IMAGE CORRECTION USING REED MULLER CODE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Siva Krishna Ginjupalli

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2013

Fargo, North Dakota

North Dakota State University

Graduate School

Title

Image Correction using Reed Muller Code

By

Siva Krishna Ginjupalli

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard

Chair

Dr. Simone Ludwig

Dr. Luis Del Rio Mendoza

Approved:

7/1/2013	Dr. Brian Slator
Date	Department Chair

ABSTRACT

Image signal processing is one of the important aspects while communicating from long distances. Background noise of images is one of the primary concerns in obtaining clear accurate images. This problem is further amplified if the spacecraft transmitting images are farther away from the earth's orbit. The larger the distance of the transmitter from the earth the greater is the problem of background noise.

In order to overcome this problem, the image information obtained is reconstructed using error-correcting codes. In my paper I have used the Hadamard matrices to generate a 32x32 matrix consisting of binary digits with each row representing a number between 0-31, the first row represents the number 0 and the 32nd row represents the number 31. The results proved that the error correction approach employed in this paper is very accurate when the number of errors in each row is less than 8.

ACKNOWLEDGEMENTS

I would take this opportunity to thank my advisor, Dr. Kendall Nygard, who has given me valuable support, encouragement and advice without which this work would not have been completed. I am thankful to the members of the committee, Dr. Simone Ludwig and Dr. Luis del Rio for their support. I would also like to thank my parents for their valuable support and constant encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Wireless data transmission has become an essential part of many applications. It can be of any type for how we transmit using the devices available. Some of the communicating devices are a wireless internet connection, cell phone conversation, radio broadcast, or some military application. The need for data to be decoded error-free across a wireless medium is vital. The maximum capacity of any given channel is determined by Shannon's limit. This limit expresses the error free transmission rate for any channel given a specific signal-to-noise ratio (SNR) and channel bandwidth developed by Claude Shannon [1]. A communication between two channels is to connect the geographical locations to transmit the data among these channels and to know the locations of each state. The transmission of a signal can be varies based on each state of the signal received and external factors can also affect the transmission of the signal based on the noise that has been induced.

When a message is transmitted, it has the potential to get scrambled by noise. This is certainly true in case of digital messages where the transmission is done through the sequence of binary bits 0s and 1s. During transmission the signals get distorted resulting in a change in the order the bits are transmitted or the bits could be replaced with newer ones at the receiving station. The process of encoding the transmitted bits to get the original set of transmitted bits based on the accuracy is called the error correction [10].

In this paper we discuss how Reed Muller error correcting codes can be used in constructing an image when an image is transferred from one station to another. NASA has done a lot of research to get information about the environment in surrounding planets and the existence of livable conditions for the humans. In this paper the implementation of an error correcting mechanism for data transmitted from the outer space is evaluated. In this paper I worked on building a software application that will accept an image, I will add noise to a signal by introducing some errors in that image’s digital representation and then recover the original image from the bad data., So the main objective is to be able to recover that image from the bad data and also make some observations like how much noise is acceptable. I also conducted some experiments in which the correcting software was used to retrieve the same image after it was altered by inserting varying number of errors.

When some data is transmitted to the Earth it is possible that the signal can get garbled. When the data reaches its destination there is a good chance it could be confused with background noise, resulting in an erroneous recording of the data observed in the space. The Reed-Muller Error-Correcting Code is a technique used to remove errors that were induced in the digital imaged while they were transmitted from the outer space. The experiments in this paper consider that the data to be transmitted from the outer space is digital images.

In this paper we explain the process of how the images are transmitted from the space station to the earth receiving station. Different results were conducted for how the errors were induced and for each case the program randomly induces the error in to the transmitting bits which can be imagined as a scrambled image.

2. LITERATURE OVERVIEW

2.1. Error-correcting codes

In this paper [5] the author describes the use of different error correcting codes. Error correcting codes are used to protect the digital data against the transmission of errors. Linear codes, where a message can be transferred in the string format of bits 0s and 1s and due to noise the probability(p) of receiving the error bit would be p=1/100 of that channel as explained by the author. This is called a binary symmetric channel. Golay code can be used to construct non-linear error correcting codes in the vector format. Some interesting properties of Golay Code are Cyclic Invariance, Inversion, Minimum Hamming distance and Error Correction. The Golay code [4] is obviously not able to encode a large amount of data in one code word as it needs to transmit a large amount of check bits as data bits. It could be used in even the smallest microcontrollers, such as the PIC series and 68HC05.

2.2. Reed Muller error-correcting codes

The author Ben Cooke [3] clearly describes the use of the Reed Muller codes. These are some of the oldest error correcting codes. Error correcting codes are very useful in sending information over long distances or through channels where errors might occur in the message. This came in to popular by self- correcting the codes when a message is transmitted. Reed Muller codes were invented in 1954 by D. E. Muller and I. S. Reed. In 1972, a Reed Muller code was used by Mariner 9 to transmit black and white photographs of Mars. Reed Muller codes are relatively easy to decode, and first-order codes are especially efficient. Encoding and Decoding can be done for better transmission of the signal without the noise. Decoding Reed Muller encoded messages is more complex than encoding them. Encoding and decoding of the messages is based on the distance between vectors. The distance between any two vectors is the number of places in the two vectors that have different values. The distance between any two codewords in $R(r, m)$ code is 2^{m-r} where m is any positive integer between {0, 1} and r is the r^{th} order of Reed Muller Code.

In this paper [2] the author describes about some interesting properties and some among them are; these codes form an infinite family of codes, and they can be constructed as larger Reed-Muller codes from the smaller ones. This particular observation leads us to show that Reed-Muller codes can be defined recursively. The drawback while constructing the larger ones are they become weaker as their length increases. However, they are often used as building blocks in other codes. One of the major advantages of Reed-Muller codes is their relative simplicity to encode messages and decode received transmissions. We examine encoding using generator matrices and decoding using one form of a process known as majority logic.

The r^{th} order Reed Muller code, denoted *R(r, m),* is the set of all polynomials of degree at most r in the ring R_m. Different set of vectors can be used for the representation of the matrices. Some of them can be constructed by the addition, complement or by multiplication of them. Vectors can be associated with Boolean polynomials. A Boolean polynomial is a linear combination of Boolean monomials with coefficients {0, 1}.

