Reliable and Efficient Transmission of Compressive-Sensed Electroencephalogram Signals

by

Patrick Rmeily

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2014

Abstract

As technologies around us are emerging at a rapid rate, wireless body sensor networks (WBSN)s are increasingly being deployed to provide comfort and safety to patients. WBSNs can monitor the patient's health and transmit the collected data to a remote location where it can be assessed.

Such data is collected and transmitted using low battery devices such as specialized sensors or even smart phones. To elongate the battery life, the energy spent on acquiring, processing and transmitting the data should be minimized. The thesis addresses the case of electroencephalogram (EEG) signals. It studies the energy spent in the sensor node, mainly in the processing stage, i.e. after acquiring the data and before transmitting it to a certain receiver-end in a wireless fashion. To minimize this energy, the number of bits to be processed and transmitted must be minimized. Compressive sampling (CS) is ideal for such a purpose as it requires minimal number of computations to compress a signal.

For transmitting the signals acquired by CS, we studied their quantization followed by 2 different schemes. Scheme 1 applies lossless Huffman coding for further compression that allows perfect reconstruction. This is followed by a Reed-Solomon (RS) code to protect the data from errors during transmission. Scheme 2 does not apply any further compression. It only

quantizes the data and applies the RS code to it.

Both schemes were then enhanced by adding an interleaver and de-interleaver that improved the results. The data was then sent in packets over a transmission channel which was simulated using a 2-state Markov model.

Under ideal channel conditions, Scheme 1 with Huffman compression decreased the total number of bits sent by 5.45 %. The best scheme however was scheme 2 followed by an interleaver. It achieved the best signal reconstruction results under normal or noisy channel conditions.

Preface

This thesis presents the research conducted by Patrick Rmeily, in collaboration with Professor Dr. Rabab K. Ward. I hereby declare that I am the first author of this thesis. The chapters in this thesis are based on work that I have conducted by myself and projects that I have submitted as assignments as part of my past coursework. Section 2.1.3 is based on the work done by my colleague at UBC, Mr Hesham Mahrous.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AWGN	Additive White Gaussian Noise
BCI	Brain Computer Interface
BSBL	Block-Sparse Bayesian Learning
BSBL-BO	Block-Sparse Bayesian Learning - Bounded Optimization
BCS	Binary Symmetric Channel
BW	Bandwidth
CR	Compression Ratio
CS	Compressed Sensing
DCT	Discrete Cosine Transform
DM	Delta Modulation
DMC	Dynamic Markov Compression
DWT	Discrete Wavelet Transform
EEG	Electroencephalogram
FEC	Forward Error Correcting
GSM	Global System for Mobile communications
IID	Independent and Identically Distributed
LDPC	Low-Density Parity-Check
NMSE	Normalized Mean Square Error
PCM	Pulse-Code Modulation
RIP	Restricted Isometry Property
RS	Reed-Solomon
RV	Random Variable
SNR	Signal-to-Noise Ratio
SSIM	Structural SIMilarity
SVM	Single Measurement Vector
TDM	Time Division Multiplexing
WBSN	Wireless Body Sensor Network
WT	Wavelet Transform

Acknowledgments

There are many people that I would like to thank and acknowledge for their support and help, but first and foremost I would like to thank my supervisor Professor Rabab Ward for her unconditional support and encouragement. I cannot express the gratitude and appreciation I have for her professionalism, body of knowledge, achievements, and most importantly her kindness. I will never forget that kindness of hers, and I can say that this is what made all the difference for me in having her as my supervisor.

I would like to thank my lab mates, and specifically Simon, Hesham and Hiba for all the laughs we had together and all the stories we shared. There have been lots of ups and downs in the last couple of years, and having you around helped make the downs easier to go through!

Last but definitely not least, I would like to thank my family and closest friends for their continuous belief in me, even when things seemed to always go against what I was hoping for at that time. But as life teaches us so often, you need to accept the things you cannot change and with time the reason for such events becomes clearer. So thank you life for all those lessons, and thank you for sending me all those beautiful people to enrich my journey!

Chapter 1

Introduction

1.1 Telemetry for Medicinal Purposes

One of the basic human rights is that of having a good health care system that is properly functional to ensure the safety and health of human beings. Most countries around the world spend huge amounts of money for research into health, especially that many of the advanced countries have witnessed increases in their elderly population [4].

The current age demography shows that more people suffer from chronic diseases which come naturally with age. However, the amount of chronic diseases is increasing drastically in younger people due to unhealthy eating habits and lifestyles [16]. This creates a bigger financial burden for the health care system, estimated to be in the billions of dollars. Chronic diseases are not something that younger people would usually suspect they have. This means that constant medical check-ups and supervision could be beneficial. However, that is practically impossible to do as it requires that each patient has one caretaker specifically assigned to him/her. Even if that were to be possible, the costs alone would be far too much to deal with. Therefore solutions that are inexpensive - or at the least cost-effective - need to be presented.

Wireless body sensor nodes (WBSN)s are one solution that is gaining a lot of ground in the health care industry [27], [3]. They allow patients to continuously monitor themselves, from the comfort of their homes, while remote caregivers can be at clinics for example. By using sensors that are attached to the body, WBSNs allow patients to check on vital signs such as their heart rate, diabetes and brain activity. Considering that WBSNs are cost-effective, efficient, and can be produced at a large scale, the solution that the health care industry has been looking for is now available. The fact that such devices could save lives forms an enough reason for the industry to spend or carry a lot of research in this area, to further improve the services that they can provide.

1.2 Conditions for Telemetry

As exciting as WBSNs sound, it is not as easy to design such devices as it seems. There are some conditions and restrictions on parameters that are essential in the designing of WBSNs which will be discussed now.

In many applications, telemedicine uses data channels that have low bandwidth (BW). The BW of a channel dictates how many bits per second one can send through the channel and since low BW channels have low bit rates, we already are faced with the first restriction of not being able to send large chunks of data all at once.

To be able to directly monitor one's vital signs, the results have to reach the remote center in real-time or close to real-time. This means that the less data we send, the faster the signal will be reconstructed and viewed at

the receiver end. To achieve that, one must try and make sure that the data to be sent is composed of only useful data that is also not redundant. This poses another condition on the processing stage of the WBSN encoders.

Another restriction, which is basically the most costly one, is that of the processing power at the WBSN nodes. WBSNs usually use batteries that do not last long. Think of a cell phone battery for example. If you open applications that require too much processing power, the battery will unfortunately die out pretty quickly. Hence, for WBSNs to be convenient and practical to use, the algorithms executed inside them should be designed so that their processing time is extremely fast. Therefore the battery would not get drained out and the patient wouldn't have to change batteries often (since WBSNs are operated using a battery).

This previous restriction leads us to another costly one, which is the energy needed to transmit all the data. As mentioned earlier, lesser amounts of transmitted data lead to a more real-time viewing of the signal at the receiver end. This is not the only benefit that comes out when the data amount to be sent is decreased. Saving up on energy is just as important of a benefit, if not even more, because the lesser the data to be sent, the lesser the energy that will be needed, and hence, the more the battery's lifespan is preserved [27].

1.3 Electroencephalogram Signals

Even though the work frame in this thesis can be applied to many signals, the focus has been on electroencephalogram (EEG) signals, which are signals

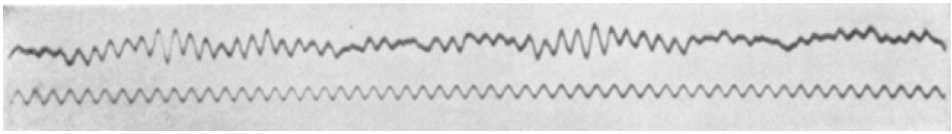

Figure 1.1: Sample of an EEG output

that convey the electrical brain activity and are acquired by non-invasive sensors placed on the patient's head. They are then transmitted wirelessly.

The information contained in the EEG signals obtained from multiple electrode sensors can help diagnose epilepsy, sleep disorders, strokes, Alzheimer's, tumors and comas among other illnesses. EEG monitoring needs on average 30 minutes of recorded signals for specialists to be able to detect a disorder. In the case of some disorders, the monitoring should be done for a longer period of time [13], [14]. This makes the use of WBSNs at home much more favorable than spending time at the hospital. Figure 1.1 shows a sample of a recorded EEG output.

Such WBSNs, specifically for EEG signals, have a certain number of electrodes placed on the patient's head as specified in the international 10-20 system. The number of electrodes used can vary depending on the purpose. Each of these electrodes results in an output signal, and that sensor is known as an EEG channel. EEG electrodes provide high temporal resolution. However, this is not the case for spatial resolution (i.e. resolution of the EEG signal over the scalp) which can be a limitation. To increase the spatial resolution, the number of sensors used in the EEG measurements has to be increased.

Before transmitting the EEG data, the channels have to be processed as

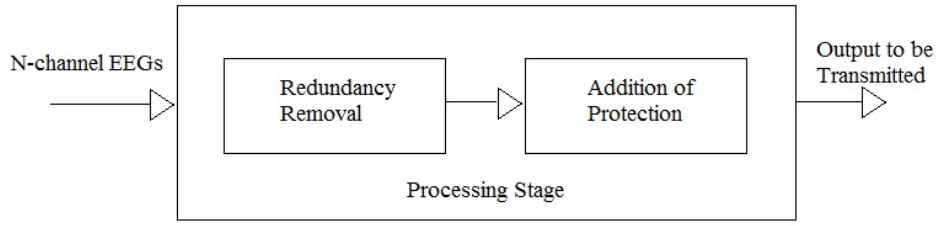

Figure 1.2: Block diagram for the processing of EEGs using WBSN

shown in Fig. 1.2. All the channels are collected and the resulting output forms a one stream of data. This data is then processed in two parts. The first part removes any redundancy that may be present in the data in order to transmit the relevant data only, and the second part adds protection bits to the data to protect it against the channel noise that can greatly distort it. Finally this data is transmitted using Bluetooth.

Since this processing is done within the WBSN [8], the complexity should be minimal so as not to drain out the system battery. That is not the case on the receiver-end where the computations to retrieve and reconstruct the original data sent can be more complex, since there are no restrictions placed on the energy consumption at this receiver-end [7].

1.4 Aim and Motivation of the Thesis

The aim of this thesis is to minimize the energy spent at the sensor node. The energy is spent on acquiring the data, processing it so it can be transmitted and then wirelessly transmitting it. To minimize the transmission energy, as few bits as possible should be transmitted. Therefore the acquired

data should be compressed before its transmission so that it is represented by as less bits as possible and it can be reconstructed from these bits. The compression should also require as little energy as possible for use in applications such as WBSNs that transmit EEG data [1]. Compressed Sampling is ideal for this purpose. With present day technology however, it is not yet possible to sample the analog signal directly using CS. Therefore we resample the acquired signal using CS. We then quantize the CS resulting data using the minimum number of levels that allow good quality reconstruction. To transmit the signals, we aim to come up with a joint source and channel coding combination that can be used for EEG WBSN purposes. As already discussed, the power consumed by the algorithms that process the data at the encoder side (the WBSN) should be decreased as much as possible. We also need the signals to be reconstructed and displayed in real-time, and we have to do that using a low bandwidth channel, that is also noisy. So the goal is to find a suitable framework that can address these demands.

Processed signals need to be quantized. This process already introduces an error, known as the quantization error. Depending on the kind of signals being sent, the bits required for quantization can be lowered. In other words, some signals require the smallest variations in the amplitudes to be reconstructed clearly because such variations are pivotal to the diagnosis of a patient. In such cases, a higher number of bits can be used for quantization, which leads to more quantization levels that allow for such variations to be captured. Other signals can allow for a more flexible quantization error and hence the number of quantizing bits can be smaller.

When applying source coding, a lossless compression algorithm is desired

for the simple fact that it can perfectly reconstruct the transmitted data. Lossless algorithms however suffer from the high number of computations required to compress the information. The Huffman code is usually the algorithm of choice as it can be found in numerous applications that cover several fields. In comparison, lossy-algorithms are simpler in terms of their computational requirements.

For channel coding purposes, there are two main types of algorithms: block and convolutional algorithms. Both have very powerful existing algorithms. However, when dealing with real-time applications block codes are usually more prevalently used because they can be computed faster.

The work in this thesis is motivated by the use of compressed sensing (CS), which is a technique that samples and compresses at the same time. CS was discovered recently. The sampling is done at a rate far below the Nyquist rate, which for many decades was not explored. Because CS samples the data by taking linear projections of the raw data [7], it requires little energy at the transmitter side. The computationally complex part of CS is at the receiver end. For our application however, this does not constitute a problem as there is no limitation on the power in the receiver end. Those two properties make CS a perfect algorithm of choice to be applied in WBSN applications.

In summary, this thesis aims to find a good combination between compressing the data, quantizing and adding protection to it while taking into consideration the computational effort and the energy spent at the sensor node, the noise in the channel and the reconstruction quality. The thesis will study the results of each scheme applied and propose improvements for

future work.

1.5 Thesis Structure

The rest of the thesis is divided into 4 chapters. The first chapter, chapter 2, gives a theoretical review of the concepts used in this work. Section 2.1 of this chapter introduces the concept of compressed sensing and presents the theory behind the encoding and decoding stages of this algorithm. Section 2.2 introduces the concept of source coding and information theory. It goes into the detail of one of the most famous source coding algorithms, the Huffman code. Section 2.3 deals with the transmission channel, its properties and how to model the Bluetooth channel that is used for sending the data. Section 2.4 explains the need for channel coding algorithms in any transmission scheme. It gives a comparison between two families of codes: block and convolutional codes. Its also discusses interleaving and its benefits. This is followed by chapter 3 which presents the different schemes used for the transmission of the data and then chapter 4 which analyzes the results obtained from the experiments. Finally, the whole results and observations are summarized in chapter 5.

Throughout the thesis, it is important to point out that bold lowercase letters mean the variable used is a vector, whereas bold uppercase letters mean it is a matrix. Scalar quantities are represented in normal letters.

Chapter 2

Literature Review and Background Theory

2.1 Compressed Sensing

Signals in life are usually of an analog, continuous nature. When we want to discretize them, we have to apply a technique called sampling, which as the name implies, takes samples from the signal, and these samples can then be analyzed or manipulated for whatever purpose intended. Then when these samples are dealt with, they are retransformed into an analog continuous signal by interpolation.

The samples though could not be taken randomly and without any constrictions. There is a whole area of study behind it called the Nyquist-Shannon sampling theory, or in short, the sampling theory [26]. What it states is that for a signal $x(t)$ we must take at least two samples per cycle of the highest frequency component of that signal [21]. In mathematical term, if the highest frequency of the signal $x(t)$ is

$$2f_M \leq f_S \tag{2.1}$$

where f_S is the sampling frequency. If this condition is not satisfied, aliasing would occur causing the signal to not be properly reconstructed from the samples taken. In cases where the bandwidth is very large and the constraints that are imposed on the sampling architecture are a bit too much to cope with, the traditional sampling theorem proves to be too demanding. In other cases where the signals are sparse in their nature, applying Shannon's sampling theorem produces a large amount of redundancy in the samples, which are costly to wirelessly transmit as is the case with telemonitoring where the bandwidth is small and one needs to make the most of it, therefore using that theorem limits the sensor nodes lifetime.

