
  while x<257              %
    decoded_row=[];        %
    for j=x:y              %
    decoded_element=TOTAL_BLOCKS(i,j);           %
    decoded_row=[decoded_row  decoded_element];   %Converting  the  data
from binary to decimal
    end                   %
   decoded_matrix=[decoded_matrix; decoded_row]; %
    x=y+1;                %
    y=y+z;                %
  end                     %
end                       %
```

```
Pixels=bi2de(decoded_matrix,'left-msb');     %
Primary_Result=[];                           %
x=0;                                         %
y=0;                                         %
z=256;                                       %
for i=1:256                          %Sorting the data in image size
  Image_row=[];                      %
  x=y+1;                             %
  y=y+z;                             %
  for j=x:y                          %
    pixel=Pixels(j);                 %
    Image_row=[Image_row pixel];     %
  end                                %
 Primary_Result=[Primary_Result; Image_row];  %
end                                          %
Img=Primary_Result';                         %
Reconstructed_Image = uint8(Img);            %

  %%%%%% PROGRAM FOR IMPLIMENTING CHOSEN OTHER THREE FILTERS %%%%%%%%
clear all,clc;
  starttime = cputime;
   I = imread('Moon Image','tiff');
   J = imnoise(I,'gaussian',0,0.09);

   figure, IMshow(I)
   image([],[],I),title('Original Moon Image')
   figure, IMshow(J)
   image([],[],J),title('Noisily Image with Gaussian Variance=0.09')

   Test_time =cputime - starttime;
   disp(sprintf('CPU PROCESSING TIME = %5.4f Second',Test_time))
        %%%%%%%%%%%%%%%%%%% HARMONIC MEAN FILTER %%%%%%%%%%%%%%%%
   starttime = cputime;
   A = SPFILT(J,'hmean',3,3);
   figure, IMshow(A)
   image([],[],A),title('Filtered Image by Harmonic Mean Filter')
   Test_time =cputime - starttime;
   disp(sprintf('CPU PROCESSING TIME = %5.4f Second',Test_time))
        %%%%%%%%%%%%%%%%%%% ALPHA-TRIMMED FILTER %%%%%%%%%%%%%%%%
   starttime = cputime;
   B = SPFILT(J,'atrimmed',3,3,2);
   figure, IMshow(B)
   image([],[],B),title('Filtered Image by Alpha-trimmed Filter')
   Test_time =cputime - starttime;
   disp(sprintf('CPU PROCESSING TIME = %5.4f Second',Test_time))
        %%%%%%%%%%%%%%%%%%% ADAPTIVE MEDIAN FILTER %%%%%%%%%%%%%%%%
   starttime = cputime;
   C = adpmedian(J,7);
   figure, IMshow(C)
   image([],[],C),title('Filtered Image by Adaptive Median Filter')
   Test_time =cputime - starttime;
   disp(sprintf('CPU PROCESSING TIME = %5.4f Second',Test_time))
  %%%%%%%%% CALCULATING PSNR FOR OUTPUT IMAGE FOR EACH FILTER %%%%%%%
  %error = I - A;
  error = double(I) - double(A);
  decibels = 20*(log10(255./(sqrt((1/256^2)*(sum(sum(error.^2)))))));
  disp(sprintf('PSNR = +%5.4f dB',decibels))

  %error = I - B;
  error = double(I) - double(B);
```

```
decibels = 20*(log10(255./(sqrt((1/256^2)*(sum(sum(error.^2)))))));
disp(sprintf('PSNR = +%5.4f dB',decibels))

%error = I - C;
error = double(I) - double(C);
decibels = 20*(log10(255./(sqrt((1/256^2)*(sum(sum(error.^2)))))));
disp(sprintf('PSNR = +%5.4f dB',decibels))

    %%%%%%% PROGRAM FOR CALCULATING BRIGHTNESS AND CONTRAST %%%%%%

FIL = A;
FILB= double(FIL);
k=0;
N=0;
b=0;
M=0;
for i=1:256
  for j=1:256
    if FILB(i,j)>128
      FILB(i,j);
     k=k + FILB(i,j);
      N=N+1;
    else
      b=b+FILB(i,j);
      M=M+1;
    end
  end
end
whitest=k/N;
Darkest=b/M;
Contrast_A=whitest-Darkest
Brightness_A=mean(mean(FIL))
```