2.3. Hadamard matrices

A *Hadamard matrix H* of order n is an $n \times n$ matrix of 1s and -1s in which $HH^T = nI_n$. (I_n is the $n \times n$ identify matrix). In this paper [6] the author describes the concept of a Hadamard matrix as a binary orthogonal matrix is extended to higher dimensions. An n-dimensional Hadamard matrix [h_{ijk cdots n}] is defined as one in which all parallel (n - 1) dimensional layers, in any axis-normal orientation, are uncorrelated. The n-dimensional Hadamard matrices can be defined in a special way where above all two-dimensional layers, in all axis-normal orientations, are Hadamard matrices, and as a consequence that all the intermediate-dimensional layers can be the same Hadamard matrices. Different procedures are described for deriving three- and four-dimensional Hadamard matrices of varying propriety from two-dimensional Hadamard matrices. A formula is given for a fully proper n-dimensional matrix of order two, which can be expanded by direct multiplication to yield proper(2^{t})^{n}Hadamard matrices. It is suggested that proper higher dimensional Hadamard matrices may find application in error-correcting cedes, where their hierarchy of orthogonalites permit a variety of checking procedures. Other types of Hadamard matrices may be of use in security codes on the basis of their resemblance to random binary matrices.

2.4. Optical communication over RM codes

The author [7] considers the GLDPC (Generalized low-density parity check) codes with RM (Reed-Muller) and BCH (Bose-Chaudhuri-Hocquenghem) codes as the component codes and he also said the combination of GLDPC codes with RM codes as component codes is the best option for high speed optical transmission. The GLDPC codes gained high importance for optical communication by improving the characteristics of LDPC codes by decreasing the complexity of the decoder. Replacing the parity check equations in a parity-check matrix of GLDPC code by a linear block code is achieved. Replacing this parity check is known as the constituent code and this construction is proposed by Lentmaier and Zigangirov. An interesting property of RM codes is that they can be defined recursively and if that can be applied successively several times, then it can be decomposed in to several parity check codes. With all these the complexity of GLDPC codes with RM component codes is of order $N \log_2 n$. The author also provides the simulation results which prove the GLDPC codes along BCH and RM component codes are possible options for high-speed optical transmission.

2.5. Image construction using Reed Muller

In this paper [8] the author explains how he utilized the use of Reed Muller Sequences for reconstruction of the 1D image by compressing sensing and this way has the loss in speed and accuracy when the degree of sparsity is not high. The solution that the author has proposed is based on the knowledge of the Fourier analysis that the energy of the wavelet coefficients is concentrated in the upper-left region to detect the large portions of locations in one step. The results that were produces by doing this experiment were each image was sparsified by computing to get the original image by taking 25% of the noiselet measurements of RM measurements. A image reconstruction algorithm has been proposed for the compressing sensing of the images where this algorithm provides improved construction in terms of error and computational efficiency. An updated least squares method has been used to get the increase in computational efficiency and stability.

2.6. Multiple bit error correction

In this paper [9], the author explores the use of Reed Muller codes in memory interface applications to address the multiple-bit soft errors. The author also explains the construct and decoding a simple RM code and these codes has multiple bit error correction capability with relatively low latency and high performance. The paper also explains the design of the encoder and decoder where the encoder takes a 16-bit message and encodes in to 32-bit code word based on the RM code input matrix multiplication. In the error mode a bit was introduced for the functionality and during this phase the bits were reversed where the 1 becomes 0 or vice versa. The decoder has three stages where each stage can increase the performance. The major components were orthogonal checksum Generator (OCG) and Majority Logic Decoder (MLD).

The codes can be concatenated to get the message width. For single-data rate or double-data rate applications the external memory interface can be adjusted with the width of the message.

3. METHODOLOGY

In this chapter the approach employed to convert the image into an array of numbers, inducing errors and then recovering the image from the bad data is explained. I included screenshots for the user interface of the software application I built. I also explained about the coding platforms and techniques I used to build this software application.

3.1. Solution proposed

The standard practice in computer graphics is to assume that each picture is composed of pixels. In The Reed-Muller Error-Correcting technique I used the fact that each pixel can be represented by 3 numbers, they are its R quotient, G quotient and B quotient. I used Bitmap class from .net libraries to extract these numbers, the range for these RGB numbers is 0-255.

So when an individual pixel, encoded in a bit pattern of 0s and 1s reaches the receiving station, its original form, say

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

It might have changed to

0 1 0 **0** 1 0 1 0 0 **0** 0 1 **0** **1** 1 0 0 1 **1** 1 1 0 1 **1** 0 1 **1** 1 1 0 1 0

where in this illustration seven bit-change errors have occurred in the message. Since, 2**5 = 32, one might think that 5 bits of information would suffice to represent 32 numbers. Yet the string displayed above has 32 bits, not 5. It is part of a code designed to make errors correctable.

The code used by Mariner spacecraft was essentially the (32, 5) Reed-Muller code, defined in the following discussion.

Let the matrix H1 be given by

1	1
1	-1

And define $H_{n+1} = H_n \otimes H_1$, where $\otimes$ denotes the Cartesian product of matrices in which we replace each + 1 entry of H_n by H_1 and each -1 entry by - H_1. These equations define what are called Hadamard matrices. The Mariner telemetry code consists of the rows of a matrix M_5 obtained by replacing 1 by 0 and -1 by 1 in H_n and turning the matrix on its side. The matrix can then be displayed as a 32 X 32 square grid. The 32 X 32 matrix generated with 0s and1s is shown below.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1