This theorem has been lately challenged by the advent of a new technique called compressed sensing which will be discussed in this chapter.

For WBSNs that were discussed in chapter 1, increasing the battery's lifetime and achieving high compression ratios (CR)s is important, as well as the cost of the device. The cheaper the device, the more willing the patients would be to purchase them. This also signifies that the hardware cost should be low, which in turn means that the encoding algorithm should have low complexity, leaving the complexity to the decoder at the remote center or laptop [27].

As far as choice of transforms for signal reconstruction go, the wavelet transform (WT) was usually the choice to use for excellent signal reconstruction at the receiver. However, the WT compression fails to satisfy such constraints, which has led to the introduction of CS, which is a compression technique that depends on the sparsity of the signals on hand, and in comparison to the WT, reduces energy consumption while maintaining a

competitive data compression ratio, and largely reducing the device's cost as was shown in [16].

Basically CS allows us to take far fewer samples from a signal than is actually needed by conventional techniques and still manage to properly reconstruct that signal. To do so, two properties must be satisfied: that of sparsity and that of incoherence [5].

What is meant by sparsity is the idea that for a signal of finite length, the information rate contained in that signal is much less than its length suggests, hence making it compressible. That is a point that allows CS to not only act as a sampling technique, but also a compressing algorithm, since it drastically decreases the amount of data taken from the signal.

Incoherence on the other hand refers to the duality between the time and frequency domain representations of the signal where in one domain, that representation is spread out, and in the other domain, it is sparse [5].

Taking advantage of those two properties allows for a sampling and compression algorithm that is practical and energy efficient for applications where the encoding energy is quite limited. Both sampling and compression are replaced by a randomized sub-sampling technique that linearly encodes the samples without the need of high resolution for the reconstruction of the signal [15]. Two matrices are essential for CS: the linear measurement sensing matrix and the dictionary matrix.

2.1.1 Encoder

CS starts by collecting M samples of a signal $\boldsymbol{x}$ whose dimension is Nx1, where $M \ll N$ using simple analogue measurement waveforms, thus sam-

pling and compressing at the same time which helps in removing a large part of the digital architecture. The M samples are collected using the simple measurement vectors $\boldsymbol{\phi}_i$, where $1 \leq i \leq M$ and $\boldsymbol{\phi}_i$ is of length N. Each of those vectors has k-nonzero elements. The transpose of these vectors form the rows of the sensing matrix $\boldsymbol{\Phi}$ which compresses the signal $\boldsymbol{x}$ as follows:

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{x}. \tag{2.2}$$

The output signal $\boldsymbol{y}$ has a dimension of Mx1 where $M \ll N$ [16].

2.1.2 Decoder

The original signal $\boldsymbol{x}$ is recovered from the compressed data y using the sensing matrix $\boldsymbol{\Phi}$ and assuming that x is actually a sparse signal. If $\boldsymbol{x}$ is not sparse (as is the situation with EEG signals), but it is sparse in another domain, that is there exists a matrix $\boldsymbol{D}$ such that $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{z}$, where $\boldsymbol{z}$ is sparse and $\boldsymbol{D}$ is a dictionary (transform) matrix of dimension NxN, then equation 2.2 can now be rewritten as

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{D}\boldsymbol{z} \tag{2.3}$$

Thus in this case, the CS algorithms would first have to recover $\boldsymbol{z}$ by using $\boldsymbol{y}$ and $\boldsymbol{\Phi}\boldsymbol{D}$, and then recover the original signal $\boldsymbol{x}$ from the relation $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{z}$.

When CS is used in applications such as telemonitoring, the signal $\boldsymbol{x}$ is first compressed by the sensors according to equation 2.2. The compressed data $\boldsymbol{y}$ is then transmitted to a server that may reside in a clinic or another

facility. The original data $\boldsymbol{x}$ is recovered from $\boldsymbol{y}$ according to equation 2.3, where the matrix $\boldsymbol{\Phi}$ is known and $\boldsymbol{D}$ is determined from the nature of $\boldsymbol{x}$.

To see the advantages of this method, all one has to do is go deeper into the mathematics that is involved in CS. The simple linear sampling strategy applied by CS yields results that are marginally off the optimal adaptive strategy (which is too complex) [23].

To guarantee the robust and efficient recovery of any S-sparse vector $\boldsymbol{x}$, the sensing matrix $\boldsymbol{\Phi}$ must obey the key restricted isometry property (RIP)

$$(1 - \delta_S)\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{\Phi}\boldsymbol{D}\boldsymbol{z}\|_2 \leq (1 + \delta_S)\|\boldsymbol{x}\|_2 \tag{2.4}$$

where δ_S is the isometry constant of the matrix $\boldsymbol{\Phi}$ - which must not be too close to one ($\delta_S < 1$). The RIP ensures that the energy of the signal is bounded from below and above. The smaller δ_S is, the more energy is captured and the more stable the inversion of $\boldsymbol{\Phi}\boldsymbol{D}$ is [5].

The key factor that bounds the restricted isometry constant δ_S is the mutual coherence amongst the columns of $\boldsymbol{\Phi}\boldsymbol{D}$ as follows [11]:

$$\delta_S \leq (k - 1)\mu \tag{2.5}$$

where $\mu = max_{1 \leq i \neq j \leq N} < \boldsymbol{\phi}_i, \boldsymbol{d}_j >$, and k is the number of non-zero entries in $\boldsymbol{x}$.

A good choice for the measurement matrix $\boldsymbol{\Phi}$ is a random matrix with independent and identically distributed (iid) entries [9].

The signal $\boldsymbol{x}$ is recovered at the decoder side via a convex optimization problem. One major advantage of decoding by optimization, is that CS

decoders are more robust to noise and quantization errors, which helps in further enhancing compression and reduces the demands on the digital back-end and the on-board memory.

If the RIP in equation 2.4 holds, then a reconstruction that is faithful to the original signal can be accomplished by solving the convex optimization problem:

$$\bar{\boldsymbol{z}} = \min_z \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{z}\|_2^2 \ + \lambda g(z); \tag{2.6}$$

where λ is a regularization term and $g(z)$ is the penalty term, which is a function of z [15]. The regularization and penalty terms are needed to account for the noise factor introduced by the l_2 norm. After solving for $\bar{z}$, we can get the reconstructed signal as follows $\bar{\boldsymbol{x}} = \boldsymbol{D}\bar{\boldsymbol{z}}$.

2.1.3 Block-Sparse Bayesian Learning

The problem of EEG signals is that they are not sparse by nature, neither in the time domain nor in a transform domain. Since the CS theory is developed for signals that are sparse or have sparse representation coefficients in some transform domain, the existing CS algorithms cannot achieve good signal recovery for EEG signals. Therefore, the aim in [27] was to find an algorithm capable of recovering the EEG signal from its compressed sensing data using equation 2.3.

The block-sparse Bayesian learning (BSBL) algorithm was developed to try and deal with that. BSBL uses a general dictionary to reconstruct a non-sparse signal instead of finding an optimal dictionary for a specific type

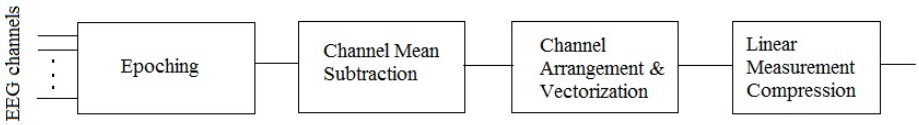

Figure 2.1: Block diagram of the adjusted algorithm

of data. The BSBL algorithm was proven in [27] to be effective for the compressed sensing and reconstruction of non-sparse signals that also have no distinct block structure. It uses the discrete cosine transform (DCT) and discrete wavelet transform (DWT) as dictionaries.

BSBL addresses the single channel signal case, also called the single measurement vector (SVM) case. The algorithm exploits the intra-correlations within the structure of an SMV EEG channel. Let us denote the number of EEG channels as P. For the case when $P = 1$ (the SMV case), the data is encoded using the algorithm explained in section 2.1.1 with a few processing steps before compressing the data. For the multi-channel case when $P > 1$, BSBL reconstructs each column of the data one by one.

It is in the way with which the original data is reconstructed that the BSBL differs from traditional CS algorithms. CS exploits the sparsity of the signals when reconstructing them. BSBL exploits either the temporal correlation in the signal or its block-sparsity [27].

Assume a block sparse signal of the form $\boldsymbol{x} = [x_1, ..., x_{h_1}, ..., x_{h_{g-1}+1}, ..., x_{h_g}]^T$, where $[x_1, ..., x_{h_1}] = x_1^T$ up to $[x_{h_{g-1}+1}, ..., x_{h_g}] = x_g^T$, and x_i has dimension $h_i \times 1$, $i = 1...g$.

BSBL models each block $x_i \in R^{d_i \times 1}$ as a parametrized multivariate

Gaussian distribution of the form

$$p(x_i; \gamma_i, \boldsymbol{B_i}) \sim N(0, \gamma_i \boldsymbol{B_i}), \tag{2.7}$$

where $i = 1...g$. γ_i is a non-negative parameter that controls the block sparsity of $\boldsymbol{x}$. If $\gamma_i = 0$, then the i^{th} block of $\boldsymbol{x}$ is all zero. $\boldsymbol{B_i}$ is a positive definite matrix with dimension $d_i \times d_i$. It captures the correlation structure of the i^{th} block.

Assuming that the blocks are mutually uncorrelated, the prior of $\boldsymbol{x}$ according to 2.7 is $p(\boldsymbol{x}) \sim N(0, \Sigma_0)$ where Σ_0 is a block diagonal matrix with the i^{th} principal block given by $\gamma_i \boldsymbol{B_i}$. The noise vector is assumed to satisfy a multivariate Gaussian distribution $p(\boldsymbol{v}) \sim N(0, \lambda I)$, where λ is a positive scalar and I is the identity matrix.

Now, the estimate of $\boldsymbol{x}$ can be obtained by the Maximum-A-Posterior estimation, assuming that all the parameters λ and $(\gamma_i, \boldsymbol{B}_i)_1^g$ have been estimated using the Type-II maximum likelihood estimation [28].

The reconstruction of the data is done using a bound optimization BSBL algorithm. This algorithm has the ability to exploit the intra-block correlation through the estimation of the matrices $\boldsymbol{B_i}$. Even though the user needs to determine the block partition, it still works well for arbitrary block partitions.

The work done in [15] improves the BSBL algorithm presented in [29] by investigating both the intra-correlation and inter-correlation of multivariate EEG channels and processing all channels simultaneously. This approach processes the data epoch per epoch and then takes the output for every

epoch and forms a stream of data, i.e. it is divided into equal blocks of size N, where N is chosen to be equal to 256 samples per epoch. Assuming there are P channels, the stream would be of size $P \times N$. This step is known as epoching, and it is the first block in figure 2.1.

The second block subtracts the channel mean from every channel, hence normalizing each channel. Such a step leads to better results by the algorithm because it removes the biasing that affects the data by the acquisition amplifiers. The means are added again after the reconstruction of the data.

The third block in figure 2.1 exploits the intra-correlation and inter-correlation of the P channels. A random channel is first selected. Then its cross correlation is computed among other channels. The three channels with the highest correlation among the others are placed next to one another along the rows of the matrix $\boldsymbol{X}$. Next, the sorted channels are excluded and the previous steps in this block are repeated again until all channels are sorted. Once the blocks are reordered into P rows, they are vectorized to form one output, $\boldsymbol{x} \in R^{PN \times 1}$, using the equation

$$VEC(X) = [a_{1,1}, a_{2,1}, ..., a_{L,1}, a_{1,2}, a_{2,2}, ..., a_{L,2}, ..., a_{1,P}, a_{2,P}, ..., a_{L,P}]^T, \tag{2.8}$$

where a is the cell matrix of the data matrix X of dimension $P \times N$.

The last block deals with the compression of the data, which compresses all the channel outputs at once instead of one at a time. It was shown in [11] that compression that can lead to perfect reconstruction depends on the degree of coherence between matrices $\boldsymbol{\Phi}$ and $\boldsymbol{D}$, and the higher that

incoherence, the larger the achievable compression ratio. The main condition for this to happen is that $\mathbf{\Phi}$ must be iid.

2.2 Source Coding

When one wishes to transmit data from one end to another, it is very important to ensure that the transmitted signal only carries the necessary information and nothing more, especially in cases where we have constraints on the data rate we are transmitting with. Therefore, compressing and deleting unwanted data is an essential step before sending the signal. That is what source coding does. To define source coding in just one sentence, we can say that it is an algorithm that is used to get rid of all redundant and irrelevant bits in the data. Before going into a deeper and more mathematical explanation of what that means, it is necessary to define what is meant by redundant and irrelevant bits.

Redundancy means something that is repetitive, hence why source coding helps us get rid of that. This means that when the representation of a symbol is not efficient enough, one can decrease the number of bits used to represent this symbol. This of course is also dependent on the probability of occurrence of this symbol as will be demonstrated later. The second term, irrelevancy, can also be directly interpreted as something that is of no importance to the correct reception and understanding of the data. Therefore, this irrelevant part can be simply discarded (by the use of filters for example). However, out of these two terms, the one that is most important to us is redundancy, and most source coding algorithms work on taking advantage

of it [10].

2.2.1 Overview of Information Theory

In this section we consider a more in-depth explanation of the importance of source coding. It is a means that allows us to achieve compression of data, which reduces the amount of bits that need to be transmitted. As mentioned in the introduction of this chapter, the probability of the occurrence of a symbol is of great importance, because it is what can be targeted in order to reduce the redundancy. Therefore one must know the statistics of the symbols that are being transmitted and the probability of occurrence of each symbol. This is the basic building block of all information theory.

Let x_i and y_j be the outcome observed for the two random variables X and Y. From their probabilities, we can get the first important parameter in information theory, which is the information of an event. There are two types of information: the first is the mutual information between x_i and y_j and the second is the self-information. Mathematically, these are expressed as

$$I(x_i; y_j) = \log_2 \frac{P(x_i|y_j)}{P(x_i)} \tag{2.9}$$

$$I(x_i) = \log_2 \frac{1}{P(x_i)} = -\log_2 P(x_i) \tag{2.10}$$

where equation 2.9 and equation 2.10 are the mutual and self-information respectively.

The self-information can be the same as the mutual information in the

case where the occurrence of the event $Y = y_j$ uniquely determines that of the event $X = x_i$. In this case, the conditional probability $P(x_i|y_j)$ is unity and hence we get the self-probability. Another interesting scenario is when the two random variable X and Y are statistically independent. Here, the conditional probability would be equal to $P(x_i)$ and hence the mutual information would be 0. Also worth noting is that from equation 2.10, it is clear that the higher the probability of the event, the less the information that is being conveyed, and the lower the probability, the higher the information being conveyed.