0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1

0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0

0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1

0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0

0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0

0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1

0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1

0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0

0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0

0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1

0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0

0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1
```

0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1

0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0

0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0

0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1

0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0

0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1

0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1

0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0

0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0

0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1

0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1

0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0

0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1

0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0

0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0

0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1]

The rows represent the 32 possible code words to be transmitted by the Mariner spacecraft, and they have a very interesting property: any two of them differ in exactly 16 places. For example, if we compare the second and third rows, we find that in the following positions one row has a 0 while the other has a 1, or vice versa.

2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31

When a new 32-bit code word is received, it is compared to the rows of the matrix and the row it most closely resembles is selected as the word transmitted. For example, if there has been only one error in a word, it will differ from one of the rows of M in only one digit and will not be mistaken for any other row. If two errors have occurred, the same thing is still true. Up to seven errors may occur with no danger of confusion about which word was transmitted by the spacecraft.

But if eight errors have occurred, then the received word may differ as much from some other row of the matrix as from the intended row. This is due to the fact that those two rows differ in only 16 places, and the eight errors might have resulted in a word which is just as "close" to an incorrect row as to the correct one.

The number of places in which two binary words or vectors differ is called the Hamming distance between them (named for the famous scientist Richard Hamming, who was a founder of the Association for Computing Machinery). In general, a set of code words which are all mutually at Hamming distance d or more enables users to detect and correct up to (d - 1)/2 errors.

When the transmission of a pixel is done, the pattern for an entire row is transmitted to represent one of the 32 possible levels of grayscale. At the ground station, each received word is easily matched against the rows of the matrix and the row with the smallest Hamming distance best match is selected.

3.2. Implementation

I created a WPF application and when we run this application it will open a window which will let us choose an image, and then we can click on the "Generate Output Image" button and then my application will go through each pixel and gets it RGB (RED, GREEN and BLUE) values.

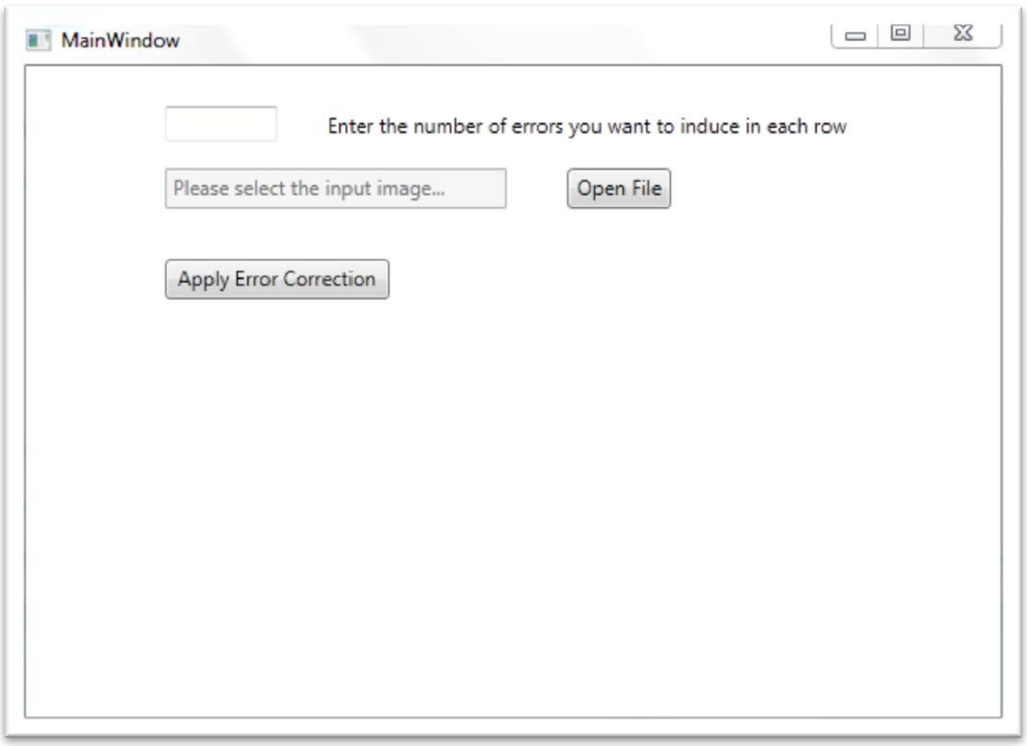
MainWindow
Enter the number of errors you want to induce in each row
Please select the input image...
Open File
Apply Error Correction

Figure 1. Window for selecting the image

An image is just a collection of pixels grouped together, each pixel has a color and every color can be represented using three numbers they are its RED quotient, GREEN quotient and BLUE quotients. These RGB numbers vary from 0 - 255 so I extract these RGB values for each pixel and put them in an array. Now since the 32 X 32 Hadamard matrix only has 32 rows we can only transmit numbers from 0-31 whereas the numbers we have are in the range 0-255. So for my experiments sake I took an image whose colors are limited so that its RGB numbers are between 0-31. And then I extract all these RGB numbers from the image and load them into an array. Let’s call this array as X; in real scenario (The image transmission from outer space) the requirement is to transmit this array to the earth. I am then creating a new array Y and in this array I am loading the rows of the Hadamard Matrix the numbers in array X represent.

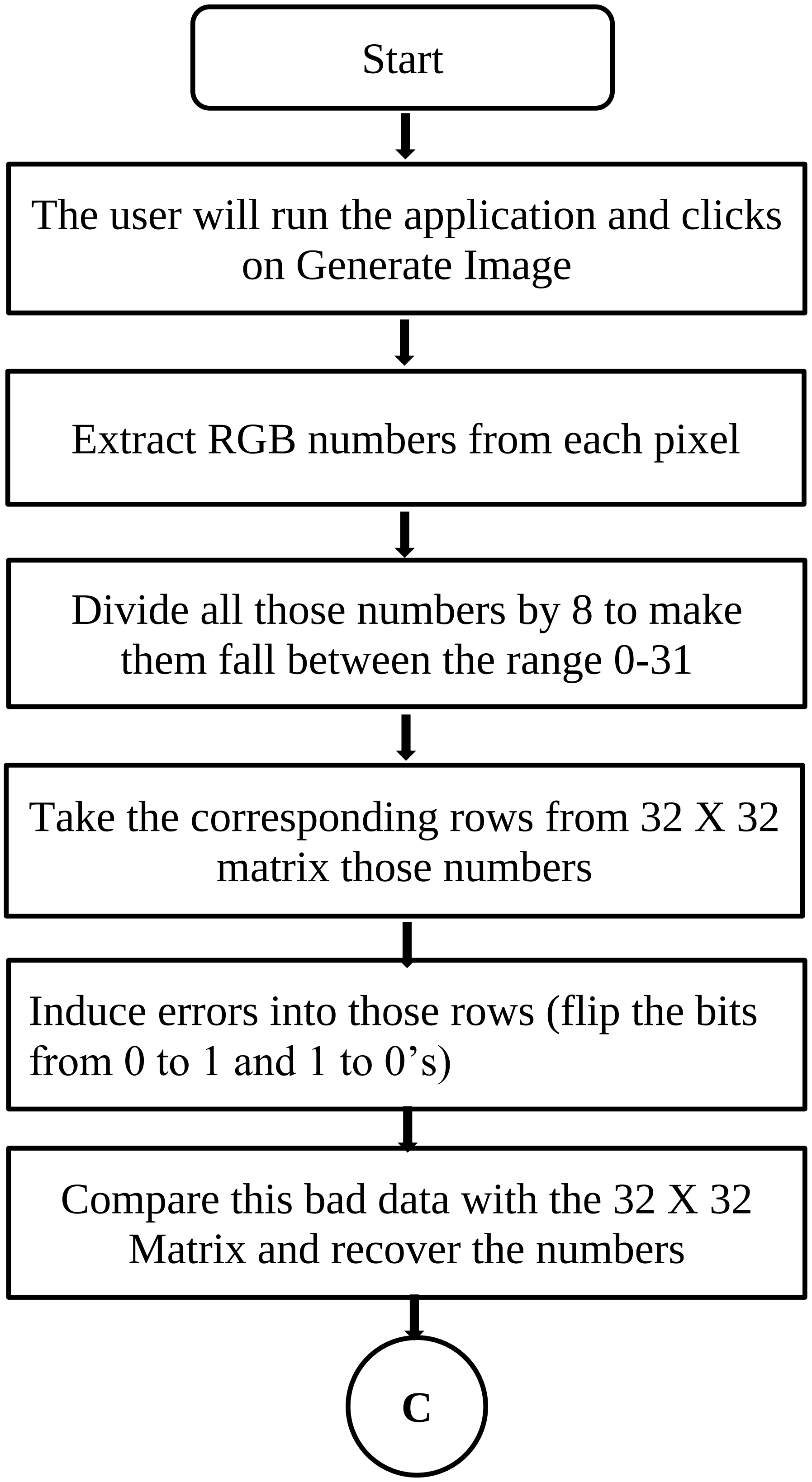
Start
The user will run the application and clicks on Generate Image
Extract RGB numbers from each pixel
Divide all those numbers by 8 to make them fall between the range 0-31
Take the corresponding rows from 32 X 32 matrix those numbers
Induce errors into those rows (flip the bits from 0 to 1 and 1 to 0's)
Compare this bad data with the 32 X 32 Matrix and recover the numbers
C

Figure 2. Flowchart explaining the application process

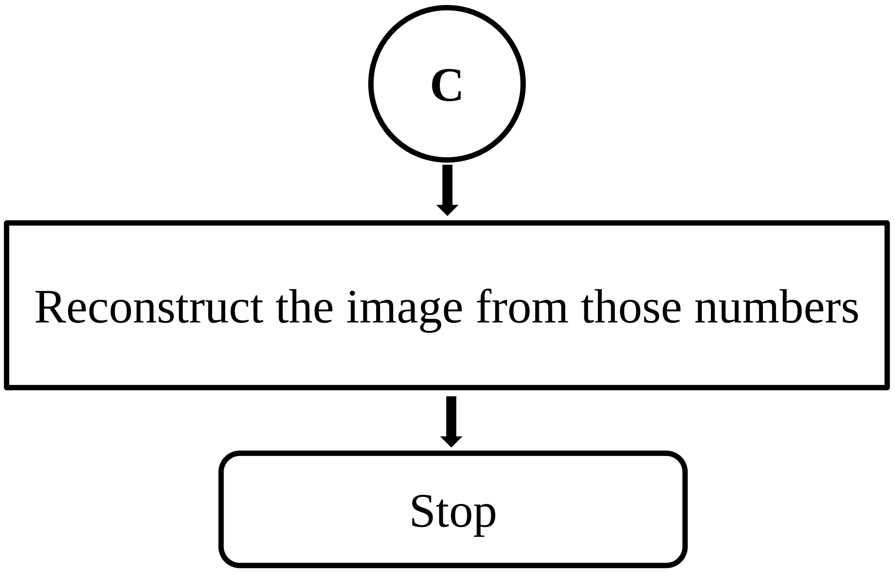

Figure 2. Flowchart explaining the application process (continued)

So now I have array Y and each element in it contains 32 bits of 0s and 1s but at this point the data is perfect and during its journey from outer space to earth some bits will be corrupted. So in order to replicate this I am looping through each element of array Y and inducing up to 7 errors to it and inserting the new bad data into a new array Z. Now in real scenario the data in array Z is what the earth receives and we have to construct an image using it. Now I am looping through each element of array Z (each element of array Z in turn contains 32 bits of 0s and 1s) and comparing it with each row in the 32 X 32 Hadamard matrix and determining which row this element is closest to and I am inserting its row number into another array P and from the resultant RGB values I am reconstructing the image. I also did different experiments like inducing more than 8 errors and others which will be discussed in the results section.

3.3. Coding platform and techniques used

I used WPF windows forms to build the front end user interface which lets the user pick the image he wants to test this Reed Mueller Error correction code technique, also allows the user to select the number of errors. The back end code is done using the programming language c#. So when the user selects and image and clicks on the button “Apply error correction” I am creating an object for the bitmap class and then using that object to find out the RGB values at each pixel and I am storing them in an array. I then load the rows from the 32 X 32 matrix these numbers represent into a new array. At this point a function is called which will induce errors into the data we have, this function will accept the array holding the 32 X 32 matrix rows and number of errors we want to insert in each row, this function is also capable of choosing the number of errors randomly. I then compare this bad data with the 32 X 32 matrix to determine which row is closest to the bad data and load the row numbers into a new array and use the same bitmap object to reconstruct the image from these numbers.

3.4. Pseudo code for the function which is inducing the errors