The second important parameter that we need to know is the entropy or the average self-information, which gives an indication on how well the source coding performance is. The entropy is defined as

$$H(X) = \sum_{i=1}^{n} P(x_i) I(x_i). \tag{2.11}$$

If we replace the equation 2.10 in equation 2.11, then we would get the following:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{2.12}$$

where the unit is bits/symbol in both equations 2.11 and 2.12.

Figure 2.2 shows the entropy plotted against the probability of an event of a random variable. It can be seen that the entropy is maximum when the probability is equal to half. In other words, the entropy would be maximum when all the symbols are transmitted with equal probability.

In order to see how much redundancy is present in a certain source

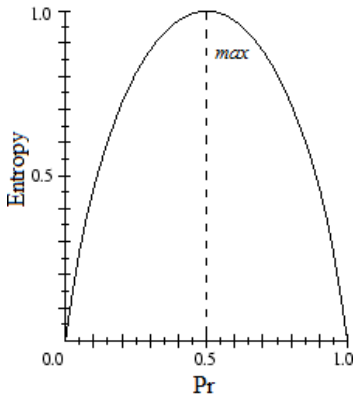

Figure 2.2: Plot of the entropy w.r.t. the probability of a random variable

coding scheme, one has to simply calculate the maximum entropy and then subtract from it the entropy of the scheme used. This is defined in equation 2.13. The smaller the difference is, the less is the redundancy available in the symbols being transmitted. In such cases, the complexity of the source coding algorithms would come into play, because if the system designed has limitations on its power or computational ability, and the redundancy is minimal, then the following question arises. Is it worth it to use such an algorithm for minimal improvements, but at the same time consume more resources? The answer to this question will be clearly stated in chapter 4.

$$Redundancy = H_{max} - H(X) \tag{2.13}$$

Another parameter of importance in information theory is the average code-word length L, defined as

$$L = \sum_{i=1}^{n} L(x_i)P(x_i). \tag{2.14}$$

This parameter allows us to see the difference in the code-word length when using a source coding algorithm compared to when the data is sent as is. Mostly, when source coding is used, L decreases, which is to be expected, since the task of the algorithm is to decrease the redundancy.

2.2.2 Huffman Code

Perhaps the best way to show the efficiency and advantage of source coding is to simply give an example about a source coding algorithm. The algorithm of choice here will be the Huffman Code. The basic idea behind Huffman coding is to assign the symbol with the highest probability of occurrence the shortest code-word, and the least probable symbol the longest code-word, all the while making sure that the combinations are unique and would not be confused with other symbols upon receiving them. This is important or else the code would be invalid.

The algorithm will be explained step by step. It has 4 stages presented below:

1- The symbols are sorted according to their probability in descending order.

2- Add the probability of the two lowest probabilities and assign a 1 to the lower symbol of the two and a 0 to the higher one.

3- Repeat steps 1 and 2 for the probabilities till you end up having only two values.

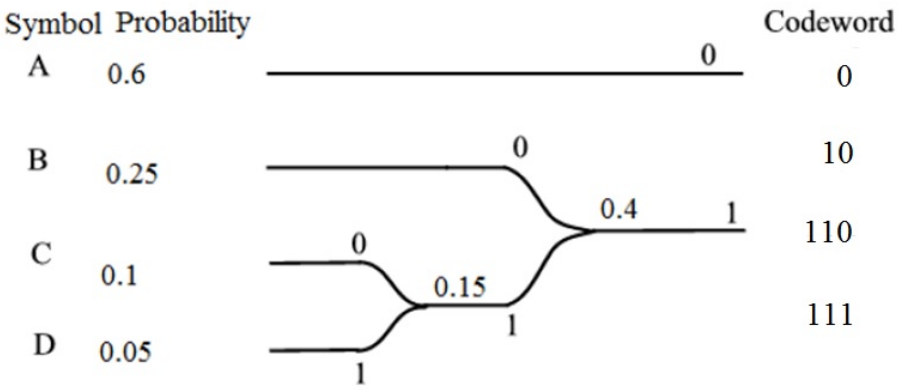

Figure 2.3: Example of how a Huffman algorithm works

4- Then move from right to left column-wise and symbol-wise to get the representation of each symbol.

The following numerical example will help show the advantages of the Huffman code. Let us consider four symbols A, B, C and D with probabilities 0.6, 0.25, 0.1 and 0.05 respectively. Figure 2.3 below shows the graphical implementation of the algorithm presented above.

After applying the 4 steps of the algorithm, symbol A, which had the highest probability, ended up being assigned only one bit, whereas the symbols C and D which had the lowest probabilities were assigned three bits each. Had we chosen not to apply source coding, since we have 2^2 symbols, then the logical thing to do would be to give each symbol a two-bit representation, say 00, 01, 10, 11 for A, B, C and D respectively. Let us take any transmitted sequence abiding by the above probabilities, that is we will transmit 40 symbols (24 A, 10 B, 4 C and 2 D):

$AAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBCCCCDD.$

Now let us replace each symbol by its bit representation first in the case where no source coding is used and then directly after it in the case where Huffman encoding was used, and we get:

Without Huffman Encoding:

0000000000000000000000000000000000000000000000000010101010101010101011010101011111

and after Huffman Encoding:

00000000000000000000000010101010101010101011011011011011111111.

In the former scenario, for 40 transmitted symbols, we get a total of $40x2 = 80\ bits$, whereas for the latter scenario, we would get a total of $24x1 + 10x2 + 4x3 + 2x3 = 62\ bits$, hence a decrease of 22.5% in the amount of bits transmitted.

From the above values obtained, it is easy then to calculate the average code-word length L as follows:

$L = 2x0.5 + 2x0.25 + 2x0.125 + 2x0.125 = 2\ bits/symbol.$

$L_{huffman} = 1x0.6 + 2x0.25 + 3x0.1 + 3x0.05 = 1.5\ bits/symbol.$

The above result reflects the improvement that source coding can introduce, since the average length of one code-word decreased from 2 bits to 1.5 bits, or 25%!

The Huffman code is an algorithm applied for the compression of digital data. It is one of many such schemes belonging to the lossless data compression algorithms, as opposed to lossy data compression algorithms.

The difference between the two types of algorithms is in both the amount of compression achieved and the quality of the reconstructed signal. For lossless algorithms, the compression ratio achieved is less than when a lossy algorithm is used. On the other hand, they are able to achieve perfect reconstruction at the receiver end, something that can not be done by the lossy algorithms. Some signals do not afford losing any of its accuracy (remember that analog signals already lose some of their resolution due to the quantization error introduced by digitizing the signal), hence the reason why lossless algorithms are preferred. However, these codes require more computational processing than their lossy counterparts.

The above algorithm is a digital source coding technique. There exist several important analog source coding techniques, with the most popular being the Pulse-Code Modulation (PCM).

The concept of PCM is very simple. It converts an analog signal into a digital one by first sampling the signal, then quantizing it, and finally encoding each quantized value by a certain number of binary bits.

Another important analog source coding technique is the Differential pulse-code modulation (DPCM). The concept of DPCM is based on the correlation between successive samples that are sampled at the Nyquist rate or faster. In such cases, the correlation is high, that is, the average change in the amplitude of the successive samples is relatively small, and DPCM makes use of this correlation [22]. Also worth mentioning is the Delta modulation (DM) scheme, which is basically just a simplified form of the DPCM. However, we will not delve into those two schemes as the intention of mentioning them is simply to bring them to the attention of the reader.

2.3 Transmission Channel

The transmission process consists of three main building blocks: the transmitter, the receiver and the channel. The transmitter side deals with the encoding of the data whereas the receiver decodes it. This section explains the part that is in the middle: the channel and its effects on the transmission process. To help in the understanding of how a channel works, examples of a few well-known channel model will be discussed.

2.3.1 Channel Models

A channel is basically a physical medium through which a signal is transmitted. During transmission in such a medium, errors are added to the transmitted data, hence corrupting it. A channel can be any physical medium where data travels through on its way from the transmitter to the receiver. It can be the atmosphere in the case of wireless transmission, or wire lines and optical fiber cables in the case of telephone channels, among other possible mediums [22].

An ideal channel is one in which the signal that is transmitted arrives exactly as is at the receiver end where it is reconstructed. However, in real life that is never the case. In the simplest of cases, the channel will add in one way or another noise that is usually considered white, and hence the first channel we will discuss is the Additive White Gaussian Noise (AWGN) channel.

This is a type of channel where the received signal is simply the sum of the original signal and added noise, as defined in equation 2.15.

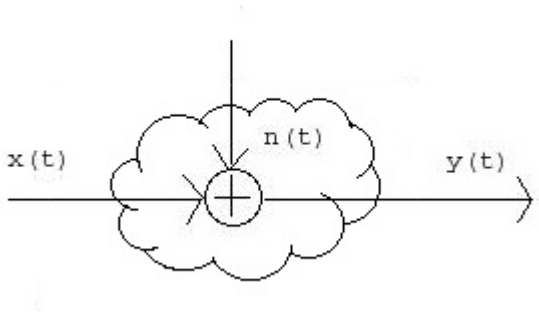

Figure 2.4: AWGN channel model

$$y(t) = x(t) + n(t) \tag{2.15}$$

where $y(t)$ is the received signal, $x(t)$ the sent signal and $n(t)$ the added noise. This equation can be easily transformed into the model shown in figure 2.4 below. The added white noise has a constant power spectral density, which means it has the same value over all frequencies.

This channel model does not take into consideration the phenomena of fading, interference, dispersion, nonlinearity or frequency selectivity. However, it gives a mathematical model that is useful in understanding the behavior of a system before such phenomena are taken into consideration [2].

All of the aforementioned phenomena occur frequently depending on the channel and the environment among other variables. The most common is fading. Fading occurs normally due to multi-path propagation, which is created because of the presence of reflectors that lie in the path between

the transmitter and the receiver. This causes interference with the original signal and it leads to attenuation, phase shift and delay in the original signal. Frequency selectivity is a kind of fading as well. It is caused when the signal partially cancels itself when multi-path exists. Dispersion is when the signal hits an object, such as the branches of a tree and it gets dispersed or deflected into several directions, which could lead to more paths due to more reflections of the signal.

Some other common examples of channel models will be given. We start things off with the simplest of all, the binary symmetric channel (BSC). This channel only transmits zeros and ones, that is binary symbols as the name indicates [22].

To model this channel, we assume that the input and output have discrete-time binary input and output sequences respectively, characterized by the set $X = 0, 1$ for the input and $Y = 0, 1$ for the output. The errors are assumed to be statistically independent, and their average probability is p. The transition probabilities are given as

$$P(Y = 0|X = 1) = P(Y = 1|X = 0) = p, \tag{2.16}$$

and

$$P(Y = 0|X = 0) = P(Y = 1|X = 1) = 1 - p. \tag{2.17}$$

Therefore the channel can be modeled as a discrete-time channel as is shown in figure 2.5.

The BSC is actually a special case of a more general discrete-time chan-

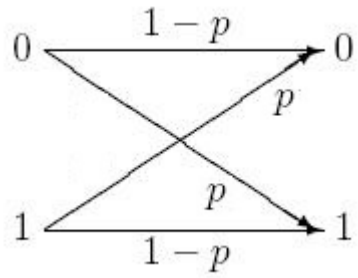

Figure 2.5: Binary symmetric channel model

nel called the discrete memoryless channel. In this case, we have q-ary input symbols going into the channel and Q-ary output symbols coming out from the detector with $Q \geq 2^q$. If we consider that the channel and the modulation are memoryless, then the channel can be described by a set of conditional probabilities

$$P(Y = y_i|X = x_j) \equiv P(y_i|x_j), \tag{2.18}$$

where $i = 0, 1, ..., Q-1$ and $j = 0, 1, ..., q-1$.

Another channel is the discrete-input, continuous-output channel, where the input alphabet is one of q possible values, with $X = x_0, x_1, ..., x_{q-1}$, and the output of the detector is non-quantized, which means that there are infinite values for the output value Y. An example of such a channel is the AWGN channel explained at the beginning of this section, which is modeled as

$$Y = X + N, \tag{2.19}$$

where N is a zero-mean Gaussian random variable with variance σ^2 and $X = x_k, k = 0, 1, ..., q-1$.

A main constraint on any channel model is the capacity of the channel, which will be briefly explained now. We start by considering a discrete memoryless channel with an input alphabet $X = x_0, x_1, ..., x_{q-1}$, an output alphabet $Y = y_0, y_1, ..., y_{Q-1}$ and the set of transition probabilities defined in equation 2.18. Assume that the symbol x_j is transmitted and the symbol y_i is received. The mutual information provided about the event $X = x_j$ by the occurrence of the event $Y = y_i$ is $\log[P(y_i|x_j)/P(y_i)]$, where [22]

$$P(y_i) = \sum_{k=0}^{q-1} P(x_k)P(y_i|x_k). \tag{2.20}$$

The average mutual information would be

$$I(X;Y) = \sum_{j=0}^{q-1} \sum_{i=0}^{Q-1} P(x_j)P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)}. \tag{2.21}$$

The channel capacity is then calculated by computing the maximum of $I(X;Y)$ over the set of input symbol probabilities $P(x_j)$, which depends on the characteristics of the discrete memoryless channel through the conditional probabilities $P(y_i|x_j)$. Therefore, the channel capacity C is

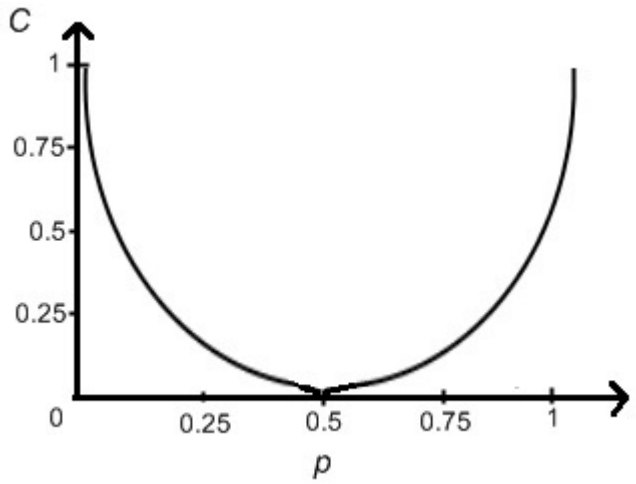

Figure 2.6: BSC capacity

$$C = \max_{P(x_k)} I(X;Y) = \max_{P(x_k)} \sum_{j=0}^{q-1} \sum_{i=0}^{Q-1} P(x_j)P(y_i|x_j) \log \frac{P(y_i|x_j)}{P(y_i)}, k = 0, 1, ..., q-1. \tag{2.22}$$

There exist two constraints for which the maximization is done:

1- $P(x_j) \geq 0$.