```
        Call the function which induces the errors and pass it the array (let’s say A) holding the
32 bit elements and also feed it the number of errors X we want to introduce
Create a new array B and insert numbers 0 - 31 into it
For each row A
    Shuffle array B randomly
    Take first X elements from the array
    For i = 0 to X
        If A [B[X]] = 0 then A [B[X]] = 1 else A [B[X]] = 0
    End For
End For
```

3.5. Pseudo code for the entire application

```
Get the image from the user
Extract RGB values for each pixel and put them in an array A
Insert their corresponding 32 X 32 matrix rows into array B
Induce errors in each row
Go through each row, compare it with the 32 X 32 matrix calculate the hamming distances
Identify input data and insert it into array C
Reconstruct the image from the numbers in array C
```

4. RESULTS

I conducted different experiments by varying the number of errors induced into the original data. Once the error bits are induced the application will compare the erroneous data against the 32x32 Hadamard matrix to fix the bad data. The time taken to generate the output image will vary on the image size and the pixel density, the more the number of pixels the more time it requires to fix data. In all these experiments I included a table in which we can see the good data and the data after errors are induced. All the experiments were conducted on a single image and we recorded the bits that were changed during this process. The process works well when the number of errors in each row is less than or equal to 7. We also ran some tests to see what happens if the number of errors in each row is more than 7 and the output image that it produced is blurry and the numbers recovered after the processing is done is not same as the original and that is because this approach works well only when the number of errors in each row is less than equal to 7. I'm using the Figure 3 for the experiments 1 to7.

Figure 3. Image used for the experiments

4.1. Experiment one

In this experiment, we induced 3 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 4 and the table showing the first few rows of good and bad data is shown in Table 1 and Table 2. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8. The output image and the values in the table show that the image is accurate.

Table 1. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
00**1**0111110000**0**111000011110000111**0**	-
01**01**01100110011001100110011001**0**0	-
00**1**0**1**000000**1**0000000000000000000000	-
Corrected Data Displayed	-
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0

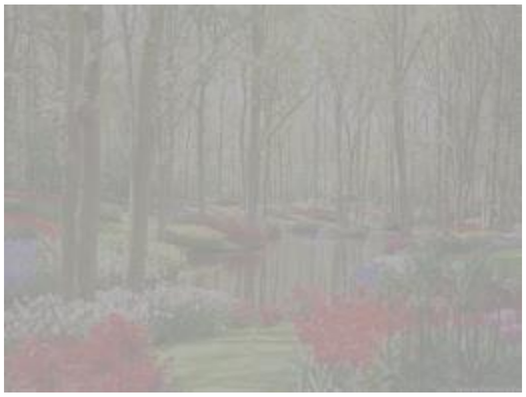

Figure 4. Generated output image with 3 error bits

Table 2. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111110000111100110000111100001 1	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
00**0**111000**1**1111001100001**0**11000011	-
01**1**110100101101010100101101**11**101	-
01**1**101010101010**0**1010**0**01010101010	-
Corrected Data Displayed	-
00111110000111100110000111100001 1	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17

4.2. Experiment two

In this experiment, we induced 5 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 5 and the table showing the first few rows of good and bad data is shown in Table 3 and Table 4. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8. The output image and the values in the table show that the image is accurate.

Table 3. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
0000111**0**0000111**0**000001**0**11000**1**1**0**11	-
01**0**001100110**10**10011001**01**01100110	-
000000**1**0000000**1**000**1**00**1**0000**1**00000	-
Corrected Data Displayed	-
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0

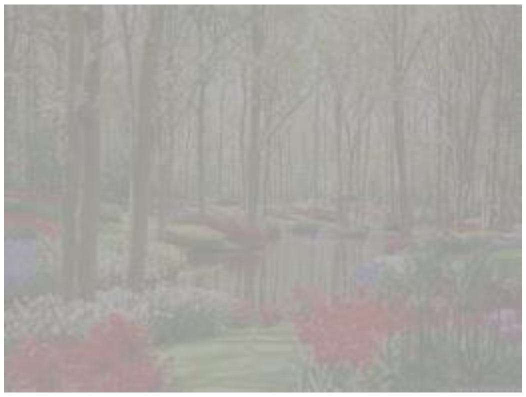

Figure 5. Generated output image with 5 error bits

Table 4. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
0011110000111**0**001100**1**0111**0**00**1**0**0**1	-
0**0**011010010**0**10101**1**10010110**0**0010**0**	-
0101010101**1**1010**00**0101010**01**101010	-
Corrected Data Displayed	-
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17

4.3. Experiment three

In this experiment, we induced 7 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 6 and the table showing the first few rows of good and bad data is shown in Table 5 and Table 6. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8. The output image and the values in the table show that the image is accurate

Table 5. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
00**1**0111**0**000**1**1**0**11**1**000**0**111**1**0001111	-
0110**1**11**1**0**0**1001100110011**1**01**01**01**0**0	-
00**1**000000000**1**00**1**000**1**0**1**00000**1**0**1**0000	-
Corrected Data Displayed	-
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0

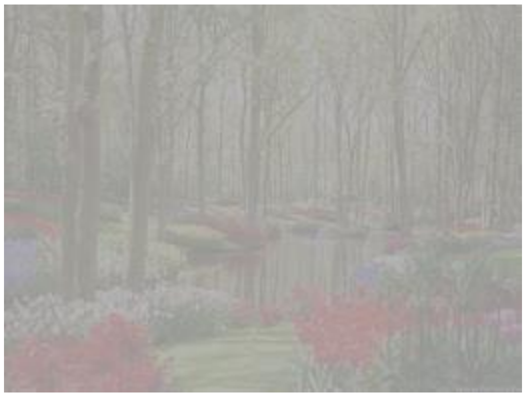

Figure 6. Generated Output image with 7 error bits

Table 6. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111110000111100110000111100011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
00**0**1**00**0**1**0**1**11110011000**10**111000011	-
010110100**0**0**0**101**1**1010**1**101**0**01**1**0**0**01	-
0**01**1**1**10**0**01**10**01011010101010**0**01010	-
Corrected Data Displayed	-
00111110000111100110000111100011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17

4.4. Experiment four

In this experiment, we induced 8 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 7 and the table showing the first few rows of good and bad data is shown in Table 7 and Table 8. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8. The output image and the values in the table show that the image is accurate.

Table 7. Data transmitted and recovered for PIXEL 11

Data Transmitted	Row #
01011010101001010101101010100101	13
01100110011001101001100110011001	19
00110011001100111100110011001100	18
Data Received	-
01011010**01**10010101**1**1**0**0**0**01**10**0**1**101	-
01100110**10011**110**0**001**01**0110011001	-
0011**1**0**0**1001**01**011110**1**1**0**0**1**11001**0**00	-
Corrected Data Displayed	-
01011010101001010101101010100101	13
01100110100110010110011010011001	11
00110011001100111100110011001100	18

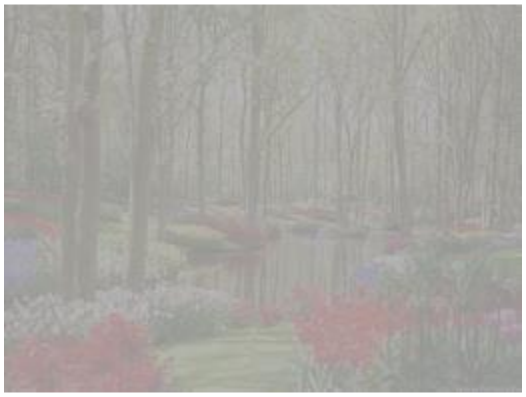

Figure 7. Generated output image with 8 error bits

Table 8. Data transmitted and recovered for PIXEL 18

Data Transmitted	Row #
01010101101010101010101001010101	25
00111100001111001100001111000011	22
01011010010110101010010110100101	21
Data Received	-
01010101**0**010101**1**1**1**1**1**10**0**001**1**10**0**0**0**	-
00111100**1**011**0**1001**01**0**1**0**0**1**0**10000**0**1	-
0**0**0**00**01001**1**1**0**01010**0**0**10**0110100101	-
Corrected Data Displayed	-
00001111000011111111000011110000	20
00111100001111001100001111000011	22
01011010010110101010010110100101	21

4.5. Experiment five

In this experiment, we induced 15 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the table showing the first few rows of good and bad data is shown in Table 9 and Table 10. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8 and when the number of errors in each row is more than 8 we are not able to recover every number perfectly and hence the image is getting blurred. The output image and the values in the table show that the image is not accurate. The number that recovered is not the actual number that needs to be recovered. The original image and the recovered image are shown below and the differences are highlighted in recovered image and compared in Figures 8 and Figure 9.

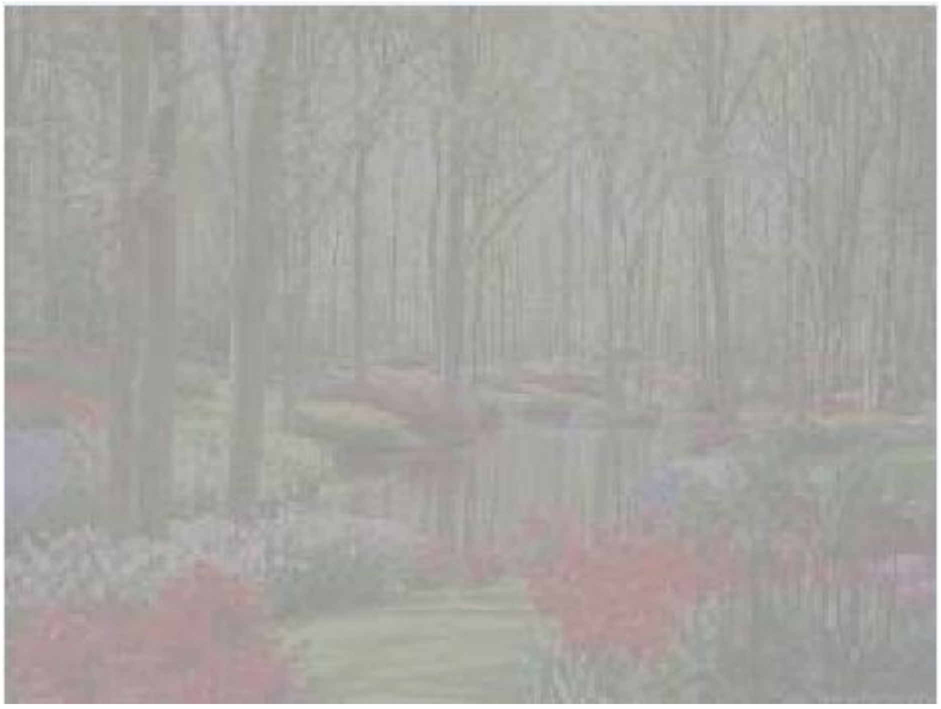

Figure 8. Original image used

Table 9. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
10**1**01111000**10**11**0**0**111**11**00**00**1100**1**0**	-
1000**1001**0**0**100**00**0**1**11**1**0**00**0011001**0**0	-
0**11**0000**11**00**11**0000**1111**000**1**0**11**0**1**0**1**	-
Corrected Data Displayed	-
00111100001111000011110000111100	6
00000000000000000000000000000000	0
01101001100101100110100110010110	15

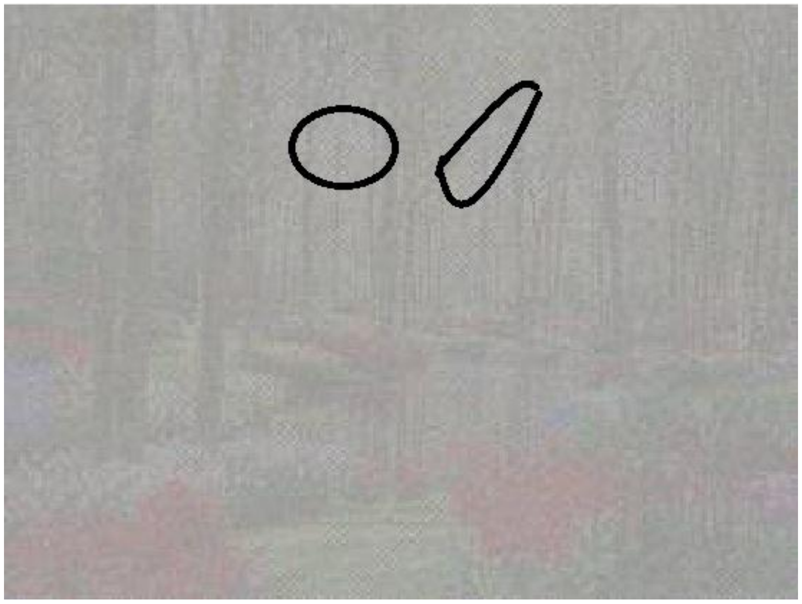