2- $\sum_{j=0}^{q-1} P(x_j) = 1$.

The unit of the channel capacity C is measured in bits per input symbol into the channel. Figure 2.6 shows the capacity of a binary symmetric channel plotted versus the probability of error p.

Next we consider the discrete-time AWGN memoryless channel. Again, the capacity is the maximum average mutual information between X and Y.

$$C = \max_{P(x_k)} \sum_{i=0}^{q-1} \int_{-\infty}^{\infty} p(y|x_i)P(x_i) \log \frac{p(y|x_i)}{p(y)} dy, k = 0, 1, ..., q-1 \qquad (2.23)$$

where

$$p(y) = \sum_{k=0}^{q-1} p(y|x_k)P(x_k). \qquad (2.24)$$

2.3.2 Two-State Markov Model

A Markov chain is a stochastic process that studies the transitions between the different allowable states in a certain situation [6]. It is a memoryless random process because the next state in the chain depends does not depend on the previous state other than the current state.

Three elements characterize a Markov chain:

1- The transition diagram (shown in figure 2.7 for a two state).

2- The probability transition matrix $\boldsymbol{P}$.

3- The steady state vector $\boldsymbol{\pi}$ [6].

The two-state Markov model is shown in Figure 2.7. The model has two states in the figure: state 1 or the good state, which means that the packet has been received, and state 2 or the bad state, which means that the packet has been lost. The variable P_{BB} is the self-loop probability for state 2, P_{AA} the self-loop probability for state 1, P_{AB} the probability of transition from state 1 to state 2, and finally P_{BA}, the probability of transition from state 2 to state 1 [6].

For the sake of simplicity, let us relabel the probabilities as follows:

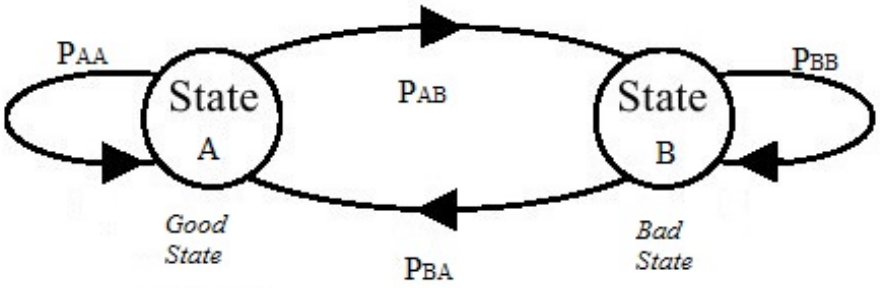

Figure 2.7: 2-state Markov model

P_{BB} is q, P_{BA} is p, P_{AB} is P and P_{AA} is Q.

For states greater than 2, the other states are added with arrows going to and coming from every other state including one self state arrow signifying that the next state remains in the same state as the current state.

The probability transition matrix has as elements the transition probabilities between each of the states and the states themselves. For an nxn matrix, which means there are n-states in the chain, $\boldsymbol{P}$ is expressed as:

$$\boldsymbol{P} = \begin{pmatrix} P_{00} & P_{01} & ... & P_{0n} \\ P_{10} & P_{11} & ... & P_{1n} \\ ... & ... & ... & ... \\ P_{n0} & P_{n1} & ... & P_{nn} \end{pmatrix} \tag{2.25}$$

where p_{ij} is the probability that the current state in the chain is in state i given that the previous state was in state j [6].

The elements of $\boldsymbol{P}$ must satisfy the two following conditions:

1- $p_{ij} \geq 0, i, j = 0, 1, ..., n.$

2- $\sum_j p_{ij} = 1, j = 1, 2, ..., n.$

The first property simply means that each element must be greater than or equal to zero, which is quite logical since those are probabilities. The second property implies that the total sum of the elements of a row must be equal to one.

For the model used in this thesis, the two-state Markov model, the probability transition matrix is as follows:

$$\boldsymbol{P} = \begin{pmatrix} P_{AA} & P_{AB} \\ P_{BA} & P_{BB} \end{pmatrix} \tag{2.26}$$

The steady-state vector $\boldsymbol{\pi}$ represents the total appearing percentage of every state in the chain [6]. We can get this vector from the probability transition matrix $\boldsymbol{P}$ as shown in equation 2.27 below

$$\boldsymbol{P}^m = \mathbf{1}\boldsymbol{\pi} \tag{2.27}$$

where $\mathbf{1}$ is a column vector of ones and m is a large power. The vector $\boldsymbol{\pi}$ must satisfy the property that $\sum_i \pi_i = 1$, where π_i is the steady state probability for the i^{th} state. What this property signifies is that the sum of the elements of $\boldsymbol{\pi}$ should be equal to 1.

There are two parameters of great importance for such a model and they are the average packet loss burst length, β_{2state}, and the probability of packet loss, α_{2state}, both of which are defined as follows:

$$\beta_{2state} = \frac{1}{1-q} \tag{2.28}$$

and

$$\alpha_{2state} = \frac{P}{2 - Q - q}. \tag{2.29}$$

From equations 2.28 and 2.29 it is easy to find the self-loop probabilities q and Q:

$$q = 1 - \frac{1}{\beta_{2state}} \tag{2.30}$$

and

$$Q = 1 - \frac{\alpha_{2state}}{(1 - \alpha_{2state})\beta_{2state}}. \tag{2.31}$$

Markovian models are quite efficient in describing the characteristics of a channel in a mathematical method, allowing us to have a better approach to simulating a real channel [6].

2.3.3 Packet-Switched Networks

In this thesis, packet-switched networks are used since we are transmitting packets of data. However, it is important to explain what is happening during transmission for both packet-switched and circuit-switched networks, and compare the two schemes.

A circuit-switched network establishes a fixed bandwidth channel between nodes before the users are able to communicate or connect. It is analogous to connecting the nodes physically with an electrical circuit [20]. Normally, circuit switching is used only for connecting voice circuits. However, it can be used in other forms of digital data, even if that rarely happens.

In circuit-switched networks, the data is transferred non-stop and without any overhead bits. During a communication between two users, the circuit cannot be used by other users until the circuit is dropped and then a new connection is set up. Such channels are labeled as busy channels. On the other hand, when a channel is available for new calls to be set up, such channels are called idle [20].

To establish a connection through the network and monitor its progress and termination requires the use of a control channel that is separate, similar to the links during telephone exchanges where the CCS7 packet-switched signaling protocol is used to communicate and control the call setup and information as well as use time division multiplexing (TDM) to transmit the actual circuit data. Examples of circuit switched networks include high-speed circuit-switched data service in cellular systems such as X.21 and the public switched telephone network.

Packet switching on the other hand is a communications method where packets are sent on different routes between nodes over data links that are shared with other traffic. In each network node, packets are either queued or buffered, resulting in some variable delay called the queuing or buffering delay respectively. Once a packet is routed using a specific path, it is highly possible that the path may change for the next packet and not be the same. This means that in some cases the packets sent from the same source to the same destination would end up being routed differently. Hence the need to know the original order of the packets [20].

Packet switching is used for several advantageous reasons. It optimizes the use of the channel capacity available in digital communication networks

such as computer networks. It also minimizes the time wasted in circuit switching to establish a connection for transmission, and increases robustness of communication. The Internet and local area networks (LAN) are the most well-known applications that make use of packet switching. [20].

To conclude this section, a small comparison is made between circuit and packet switching. The main points for circuit switching are:

1- Long call setup times

2- The setup times can be negligible compared to data length

3- Inefficient channel utilization for bursty traffic.

As for packet switching, its main points are:

1- Breaks the message into packets

2- Can interpolate packets from other nodes, therefore does not block the system

3- Error check on

4- Efficiently handles asymmetric traffic.

2.4 Channel Coding

This section covers the topic of channel coding. Channel coding is the most important building block of any transmission scheme. It is used to protect the transmitted data from errors or erasures that are introduced by the channel in which they are being transmitted, so that they can be correctly decoded at the receiver. Source coding decreases the amount of bits being sent by removing redundant bits whereas channel coding adds bits to the transmitted data to protect it. There are two types of channel codes: the

block codes and the convolutional codes.

2.4.1 Block vs Convolutional Codes

Let us start by defining the two types of codes, and then proceed to give an example of one of the most powerful convolutional codes, the Viterbi code as well as an example of a simple block code, the repetition code. Linear convolutional codes produce an output that depends on the previously received bits and the current bit by performing an operation such as binary XOR on those bits for example. The block codes on the other hand, simply take a k block of bits and converts them into an n block of data. Both codes have a rate $R = k/n$, where k is the input number of bits and n is the output number of bits after encoding. Linear block codes have two advantages over linear convolutional blocks [10]:

1- The processing delay caused by their implementation is less than their counterparts.

2- The computational and processing complexity of the algorithms is less.

The Viterbi algorithm is based on the maximum likelihood decoding technique. Figure 2.8 below shows an example of a convolutional encoder that takes k bits as input and outputs n bits, where $n > k$ [10].

The encoder shown in figure 2.8 has as parameters $k = 1$ and $n = 2$, and hence the rate of this encoder is 1/2. The first output bit is computed by applying the XOR operation on x(n), x(n-1) and x(n-2), whereas the second output bit is computed by applying an XOR on x(n) and x(n-2). One important thing to note is that the convolutional encoder is not systematic,

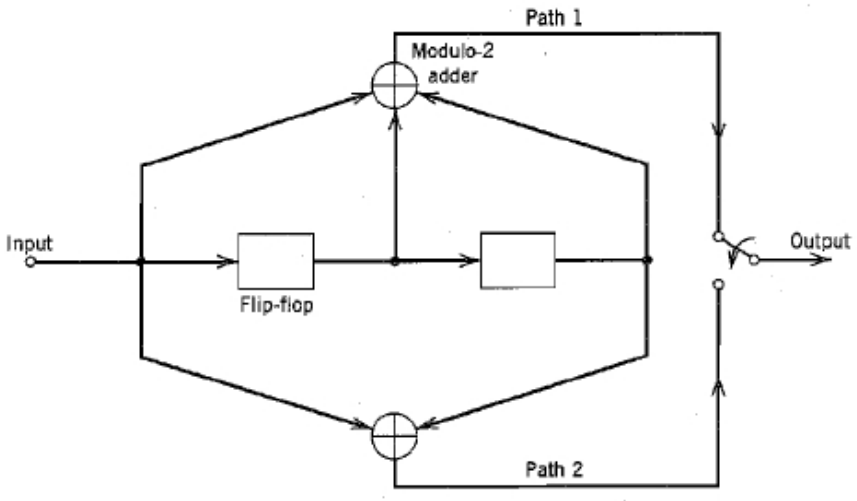

Figure 2.8: Convolutional encoder [10]

which means that the output bits do not contain any of the input bits. That can be easily noticed by observing the output. For systematic codes, the n output bits are made of the k input bits followed by the $n - k$ added bits for protection, which is obviously not the case with the Viterbi encoder.

To obtain the trellis diagram which is needed for the decoding of the received sequence, one needs to apply the Viterbi algorithm. Here is a summarized step-by-step implementation of the algorithm [10]:

A- Initialization

Label the left-most state of the trellis (i.e., the all-zero state at level 0) as 0.

B- Computation step j + 1

Let j = 0,1,2,... and suppose that at the previous step j, we have done two things: all survivor paths have been identified, and the survivor path

and its metric for each state of the trellis have been stored.

Then at level $j+1$, compute the metric for all the paths entering each state of the trellis by adding the metric of the incoming branches to the metric of the connecting survivor path from level j. Hence, for each state, identify the path with the lowest metric as the survivor of step $j+1$, thereby updating the computation.

C- Final Step

Continue the computation until the algorithm finishes its forward search through the trellis and therefore reaches the termination node (i.e., the all-zero state), at which time it makes a decision on the maximum likelihood path. Then, the sequence of symbols associated with that path is released to the destination as the decoded version of the received sequence.

One important note is that if the received sequence is very long, then the Viterbi algorithm will require a lot of storage capabilities, and in such cases, normally a compromise is made. That compromise is to truncate the trellis as follows:

We choose a decoding window of length l. The algorithm operates on a corresponding frame of the received sequence and it always stops after l steps. After that, a decision has to be made with regards to the "best" path, and the symbol associated with the first branch on that path is released to the user, whereas the one associated with the last branch on the path is dropped.

The next step is to move forward the decoding window by one time interval, and make a decision on the next code frame, and so on. It is true that in such a case, the decoding decisions made are not exactly maximum

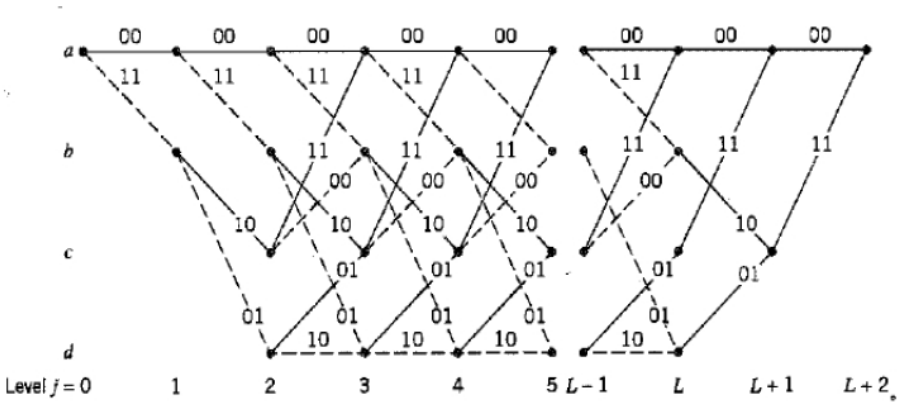

Figure 2.9: Viterbi decoder [10]

likelihood, but they can be made almost as good provided that the decoding window l is long enough [10].

It has been shown that satisfactory results can be obtained as long as the decoding window length l is about five times the constraint length K of the convolutional code, where K is equal to the number of shifting elements found in the encoder, plus 1. In the case of figure 2.8, $K = 3$. After applying the algorithm, we get the trellis diagram shown in figure 2.9.

A quick example will illustrate how a trellis diagram is built. Assume that the encoder in figure 2.8 produces an all-zero output, and after transmission, the received sequence was 0100010000.... Therefore, there are two bits that are received in error (the two bits that are 1). Figure 2.10 gives a step-by-step illustration of how the correct sequence is decoded.