Figure 9. Generated output image with 15 error bits

Table 10. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
101**0**1**0**0**1**001**0**1**01**0**00**0**1**0**10**11**0**00**1**0**0**1	-
1011**0**01**11**1**10**101**1**1010**10**01**0**0**01**0**0**01	-
11**10**0**01**1010**01**1**1**1**01**1**1**101**101**10**0**010	-
Corrected Data Displayed	-
01010101101010100101010110101010	9
00110011001100110011001100110011	2
01100110011001100110011001100110	3

4.6. Experiment six

In this experiment, we induced 20 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 11 and the table showing the first few rows of good and bad data is shown in Table 11 and Table 12. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8 and when the number of errors in each row is more than 8 we are not able to recover every number perfectly and hence the image is getting blurred. The output image and the values in the table show that the image is not accurate. The number that recovered is not the actual number that needs to be recovered. The original image and the recovered image are shown below and the differences are highlighted in recovered image and compared in Figures 10 and Figure 11.

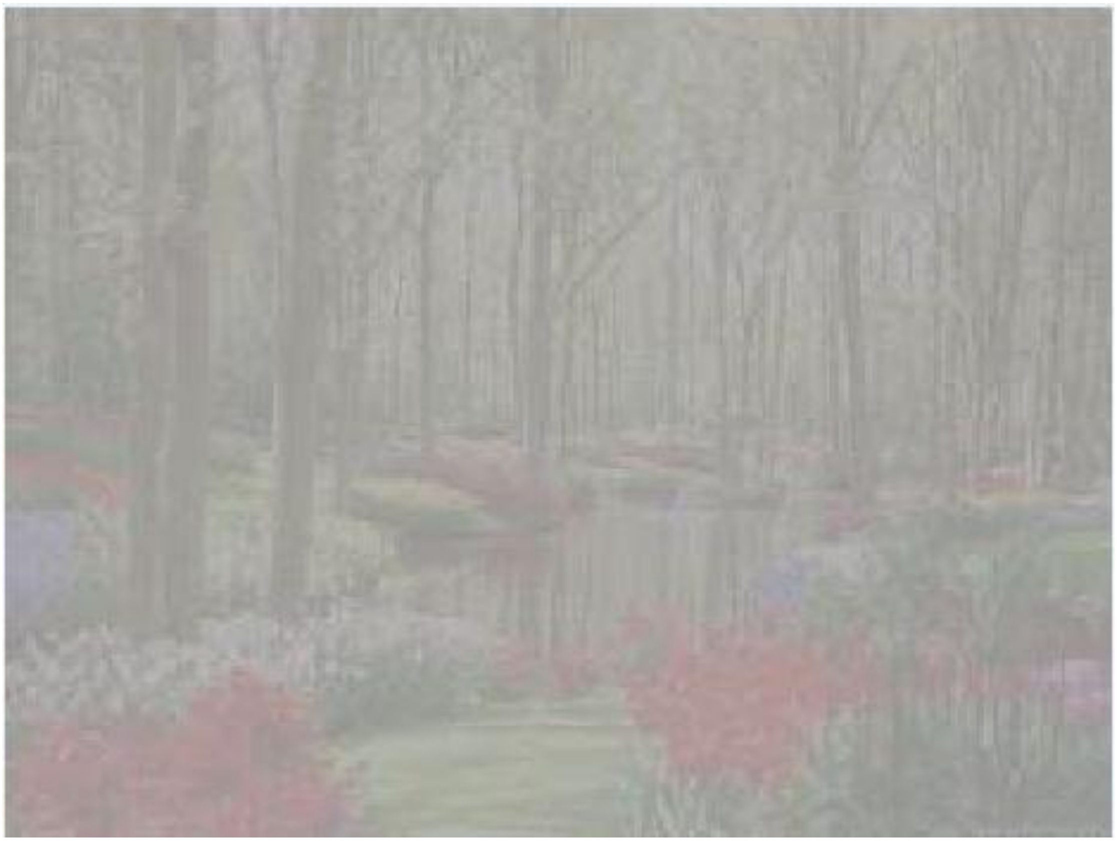

Figure 10. Original image used

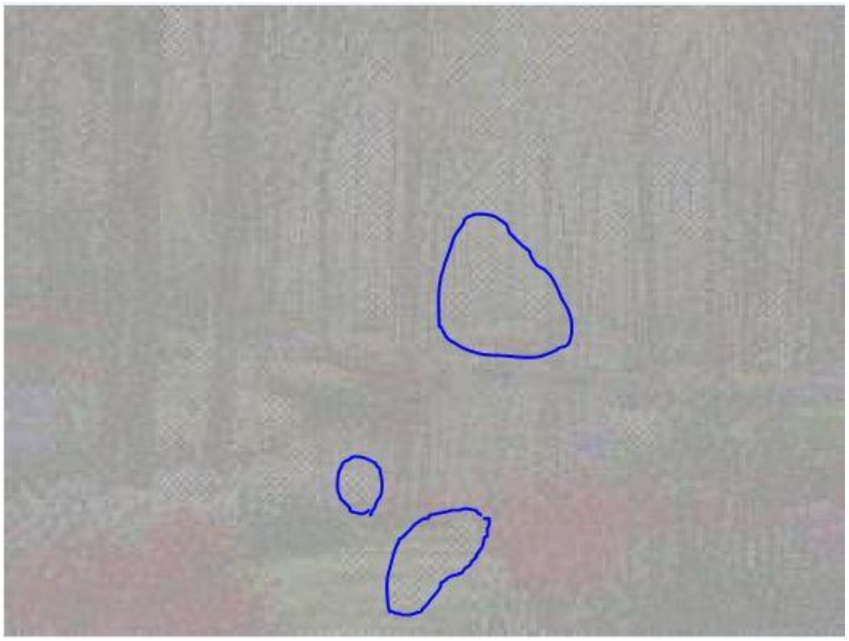

Figure 11. Generated output image with 20 error bits

Table 11. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
111100010**11**0**00**110**11**011**00**0**1110**1**0**1	-
10011000**10**1**1**01**0**0**10**10**10**100**00110**10	-
10**11111**00000**111111**0**11**0**11**00**11**0**1**0**1**	-
Corrected Data Displayed	-
01101001011010010110100101101001	7
00001111111100001111000000001111	29
01011010101001011010010101011010	30

Table 12. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
11001**011**001**00**1**11**11**1111**1**0**11**111**0**0**1	-
1011**010**0**101**110**01**10**0**0**1010**10**0**00**010**	-
11**1**1**10**0**010**0**0**0**010**1**101**1**1**1**1**1**101**101**1**	-
Corrected Data Displayed	-
00001111000011111111000011110000	20
00111100001111001100001111000011	22
00000000000000001111111111111111	16

4.7. Experiment seven

In this experiment, we induced 25 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 13 and the table showing the first few rows of good and bad data is shown in Table 13 and Table 14. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8 and when the number of errors in each row is more than 8 we are not able to recover every number perfectly and hence the image is getting blurred. The output image and the values in the table show that the image is not accurate. The number that recovered is not the actual number that needs to be recovered. The original image and the recovered image are shown below and the differences are highlighted in recovered image and compared in Figures 12 and Figure 13.

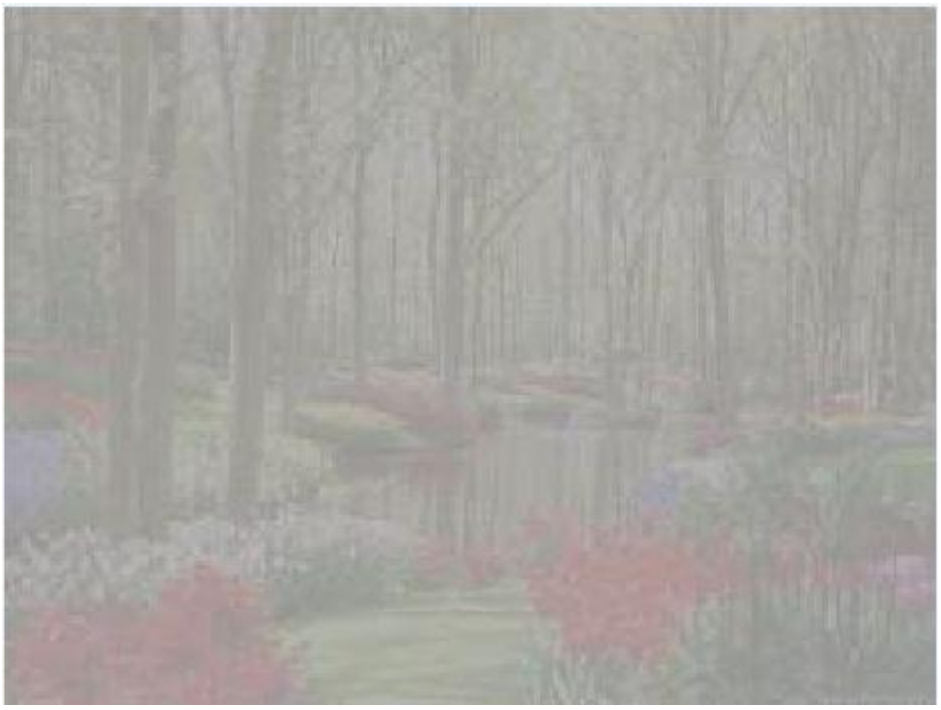

Figure 12. Original image used

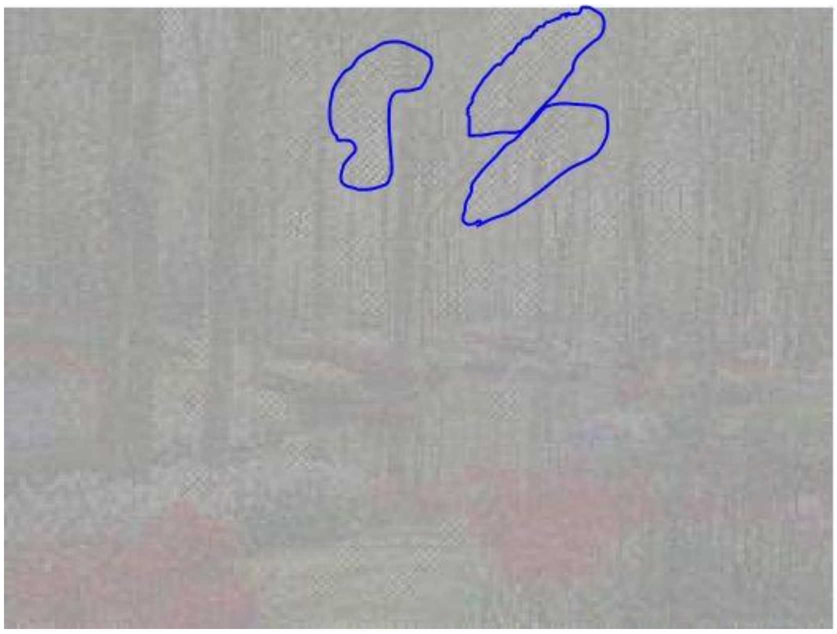

Figure 13. Generated output image with 25 error bits

Table 13. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
111100000**11**01**0**1**0**0**11**0**0000**0**1110000**	-
0**0**1**1**0**0**1**11001100110011**1**0110011**1**0**0	-
11111111110**11111**0000**111**0**1111**0**111**	-
Corrected Data Displayed	-
01100110011001100110011001100110	3
00110011001100111100110011001100	18
00001111000011110000111100001111	4

Table 14. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
0**10000**0**111000**10**10011**0**10000**00**1100**	-
10101**10**00**0**0**001010101101**11**10110**0**0**	-
1011**10101010**01**1001**1**1**10**0101010**0**01**	-
Corrected Data Displayed	-
00000000000000000000000000000000	0
00000000000000001111111111111111	16
00110011001100110011001100110011	2

4.8. Experiment eight