Next up, we discuss block codes, and we start with the repetition code, which is one of the simplest and easiest block codes to implement. The concept is to take every codeword and repeat it based on the rate of the code. For example, a $k = 2$ and $n = 6$ code has a rate of 1/3. This

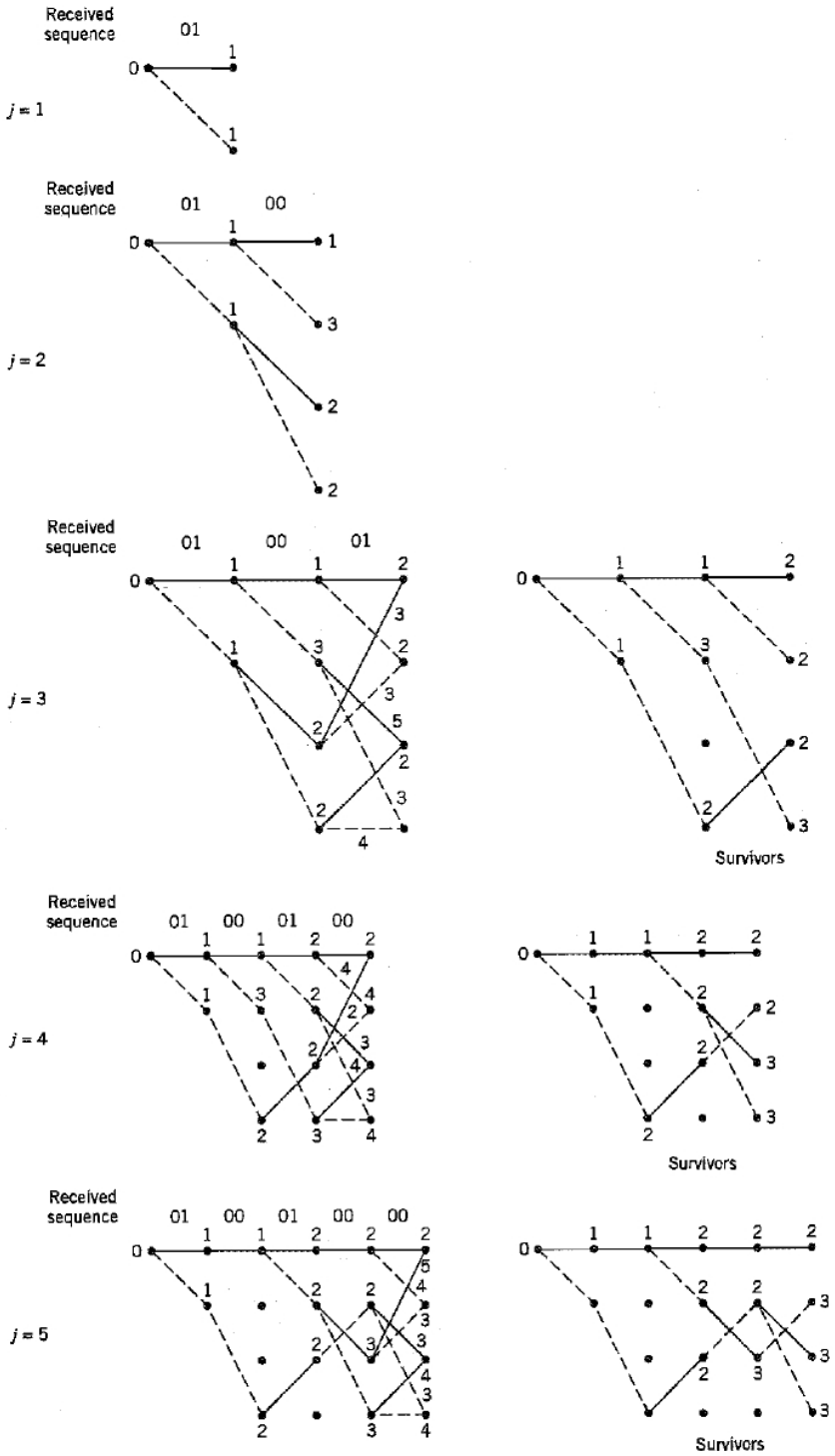

Figure 2.10: Step by step Viterbi decoding [10]

means that every original code-word has to be repeated three times. Let the original symbols be designated as $k_i, i = 1, 2, 3, 4$ and the codewords as $n_j, j = 1, 2, 3, 4$. An example of a 1/3 rate repetition code is

k_0 = [0 0], n_0 = [0 0 0 0 0 0],

k_1 = [0 1], n_1 = [0 1 0 1 0 1],

k_2 = [1 1], n_2 = [1 1 1 1 1 1],

k_3 = [1 0], n_3 = [1 0 1 0 1 0],

The decoding is done by comparing the received code-word to the dictionary containing all possible code-words. The comparison is done in terms of the differences in the bits. The codeword that has the least total difference is then chosen as the corrected received code-word [10].

2.4.2 Reed-Solomon Code

Now we move on to a more complicated block code which is the Reed-Solomon code. The RS code is a linear systematic block code based on finite field theory. It is an (n, k) code, where k is the number of data symbols, and n is the total number of symbols transmitted which contains both the data symbols and the Forward Error Correcting (FEC) symbols, as shown in Figure 2.11.

The parameter n is defined as,

$$n = q^m - 1, \tag{2.32}$$

where q is the size of the Galois Field (GF), and m is the number of bits in every symbol. Basically a GF is a finite field on which the operations of

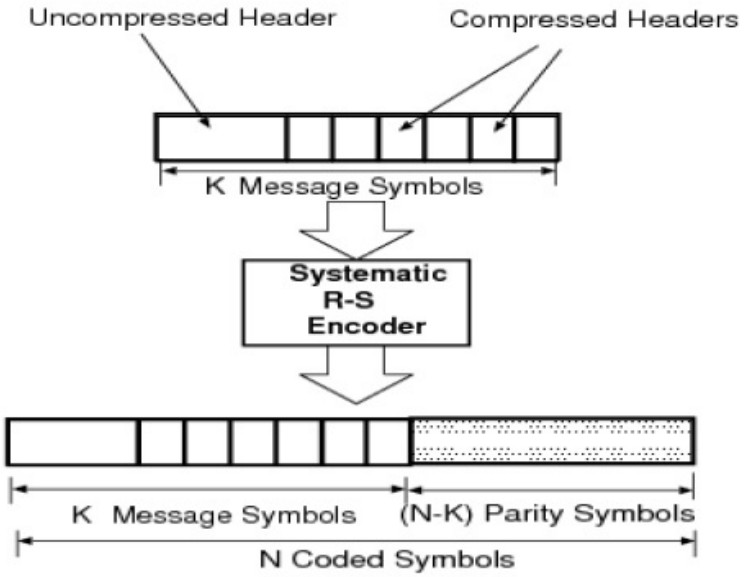

Figure 2.11: Reed-Solomon encoder [10]

commutative addition, subtraction, multiplication and division (except by zero) are defined [19].

The RS code has the ability to correct t errors, as long as the following inequality is respected:

$$t \leq \frac{(n-k)}{2}. \tag{2.33}$$

Next, let us delve into the mathematics behind the encoding of the RS code [25], which is defined by its generator polynomial. For a code capable of correcting up to t errors, the generator polynomial is given as

$$g(X) = (X+a^{m_0})(X+a^{m_0+1})(X+a^{m_0+2})...(X+a^{m_0+2t-1}) = g_0+g_1X+g_2X^2+...+g_{2t-1}X^{2t-1}+X^{2t} \tag{2.34}$$

where a, an m-bit binary symbol, is the primitive element of the finite field $GF(2m)$, and m_0 is a pre-set number that is usually 0 or 1. To encode a message sequence, the message polynomial is first constructed as

$$u(X) = u_0 + u_1X + u_2X^2 + ... + u_{k-1}X^{k-1}. \tag{2.35}$$

The parity polynomial, which has the parity sequence as its coefficients, is calculated as the remainder of X^{2t}. The code is constructed as the data sequence followed by the parity sequence. Hence, the final code polynomial is

$$t(X) = X^{2t}u(X) + v(X) \tag{2.36}$$

where

$$v(X) = v_0 + v_1X + v_2X^2 + ... + v_{2t-1}X^{2t-1}. \tag{2.37}$$

The decoding of the data takes place as follows. We assume that the transmitted code vector is

$$t(X) = t_0 + t_1X + t_2X^2 + ... + t_{n-1}X^{n-1} \tag{2.38}$$

and that the received vector is

$$r(X) = r_0 + r_1X + r_2X^2 + ... + r_{n-1}X^{n-1}. \tag{2.39}$$

The first step is to calculate the $2t$ syndrome components as follows:

$S_0 = r(a^0) = r_0 + r_1 + r_2 + ... + r_{n-1}$

$S_1 = r(a^1) = r_0 + r_1(a) + r_2(a)^2 + ... + r_{n-1}(a)^{n-1}$

$S_2 = r(a^2) = r_0 + r_1(a^2) + r_2(a^2)^2 + ... + r_{n-1}(a^2)^{n-1}$

up to

$S_{2t-1} = r(a^{2t-1}) = r_0 + r_1(a^{2t-1}) + r_2(a^{2t-1})^2 + ... + r_{n-1}(a^{2t-1})^{n-1}$.

The syndrome polynomial is then calculated as

$$S(X) = S_0 + S_1X + S_2X^2 + ... + S_{2t-1}X^{2t-1}. \tag{2.40}$$

The second step in the decoding process of an RS code is to find the error location polynomial $L(X)$ and the error evaluation polynomial $W(X)$. The error location and error evaluation polynomials are defined as

$$L(X) = 1 + L_1X + L_2X^2 + ... + L_eX^e \tag{2.41}$$

and

$$W(X) = W_0 + W_1X + W_2X^2 + ... + W_{e-1}X^{e-1} \tag{2.42}$$

respectively, where e is the number of errors. These two polynomials are related to the syndrome polynomial through the fundamental equation

$$L(X)S(X) = W(X) mod X^{2t}. \tag{2.43}$$

Both $L(X)$ and $W(X)$ are solved using the iterative Berlekamp-Massey algorithm.

The last step in decoding an RS code is to find the error location and the error value. One can find the error location by using Chan's searching algorithm. Basically X is substituted with a^n in $L(X)$ for all possible n in a code to find the root of $L(X)$. The inverse of the root of the error location polynomial is the error position. Once the error location is known, the error value is calculated with the help of Forney's error evaluation algorithm. Once the error value is found, it is added to the corrupted symbol to correct the error [24].

An interesting property of the RS code is its ability to correct not only errors but also erasures, as long as the following inequality is respected:

$$2t + S \leq n - k, \tag{2.44}$$

where S is the number of erasures that can be corrected. In such a case, the error locator polynomial is modified such that it would also include an erasure locator polynomial.

There are many applications for the RS code, most notably in data storage applications such as the CD and DVD, and in data transmission. It is also used in mail encoding and most notably, in satellite communications. The first major use of RS codes for satellite transmission was when Voyager had to transmit a digital picture back to Earth and it used the RS code in conjunction with the Viterbi algorithm, and since then this practice has become a fixture in deep-space and satellite transmission [24].

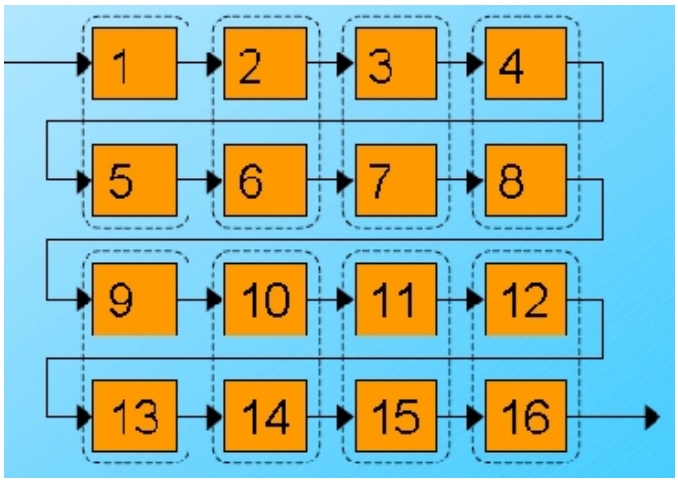

Figure 2.12: Interleaver [18]

2.4.3 Interleaving

An interleaver is a process that rearranges the order of transmitted symbols before sending them and then returns them to their original location at the receiver's end.

The main reason behind the use of an interleaver (and at the receiver a deinterleaver) is to be able to deal with channels that cause errors or erasures in bursts. The concept is very simple: just input the sequence to be transmitted horizontally and then read it vertically, as is shown in Figures 2.12 and 2.13 respectively. What this helps in is spreading out the errors caused by bursts which helps in making the decoding process in most scenarios better.

For transmission schemes that use block coded channel encoders, the error detection and correction capability can be limited as seen in equation 2.33. In case of a very poor channel, the block decoders would fail to correct

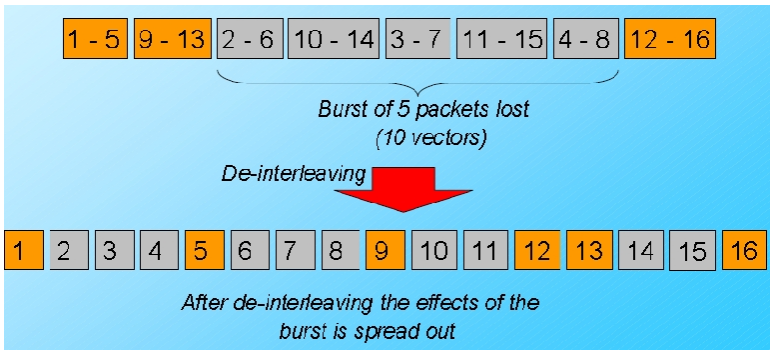

Figure 2.13: Deinterleaved output [18]

all the errors introduced by the channel, but by spreading the errors on a larger set of symbols, the interleaving/deinterleaving process in fact gives the decoders a better chance at being able to properly decode the received data.

The interleaver in Figure 2.12 is a square interleaver having a depth of four and is called a 4-by-4 interleaver. The depth of the interleaver is the number of symbols in each block of data. Of course, the interleaver can have any dimension the designer wishes to use, but it is always advisable to use one where the distance between the two consecutive output symbols is large enough, so that when the error burst occurs, the errors can be scattered as far away as possible from each other.

Chapter 3

Frameworks

This chapter discusses the different schemes applied and parameters used throughout the work.

3.1 General Framework of the CS-EEG Encoding

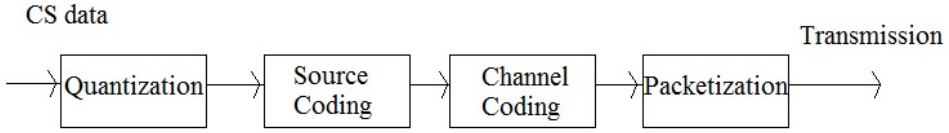

Figure 3.1: Block diagram of the encoder

After collecting the data and applying compressed sensing (CS) to it using the improved block-sparse Bayesian learning (BSBL) algorithm, the encoder stage was designed as shown in figure 3.1.

The first step in designing our transmission system was to choose the number of quantization bits from which we can know how many quantization levels will be used. The equation that relates the two quantities is given as

$$Q_{lvl} = 2^{q_{bits}} \tag{3.1}$$

where Q_lvl is the number of quantization levels available and q_bits is

the number of bits required to represent the quantized values. To decide on the number of quantization bits, we need to make sure that no valuable information is lost when quantization is done. This is because quantization introduces an error called the quantization error which should be kept at a minimum. Keeping that in mind, we quantized the signals using different number of quantization bits. We sent the data through an ideal channel. We did not apply any source or channel coding on this data in order to see the effect of this parameter on the reconstructed signals. Then we reconstructed the quantized data and compared it to the original signals available.

There are two types of quantization:

1- Uniform quantization: The quantization levels have the same interval distance between each other. Each interval is also represented by the same number of quantization bits. This type of quantization is used when the signal values are uniformly distributed among all values.

2- Non-uniform quantization: The signal information is distributed heavily around a certain range. In those areas, more levels are assigned, whereas the other areas would get lesser levels assigned to them.

Since the compressed sensing (CS) EEG signals contain equally important information in all its levels, we chose uniform quantization for our work.

In the first two sections, 3.2 and 3.3, we used the lossless Huffman code for source coding. The reasoning behind our choice is that Huffman is capable of perfectly reconstructing a signal under ideal channel conditions. Since the EEG data has values that occur more than others, applying the Huffman code to compress it is a reasonable choice. This is because Huffman coding relies on the probability of occurrence of symbols to compress the data.

Huffman is used in many applications such as the image formats joint photographic experts group (JPEG) and portable network graphics (PNG). It is also used in the archive file format PKZIP (it includes the zip archiving file format).

The Reed-Solomon (RS) encoder was used in all schemes. The RS is a block code. Block codes are known for their fast implementation compared to convolutional codes, which is highly desired for wireless body network sensors (WBSN)s. It is also capable of detecting and correcting errors. That capability can be increased by changing the data length of each codeword (remember that for block codes such as the RS, k-data symbols are encoded into n-symbols, where $n > k$).

In figure 3.1, the data was sent as packets. Each packet length is 12 symbols, where each symbol is a double variable in Matlab.

The channel was simulated using a two-state Markov model [12]. The data was transmitted through the Markov model under three noisy channel conditions:

1- Good channel conditions, which mean that the number of errors introduced by the channel is on average quite low.

2- Average channel conditions, which mean that the number of errors increases. This is where the channel encoder and decoder error correcting capabilities start to be tested.

3- Poor channel conditions, which mean that the number of errors introduced by the channel is large.

The channel error sequences were generated in two different ways using the above matrices for each channel condition:

1) Random pattern: In this approach, we generate the error sequence at the same time as the data is being transmitted. What this means is that the data transmitted will not always be sent under the same error sequences. Every transmission process will be done under a different error sequence. For fair results, the same data is transmitted 10 times over 10 different error sequences. Then the results obtained for each of the 10 cases will be averaged. This way of sending the data helps to find if the algorithm that outperforms all others once, will in general outperform them on a regular basis.

2) Fixed pattern: This approach generates one fixed error sequence for the three channel conditions (good, average, poor). These 3 sequences are used for all the data to see how the algorithms would compare under the exact same channel conditions.

We expect that the algorithm that will outperform the other algorithms using the first method will also outperform them using the second method. The error sequence consists of two values: 1 and 2. The value 1 implies that the packet has been correctly received. The value 2 implies that the packet has been incorrectly received. In the case of the latter situation, the whole packet is in error. This means that the symbols found in the packet have all had their values changed.

An interleaver and deinterleaver were used after the RS encoder and before the RS decoder. The reasoning behind the use of the interleaver is simple. The error sequences can occur in large bursts, especially in poor channel conditions. In such a case, the RS decoder might not be able to correct all the errors introduced by the channel. This would degrade the

performance. The interleaver would spread out the errors. This would allow the RS decoder to be able to correct the erroneous data.

The following sections will give the block diagrams of the schemes applied to study the individual and combined effects of source coding (compression), forward error correction and interleaving on the performance of the encoder side in terms of processing times and total number of symbols sent. It will also evaluate their performances at the decoder side in terms of data reconstruction.

3.2 Huffman Code Followed by RS Code

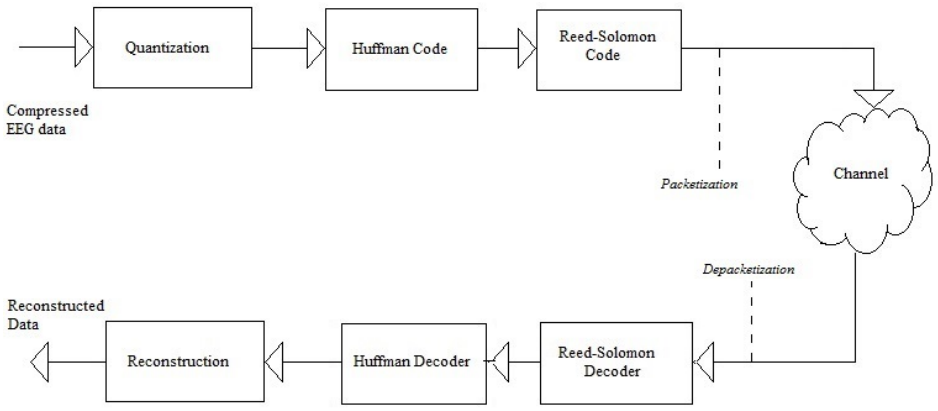

Figure 3.2: Block diagram of the first approach

The first scheme that we considered was the pairing of the Huffman code followed by a Reed-Solomon code. At the receiver side, the decoder of each algorithm is applied in reverse order from the encoder side, i.e. the RS decoder first and then the Huffman decoder. Finally, the signal is reconstructed. The block diagram for this scheme is shown in figure 3.2.

3.3 Huffman Code Followed by RS Code and Interleaving

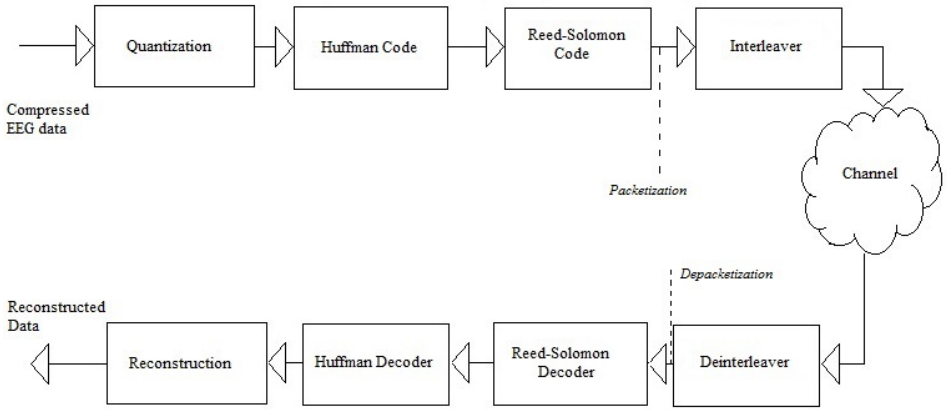

Figure 3.3: Block diagram of the first approach with interleaving

Figure 3.3 shows the block diagram of the approach used in this section. Basically it is the approach used in section 3.2 with interleaving and deinterleaving added before and after the channel respectively. We expect the results to improve with the addition of interleaving.

3.4 No Source Coding Followed by RS Code

The CS algorithm used achieved a compression ratio of 90 %. Since the addition of Huffman proved to under-perform in poor channel conditions as will be shown in the next chapter, in this framework we use no further compression than that provided by CS. Therefore we remove the Huffman code. Instead, the quantized data is sent directly into the RS encoder, as shown in figure 3.4.

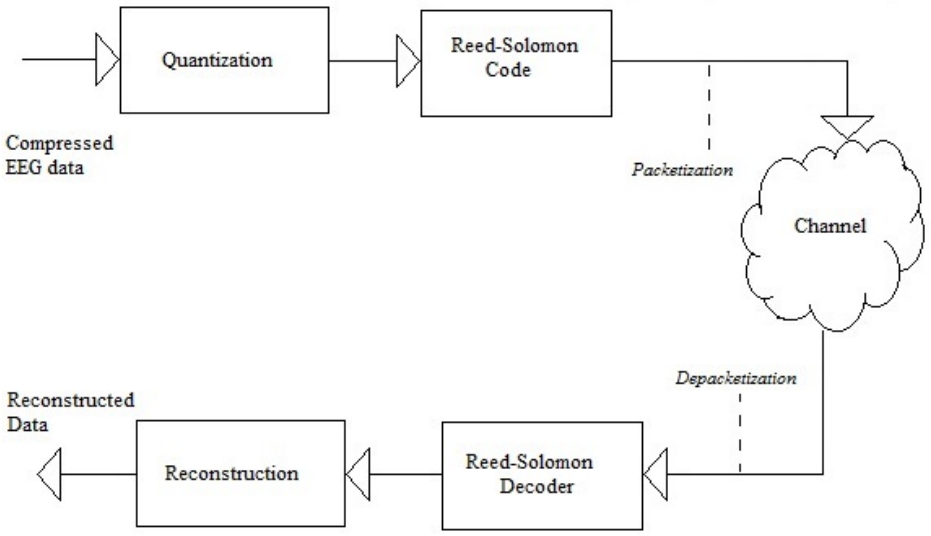

Figure 3.4: Block diagram of the second approach

3.5 No Source Coding Followed by RS Code and Interleaving

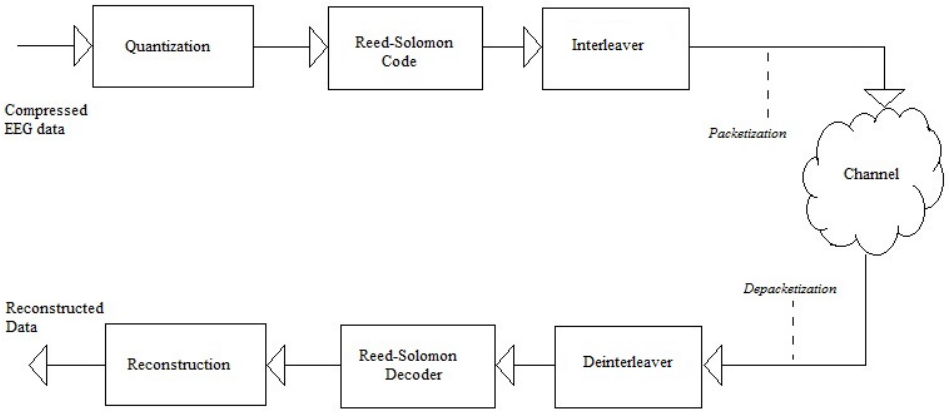

Figure 3.5: Block diagram of the second approach with interleaving

This last framework is a slight modification of the previous one in that interleaving and deinterleaving is added after and before the RS encoder

and decoder respectively, as is shown in figure 3.5. Again we expect an improvement to the previous scheme due to the presence of interleaving.

3.6 Performance Measures

The measures used to test the performance of the work done are:

1- Normalized Mean Square Error (NMSE): this parameter calculates the errors or differences between the original signal and the reconstructed signal. It is expressed as $\frac{\|(x-\hat{x})\|_2^2}{\|x\|_2^2}$, where x is the original signal and $\hat{x}$ is the reconstructed signal. The lower and closer to zero the NMSE, the better the performance of the scheme.

2- Signal to Noise Ratio (SNR): a way to see how much the noise or errors have distorted the signal. It is defined as $SNR = 10\log(\frac{P_{signal}}{P_{noise}})$. For our work, the signal is the CS output signal before quantization, and the noise is the recovered signal after reconstruction. The higher the value of this parameter, the better. The unit for this parameter is decibel (dB).

3- Structural SIMilarity (SSIM): a parameter that measures the similarity between the original and reconstructed signals. Its value is between 0 and 1. The closer it is to 1, the closer the reconstructed signal is to the original one [30].

Chapter 4

Results

4.1 Parameters Used in the Framework

This chapter discusses the results obtained from the different schemes applied. We start by presenting the parameters used for the improved compressed sensing (CS) algorithm. The signals that we are encoding are electroencephalogram signals. They are collected from 23 channels. The data was compressed up to a 90 % compression ratio (CR) with the help of CS. The dictionary used is the inverse discrete cosine transform (DCT^{-1}) dictionary. The reconstructing algorithm used is the BSBL-BO algorithm explained in [15].

Table 4.1: Results for Different Values of q_{bits}

q_{bits}	**NMSE**	**SNR(in dB)**	**SSIM**
8	$0.03526 * 10^{-3}$	38.6601	0.9756
7	$0.06316 * 10^{-3}$	32.8572	0.9611
6	$0.25824 * 10^{-3}$	26.7608	0.9224
5	$1 * 10^{-3}$	20.7383	0.841

Table 4.1 gives the NMSE, SNR and SSIM for different values of the number of quantization bits q_{bits}. We started with $q_{bits} = 8$ bits and the

results were excellent. We then decreased the number of quantization bits to 7 bits to see how much the parameters would be affected. Logically, we expect the values to be slightly worse, which was the case. But even then, the results were still excellent. We then resent the data for $q_{bits} = 6$ bits and the results were still great. We tried to decrease the value of q_{bits} to only 5 bits, but as can be seen in table 4.1, the results were not satisfactory. Usually, if SSIM = 0.841, the reconstructed signal still resembles the original signal a lot. But we can not afford such loss in signal similarity because we need to account for the losses that the channel errors will introduce to the data. This would further lower the value of SSIM and hence the lowest value for q_{bits} that we can afford to achieve good reconstruction is 6 bits. This means that there are $2^6 = 64$ quantization levels.

The RS code used had a codeword length of $n = 255$ symbols. Three different data lengths k were used: 223, 193, 153. By applying equation 2.33, we can calculate up to how many errors each code is capable of correcting. The RS(255,223) code can correct up to 16 errors. The RS(255,193) code can correct up to 31 errors. And the RS(255,153) code can correct up to 51 errors. It becomes directly clear that the RS(255,153) would yield better results than the other two codes in severe channel conditions because of its ability to correct more errors, but the trade-off is that we would be transmitting more bits.

The parameters for the 2-state Markov model were calculated from the values of the probability of packet loss, α, and the average packet burst length, β. The values of both α and β for each of the 3 noisy channels were obtained from previous statistics on global system for mobile (GSM) com-

munications (which is a standard used by mobile phones in digital cellular networks). The values for α and β are given in table 4.2.