In this experiment, I randomly induced between 0-7 errors in each row of the input data and then the application will compare this erroneous data with the 32 X 32 Hadamard matrix to determine the original data that was meant to be transmitted. In this scenario the reconstructed image is shown in Figure 14 and the table showing the first few rows of good and bad data is shown in Table 15 and Table 16. I am doing these experiments to show that this approach is working perfectly when the number of errors in each row is less than 8. The output image and the values in the table show that the image is accurate. The number that recovered is the actual number of what it needs to be.

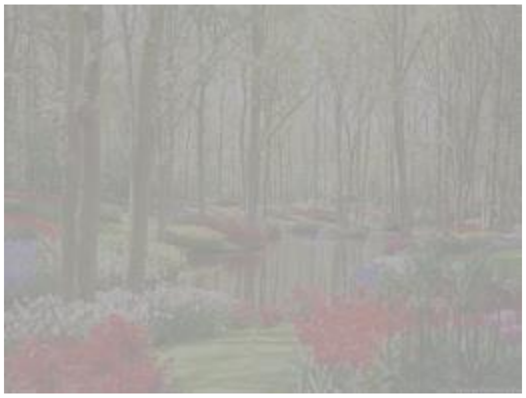

Figure 14. Generated output image with random error bits between 0-7

Table 15. Data transmitted and recovered for PIXEL 1

Data Transmitted	Row #
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0
Data Received	-
00001**0**11000011110**1**00111100001111	-
011001**0**0011001100111**1**0111**1**0110011**1**	-
00**1**00000**1**0**1**0000000000**1**0**1**000000000	-
Corrected Data Displayed	-
00001111000011110000111100001111	4
01100110011001100110011001100110	3
00000000000000000000000000000000	0

Table 16. Data transmitted and recovered for PIXEL 2

Data Transmitted	Row #
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17
Data Received	-
001**0**110**1**0011110011000**1**11**0**1000011	-
010110100**0**01101010**0**001011011**1**0**0**01	-
0**0**0101010101011**1**1101010101010101010	-
Corrected Data Displayed	-
00111100001111001100001111000011	22
01011010010110101010010110100101	21
01010101010101011010101010101010	17

5. CONCLUSION AND FUTURE WORK

In this paper, based on the experiments the application works perfectly by comparing the erroneous data with the 32x32 matrix in order to produce the desired output if the errors are in between 0 and 7. And when the number of errors in each row is more than 8 the results that were produced were not very accurate but the experiments show that the image recovered is still identifiable, the more the number of errors in each row the more blurred the image gets.

One thing that can be done and experimented is to increase the dimension of the Hadamard matrix to 64x64 or 256x256, by doing so we will have more rows, which means more numbers can be transmitted and even if the numbers of errors in each row increase the new application may do a better job in reconstructing the image than my application. One disadvantage though would be the increase in cost as we will have to transmit more bits for each number.

REFERENCES

1. Mitchell, G., (2009). *Investigation of Hamming, Reed-Solomon, and Turbo Forward Error Correcting Codes*: Adelphi, MD : Army Research Laboratory
2. Raaphorst, S., (2003). *Reed-Muller Codes.* Carleton University.
3. Cooke, B., (1999). *Reed Muller Error Correcting Codes*: MIT Undergraduate Journal of Mathematics Volume 1, MIT Department of Mathematics http://www-math.mit.edu/phase2/UJM/vol1/**COOKE**7FF.PDF
4. Wallace, H., (2003). *Using the Golay Error Detection and Correction Code*. Retrieved from http://www.aqdi.com/golay.htm
5. MacWilliams, F.J., & Sloane, N.J.A. (1977). *The Theory of Error-correcting Codes*, North-Holland, Amsterdam: North Holland Publishing Co.
6. Shlichta, P., (1979). Higher-dimensional Hadamard matrices. *IEEE Transactions on Information Theory, Vol. 25 Issue 5*, 566-572. doi:10.1109/TIT.1979.1056083
7. Djordjevic, I.B., Lei, Xu., Ting, W., & Cvijetic, M. (2008). GLDPC Codes with Reed-Muller component codes suitable for Optical Communications, *Communications Letters, IEEE, Vol. 12 Issue 9*, 684-686. doi:10.1109/LCOMM.2008.080590
8. Kangyu N., Datta S., Mahanti P., Roudenko S., & Cochran D. (2010). Using Reed-Muller sequences as deterministic compressed sensing matrices for image reconstruction, *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference.* Dallas, TX.
9. Tam, S., (2004). *Multiple Bit error Correction*. Retrieved from http://www.xilinx.com/support/documentation/application_notes/xapp715.pdf

10. Schwede, K., *Error detecting and Error Correcting Codes* [PDF document]. Retrieved from Lecture Notes Online Web site: http://www.personal.psu.edu/kes32/MichiganClasses/math217/Worksheets3/eccodes.pdf