Table 4.2: Values for α and β

Channel Condition	α	β
Good	0.001167	1.076
Average	0.02825	1.378
Poor	0.12372	1.429

From the values of α and β, we calculated the values of p_{ij} of the probability transition matrix using equations 2.30 and 2.31. These equations allow us to calculate the values of Q and q, which are the self loop probabilities of the good and bad states in the Markov model respectively. From Q and q we calculate $P = 1 - Q$ and $p = 1 - q$. The transition probability matrices for each of the 3 channels are then as follows:

$$\boldsymbol{P_{good}} = \begin{pmatrix} 0.9989 & 0.0011 \\ 0.9294 & 0.0706 \end{pmatrix}$$

$$\boldsymbol{P_{average}} = \begin{pmatrix} 0.9789 & 0.0211 \\ 0.7257 & 0.2743 \end{pmatrix}$$

$$\boldsymbol{P_{poor}} = \begin{pmatrix} 0.9012 & 0.0988 \\ 0.6998 & 0.3002 \end{pmatrix}.$$

The interleaver used in the two sections 3.3 and 3.5 is a

$$\frac{L_{epoch}}{12} \text{ by } 12$$

interleaver, where L is the length of the encoded epoch (i.e. after source and channel encoding). This means that the encoder depth is 12, which in

turn means that the data is rearranged in a way that every 12th symbol in the epoch is placed one after the other. Therefore, if we have 24 symbols

$ABCDEFGHIJKL\ MNOPQRSTUVWX$,

this interleaver would rearrange them as

$AMBNCODPEQFRGSHTIUJVKWLX$.

4.2 Huffman Code Followed by RS Code

Tables 4.3, 4.4 and 4.5 show the results obtained when the data was transmitted through the randomly generated error sequences using the three different RS codes, the (255,223), (255,193) and (255,153) codes respectively.

Tables 4.6, 4.7 and 4.8 show the results of the three RS codes using the pre-generated error sequences for each of the three channel conditions (good, average, poor).

Naturally as expected, the case when RS(255,153) was used achieved the best results. However, for poor channel conditions, the SSIM of the reconstructed signal was low, which means that this scheme is not suited for bad channel conditions.

Table 4.3: Results for Random Patterns Using RS(255,223)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$29.6 * 10^{-3}$	21.4714	0.8127
Poor	FAILED	FAILED	FAILED

Overall, the only combination in this section that fared well was the

Huffman code followed by the RS(255,153) code, but even that pairing gave results that are not satisfactory under poor channel conditions. Figures 4.1, 4.2 and 4.3 show the reconstructed signal in green superimposed on the original signal in blue for the cases when RS(255,223) was used in both good and average conditions, and when the RS(255,193) was used in bad conditions respectively.

Notice that for the RS(255,223) code, the scheme completely fails under poor channel conditions using the randomly generated error sequence approach, and completely under all channel conditions when using the fixed generated error sequence. The RS(255,193) fails as well under poor channel conditions when using the fixed generated error sequence.

The reason for such failures is because the number of errors introduced by the channel in a certain RS codeword is larger than what can be corrected by the RS decoder. In this case of the RS(255,223) and RS(255,193) codes, the number of errors that can be corrected is 16 and 32 errors respectively. This changes the symbol stream and hence introduces a new codeword that is not found in the dictionary of the Huffman code. In such cases, Huffman can no longer decode the codewords and the whole scheme fails.

In the cases where the scheme did not fail, but results achieved were less than satisfactory, there exists another reason. If the error introduced is larger than what the RS decoder can correct, it can happen that the errors are still a codeword in the Huffman dictionary. But this codeword is not the one that was actually transmitted. This still allows the Huffman code to function properly, however a propagated error would occur starting with the first codeword that was wrongly decoded. To make this clearer, assume

that the transmitted codeword was

AAABCDADBBD

.

If the error occurs at the 4th symbol, i.e. symbol B, there is no certainty that the error will not affect the rest of that sequence. The Huffman decoder would still be able to decode the sequence because even though the symbol sequence is erroneous, the errors caused symbols to appear in the sequence that are still in the Huffman dictionary. Therefore, the decoded sequence can possibly be

AAACABDDCBA

which obviously is not the transmitted sequence itself. This would degrade the reconstructed signal and hence why some numbers such as those in table 4.4 for average and poor channel conditions are not better.

Table 4.4: Results for Random Patterns Using RS(255,193)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$19.8 * 10^{-3}$	23.9993	0.852
Poor	$211.8 * 10^{-3}$	5.2918	0.4068

Figure 4.1 shows an almost perfect reconstruction of the signal using the RS(255,223) code. The small differences between the original and reconstructed signals are attributed to the quantization error introduced at the encoder. Figure 4.2 shows that the algorithm performed close to the optimal

Table 4.5: Results for Random Patterns Using RS(255,153)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$2.3 * 10^{-3}$	25.4091	0.9263
Poor	$142.4 * 10^{-3}$	6.9801	0.5438

Table 4.6: Results for Fixed Patterns Using RS(255,223)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	FAILED	FAILED	FAILED

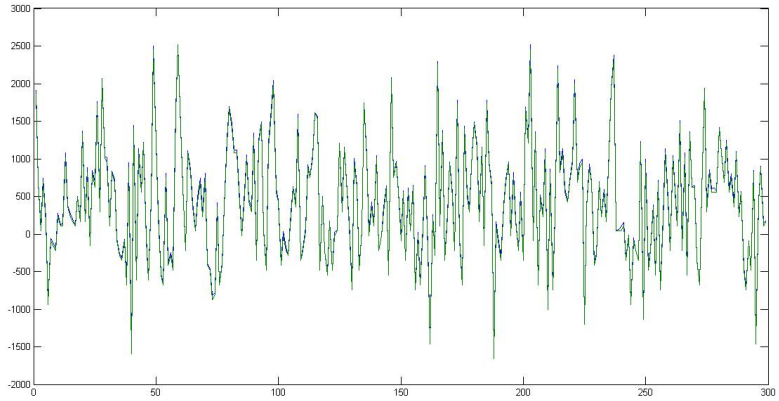

Figure 4.1: Huffman followed by RS code (255,223) in good channel conditions

case (which is the results obtained in table 4.3 for the good channel case). Figure 4.3 shows that some samples are not correctly reconstructed in the output signal. It also shows a small shift in the reconstructed samples in some parts of the signal. This explains why the SSIM was only 0.4068.

Table 4.7: Results for Fixed Patterns Using RS(255,193)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	FAILED	FAILED	FAILED

Table 4.8: Results for Fixed Patterns Using RS(255,153)

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$4.7 * 10^{-3}$	25.4211	0.8796

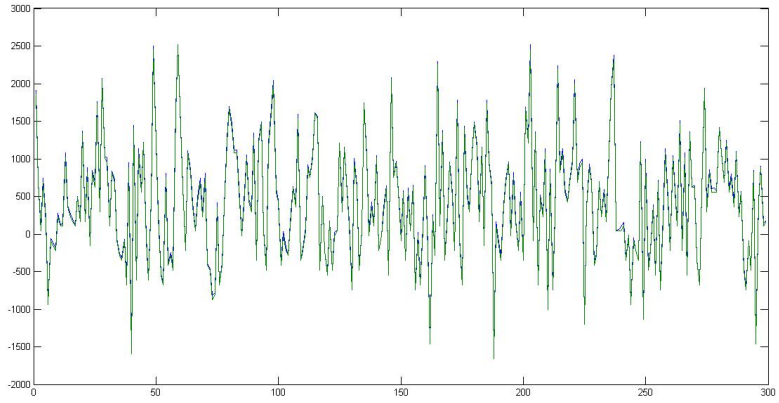

Figure 4.2: Huffman followed by RS code (255,223) in average channel conditions

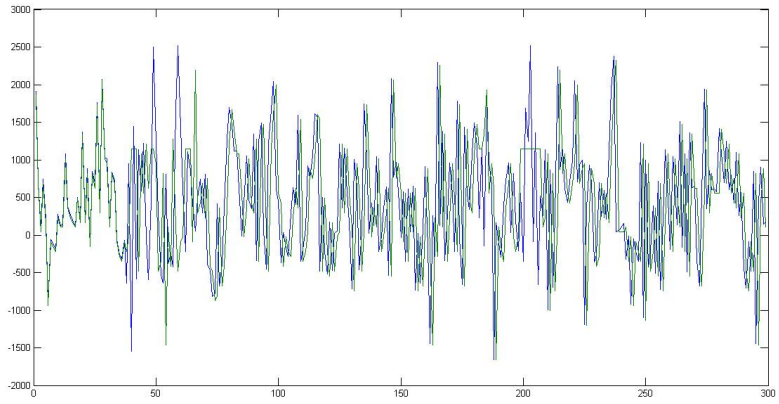

Figure 4.3: Huffman followed by RS code (255,193) in bad channel conditions

4.3 Huffman Code Followed by RS Code and Interleaving

Tables 4.9, 4.10 and 4.11 again show the results obtained when the data was sent through randomly generated channels using the three different RS codes, and tables 4.12, 4.13 and 4.14 show the results when a fixed error stream is used.

The results show that once again the RS(255,223) fails completely under bad channel conditions for randomly generated error sequences as well as for all channel conditions when using the pre-generated fixed error sequences. So does the RS(255,193) under poor channel conditions when using the pre-generated error sequence scenario.

However apart from that, the results of the algorithm fared better when compared to the previous section. This is attributed to the interleaver which spreads out the errors to other parts of the data stream. Sometimes the er-

Table 4.9: Results for Random Patterns Using RS(255,223) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$19.18 * 10^{-3}$	23.8635	0.8475
Poor	FAILED	FAILED	FAILED

rors happen in large bursts which would render the channel decoders helpless and unable to correctly decode the data. The spreading of those errors would make it easier and more possible for the decoders to retrieve the correct data.

Table 4.10: Results for Random Patterns Using RS(255,193) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$184.1 * 10^{-3}$	6.1303	0.4435

Table 4.11: Results for Random Patterns Using RS(255,153) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$71.3 * 10^{-3}$	19.506	0.7494

Figure 4.4 shows the improvement that interleaving achieves by superimposing the green reconstructed signal over the blue original one for the RS(255,223) code under average channel conditions using the random error

Table 4.12: Results for Fixed Patterns Using RS(255,223) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	FAILED	FAILED	FAILED

Table 4.13: Results for Fixed Patterns Using RS(255,193) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	FAILED	FAILED	FAILED

streams.

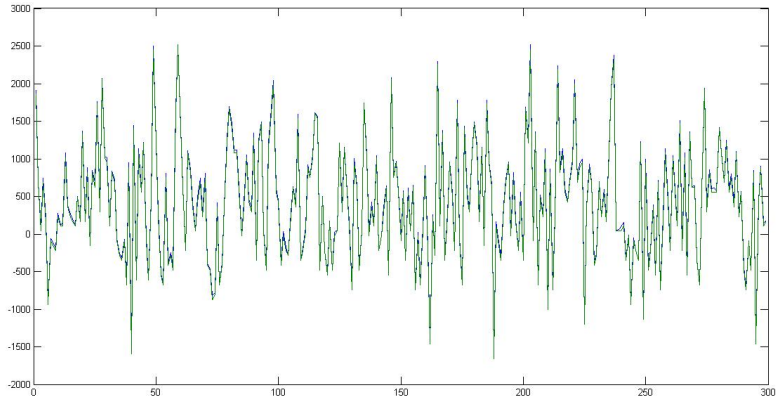

Figure 4.4: Huffman followed by RS code (255,223) with interleaving in average channel conditions

Figure 4.5 also shows the improvement in the case of the RS(255,193) under poor channel conditions when compared to the exact scenario without interleaving (in section 4.2). Those improvements though, just as the tables

Table 4.14: Results for Fixed Patterns Using RS(255,153) and Interleaving

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$0.25824 * 10^{-3}$	26.7608	0.9224

show as well, are not good enough for this scheme to be adopted for the transmission of CS-EEG signals.

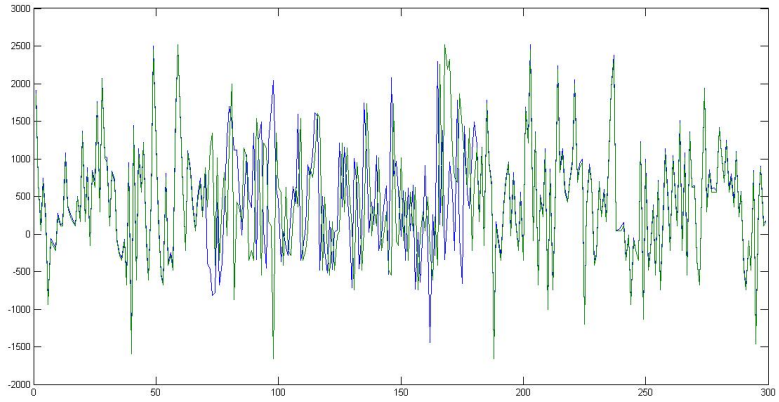

Figure 4.5: Huffman followed by RS code (255,193) with interleaving in bad channel conditions

4.4 No Source Coding Followed by RS Code

The fact that the conventional approach of using a source coder followed by a channel coder failed in its performance made us think of ditching any further source coding, since already CS acts as a source coder by compressing 90 % of the data. This is what this section implements: CS followed directly by

the RS encoder.

Table 4.15: Results for Random Patterns Using RS(255,223) with no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$10.1 * 10^{-3}$	20.9918	0.8946
Poor	$74.1 * 10^{-3}$	5.5298	0.5881

The next 5 tables (table 4.15 - 4.19) show the results obtained from this simulation. The first thing that we notice is that the algorithm never fails regardless of the channel conditions. This is because there is no pre-determined dictionary used anywhere that gives only a specific number of decodable codewords. The fact that it does not stop working under any circumstance already favors it over the two schemes presented in sections 3.2 and 3.3.

Table 4.16: Results for Random Patterns Using RS(255,193) with no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$69.3 * 10^{-3}$	9.4839	0.7423

The next noticeable improvement in the results is that this algorithm outperforms the previous two in every aspect, under every condition, with the exception of the total number of bits sent per epoch (these numbers will be compared in section 4.6). Figure 4.6 further proves the point that it

outperforms the previous two scenarios, for the RS(255,193) code using the randomly generated error sequences under poor conditions.

Table 4.17: Results for Random Patterns Using RS(255,153) with no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$0.25824 * 10^{-3}$	26.7608	0.9224

Table 4.18: Results for Fixed Patterns Using RS(255,223) with no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$33.1 * 10^{-3}$	5.6462	0.4801

Table 4.19: Results for Fixed Patterns Using RS(255,193/153) with no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$0.25824 * 10^{-3}$	26.7608	0.9224

In fact, for the RS(255,153) code, the algorithm performs ideally under all channel conditions, whether using the pre-generated or randomly generated error sequences in the channel. The RS(255,193) code also performs really well except when using the randomly generated error sequences in a

poor channel scenario. Therefore, up till now, the RS(255,153), no source coding and no interleaving combination is the scheme of choice!

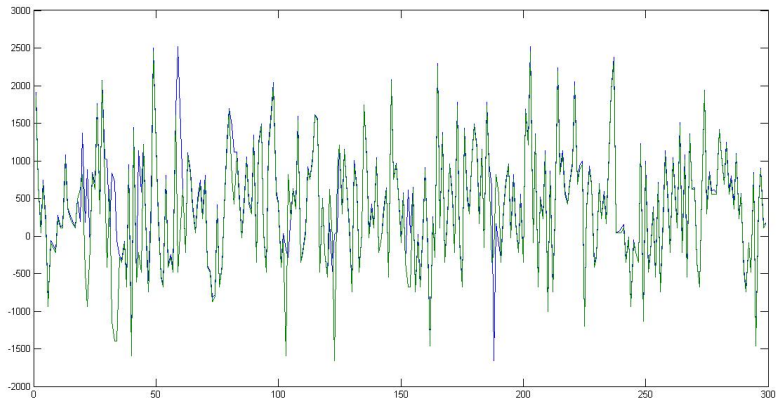

Figure 4.6: No source coding followed by RS code (255,193) in bad channel conditions

4.5 No Source Coding Followed by RS Code and Interleaving

Table 4.20: Results for Random Patterns Using RS(255,223) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$41.6 * 10^{-3}$	10.6303	0.7357

Tables 4.20 - 4.23 show the results obtained in this section. All parameters were improved under all channel conditions. The RS(255,193) code

performance improved immensely from the last section (NMSE: 0.0181 vs 0.0693 SNR: 21.2808 vs 9.4839 SSIM: 0.8795 vs 0.7423), but still not good enough to recommend it as the code of choice for the transmission of CS-EEG signals. The scheme of choice after all the results were obtained is the one that uses no source coding followed by the RS(255,153) code and interleaving.

Table 4.21: Results for Random Patterns Using RS(255,193) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$18.1 * 10^{-3}$	21.2868	0.8795

Table 4.22: Results for Random Patterns Using RS(255,153) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$0.25824 * 10^{-3}$	26.7608	0.9224

Table 4.23: Results for Fixed Patterns Using RS(255,223/193/153) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR(in dB)**	**SSIM**
Good	$0.25824 * 10^{-3}$	26.7608	0.9224
Average	$0.25824 * 10^{-3}$	26.7608	0.9224
Poor	$0.25824 * 10^{-3}$	26.7608	0.9224

4.6 General Comparisons

Table 4.24: Encoder Processing Times of the Approaches

Codeword Data Length	**Time (With Huffman)**	**Time (Without Huffman)**
RS(255,223)	1.75 sec	1.65 sec
RS(255,193)	2.44 sec	2.34 sec
RS(255,153)	4.4 sec	4.3 sec

Table 4.24 shows the average processing time of each algorithm at the encoder side (per epoch) under the 3 different RS codes used. This table only takes into account the time needed for the quantization, source coding, channel coding and packetization to be processed and implemented. In all 3 cases, the no source coding scheme performed slightly faster than their counterparts using Huffman. The Huffman code processing time is 0.1 seconds. The quantization processing time is 0.03 seconds. The interleaving processing time is 0.001 seconds. This means that the bulk of the processing time was due to the RS code. As the data length of the RS code decreases, its processing time increases. This is because the number of RS codewords generated increases, which implies that more RS codewords need to be processed. There is no escaping the use of a channel encoder when transmitting in non-ideal channel conditions. Therefore, the processing times of the RS code have to be tolerated.

Table 4.25 compares the time it takes for the encoder to be processed with and without Huffman, as well as for the encoder part of the CS algorithm. It compares those 3 for two different values of the number of

Table 4.25: Encoder Processing Times Comparison with CS Algorithm using RS(255,223)

q_{bits}	**CS Encoding time**	**Approach w. Huffman**	**Approach w/o Huffman**
6 bits	4 sec once, 0.0016 sec/ep	1.75 sec/ep	1.65 sec/ep
7 bits	4 sec once, 0.0016 sec/ep	2.1 sec/ep	1.96 sec/ep

quantization bits q_{bits}. The first thing we notice is that the processing time for the CS algorithm is the same in both cases. This is because the CS encoder part is processed before quantization is applied to the data. The second thing we observe is that the CS encoder has 2 values in it. The first value of 4 seconds is a one time value for the whole block of data (i.e. for all the epochs). The second value is the same for every epoch, 0.0016 seconds. This is a very small processing time. It was expected to be small because one of the main advantages of the CS algorithm is that its computations are quick at the encoder side, leaving most of the work to be done at the receiver end.

The third column of table 4.25 gives the processing time at the encoder of the Huffman and RS(255,223) combination. The processing time of the RS code is once again evident. The processing time increased by 0.35 seconds per epoch when the number of quantization bits was increased. The increase in processing time is also the case in the fourth column of that same table. This column gives the encoder processing times for the no source coding and RS(255,223) scheme. As it can be seen, the increase in processing time is 0.31 seconds per epoch.

Table 4.26 shows the difference in the number of bits sent per epoch with

Table 4.26: Number of bits sent per epoch

Codeword Data Length	**With Huffman**	**Without Huffman**
RS(255,223)	2170	2295
RS(255,193)	2434	2550
RS(255,153)	2938	3060

and without the use of the Huffman code. The difference is approximately 125 symbols in the 3 cases of the RS code used. That difference takes into account the number of symbols it requires to transmit the dictionary of the Huffman code with every epoch. It is possible to use just one dictionary for all the epochs, but this would not be the optimal compression scheme. It is better to send with each epoch its own dictionary of codewords, even though this leads to added symbols being transmitted. This table gives the only area that the scheme with Huffman coding actually outperformed the scheme without Huffman coding. The addition of the Huffman code achieved a 5.45 % compression ratio when compared to the other scheme. However, if we consider that the better reconstruction results occurred when Huffman isn't used, Huffman is not a very suitable algorithm for the transmission of CS-EEG signals in WBSNs, a sentiment echoed in [17].

The total data available is two hours of EEG data from a patient suffering a seizure. The total processing time of this data (encoding and decoding) is over 34 hours (the CS decoding algorithm takes about 16 seconds per epoch to reconstruct the 23 channel EEG signals). We used 5 minutes of this data for our purposes. The total processing time for 5 minutes was 1 hour 30 minutes approximately. The data was transmitted through different

channel conditions for a total of 132 times.

In conclusion, after all the results were presented and analyzed, the algorithm of choice would be the one presented in section 3.5, i.e. the one where no source coding was applied, followed with the RS(255,153) code and an interleaver. True that using the RS(255,153) means sending more encoded symbols than the RS(255,223) code because it encodes every 153 data symbols into 255 symbols in comparison to 223 data symbols encoded into 255 symbols, but the data arrives correctly and is properly reconstructed, which in the end is the most important condition. There is always a small and sometimes inexpensive trade-off to achieve this condition, and this time is one of them.

Other schemes were considered at first. Concatenated Huffman coding was an idea under consideration, but once the results of the conventional combination of algorithms came out, the idea was dropped. Another potential was to study the correlation of the EEG signals and exploit it, but since the CS algorithm used did that very well, there wasn't any correlation left to exploit, so this idea was let go as well. A third approach was to apply low-density parity-check codes (LDPC), but for lack of time, this approach was not implemented.

Chapter 5

Summary and Conclusions

5.1 Main Results

This thesis is concerned with elongating the battery life at the EEG sensor node. To save on the energy consumed by wireless transmission, the acquired EEG is first compressed using the block-sparse Bayesian learning (BSBL) compressed sensing (CS) approach and quantized. We then studied the performance of two methods for transmitting the CS-quantized EEG signals.

The first method presented in section 2 of chapter 3 proposed the use of a Huffman code followed by a Reed-Solomon (RS) encoder to protect the transmitted data. The codeword length of the RS code was varied to find a suitable length capable of correcting enough errors caused by the channel. The results were acceptable under very good and average channel conditions but deteriorated quickly when the channel conditions worsened. Another crucial point that we observed is the computation time needed at the encoder for this whole scheme to be processed, which proves to be costly in terms of energy.

In section 2 of chapter 3 we add interleaving right after the packetization of the data. The results yielded a better performance in all cases when compared to the scheme without interleaving. The only advantage of incor-

porating Huffman coding was the compression gain (5.45%) it achieved on the data that was already compressed using CS. This means that the total compression achieved including CS and Huffman is 90.545 %. This gain however is not enough to justify the use of Huffman, since its performance under noisy or not ideal channel conditions is unsatisfactory.

The second method (presented in section 4 of the same chapter) proposed transmitting the CS compressed data without any source or further lossless coding. It simply applies the RS code before transmitting the data. The results showed a noticeable improvement in performance when compared to the first proposed method, as evidenced by the results under poor channel conditions where the reconstruction of the received data was closer to the actual transmitted data.

The final method applied interleaving to the above scheme (no further compression followed by an RS encoder). Its results were again superior to the case where no interleaving was applied. In fact, this scheme is the one that outperformed all other schemes and is the scheme of choice for the transmission of compressed sensing EEG data, for the case where the RS(255,153) is used.

Before starting the experiments, we expected that the Huffman code would not perform well in very noisy channels, but we did not expect the results to be as underwhelming as they ended up being. They were in fact worse than our original expectation. From the point of view of decreasing the number of bits transmitted, the Huffman code would be optimal under ideal conditions because it can perfectly reconstruct the compressed data. As mentioned above, it decreased the total number of bits transmitted

per epoch by 5.45%. However, its poor performance under noisy channel conditions would hinder its application for our purposes. The processing times of the encoder are given for every scenario, showing that the scheme of sections 3.4 and 3.5 (using BSBL CS compression, 6 bits quantization, (255,223/193/153) Reed Solomon coding followed by interleaving) were processed slightly faster than the schemes in sections 3.2 and 3.3 (using BSBL CS compression, 6 bits quantization, Huffman coding, (255,223/193/153) Reed-Solomon coding followed by interleaving).

The bulk of the processing time was from the RS encoder. The Huffman code account for only 0.1 seconds of processing time. Quantization and interleaving accounted for only 0.031 seconds. The processing times of the RS encoder increased when the data length in each codeword decreased.

Throughout the thesis, we had to design the schemes such that they would suit the improved BSBL algorithm of [15]. To the best of our knowledge, this algorithm which uses CS is close to being optimal, even though we can not prove it mathematically. But the results obtained using this algorithm were impressive and they formed the upper limit that the work in this thesis was trying to reach. We came extremely close falling only 0.9% off the upper limit value of the structural similarity index and 3.19% off the upper limit value of the normalized mean square error.

Considering that the encoder stage of the wireless body sensor network (WBSN) must have little processing complexity due to the constraint on energy, we improved that stage through the removal of the whole source coding block since already the 90 % compression ratio achieved using the algorithm of [15] is quite high. Usually, lossy algorithms are faster in terms of

running time and achieve higher compression rates than lossless algorithms such as the Huffman code, however, they do not reconstruct the signal at the receiver end perfectly and they would still add more processing time compared to cases where source coding is not present at all.

5.2 Future Work

The channel coding algorithm used is a very efficient algorithm, both in terms of speed and its error correction and detection capabilities. Lately low-density parity-check (LDPC) codes are being applied in many applications that have bandwidth constraints. They are used most notably in the satellite transmission of digital television. Trying to implement them into the world of WBSNs could prove to be useful and hence is one way that could possibly improve the results obtained in this work.

The actual energy required for transmitting the bits at the sensor node should be studied in detail and in conjunction with building a hardware prototype for this problem.

Appendix A

The whole structure of the thesis started with the data being sampled with compressed sensing applied to it, then quantizing it, applying a source coding algorithm followed by a channel encoder, transmitting it, then applying channel and source decoding to every epoch, before all epochs were reconstructed using the algorithm of [15].

The results given in chapter 4 were for all the schemes that started before the quantization of the data up until source decoding. They did not include the blocks before and after, which is basically the work done in [15]. This Appendix gives the results of the whole framework including that of [15], i.e. from before the data was sampled till after it was reconstructed using the improved BSBL-BO algorithm.

According to [15], the best results achieved by the algorithm used for a 90 % compression ratio using a discrete cosine transform dictionary (without any source coding, channel coding or interleaving, and in ideal channel cases, that is there weren't any errors introduced by the channel) were as follows:

$NMSE = 0.0533$

$SNR = 12.7347$

$SSIM = 0.8523$.

These results are the upper limit that this thesis attempted to reach as closely as possible, and were it not for the quantization error introduced at the encoder, it would have been possible to reach those numbers. The tables

Table A.1: Results for Random Patterns Using RS(255,223)

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.1482	8.1202	0.7596
Poor	FAILED	FAILED	FAILED

Table A.2: Results for Random Patterns Using RS(255,193)

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.0924	10.1137	0.8213
Poor	0.6986	1.4722	0.3401

below show the results achieved for each scheme applied.

An important note is that the results above are slightly different than the values found in [15]. This is because the total data processed was less since the processing time of the total data available was over 34 hours. After running the algorithm once and then running the same algorithm for a lesser amount of data, it was clear that the values would remain extremely close, but the processing time of the data would be only an hour and a half.

Table A.3: Results for Random Patterns Using RS(255,153)

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.0597	11.8943	0.8412
Poor	0.4847	1.9502	0.3912

Table A.4: Results for Random Patterns Using RS(255,223) and Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.0915	10.1271	0.8067
Poor	FAILED	FAILED	FAILED

Table A.5: Results for Random Patterns Using RS(255,193) and Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.5736	2.2068	0.3082

Table A.6: Results for Random Patterns Using RS(255,153) and Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.2586	6.0035	0.6995

Table A.7: Results for Random Patterns Using RS(255,223) with no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.0893	10.0151	0.8249
Poor	0.3568	4.1059	0.5152

Table A.8: Results for Random Patterns Using RS(255,193) with no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.2998	5.0786	0.6026

Table A.9: Results for Random Patterns Using RS(255,153) with no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.055	12.5925	0.8446

Table A.10: Results for Random Patterns Using RS(255,223) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.2133	6.0178	0.6142

Table A.11: Results for Random Patterns Using RS(255,193) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.1029	10.0081	0.8124

Table A.12: Results for Random Patterns Using RS(255,153) with Interleaving, no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.055	12.5925	0.8446

Table A.13: Results for Fixed Patterns Using RS(255,223) with and without Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	FAILED	FAILED	FAILED

Table A.14: Results for Fixed Patterns Using RS(255,193) with and without Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	FAILED	FAILED	FAILED

Table A.15: Results for Fixed Patterns Using RS(255,153)

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.0681	10.876	0.8446

Table A.16: Results for Fixed Patterns Using RS(255,153) and Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.055	12.5925	0.8446

Table A.17: Results for Fixed Patterns Using RS(255,223) with no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.343	4.6471	0.4801

Table A.18: Results for Fixed Patterns Using RS(255,193/153) with no Source Coding, with and without Interleaving

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.055	12.5925	0.8446

Table A.19: Results for Fixed Patterns Using RS(255,223) with Interleaving no Source Coding

Channel	**NMSE**	**SNR**	**SSIM**
Good	0.055	12.5925	0.8446
Average	0.055	12.5925	0.8446
Poor	0.055	12.5925	0.8446

Bibliography

[1] A.M. Abdulghani, A.J. Casson, and E. Rodriguez-Villegas. Quantifying the performance of compressive sensing on scalp EEG signals. *Applied Sciences in Biomedical and Communication Technologies (ISABEL), 2010 3rd International Symposium on*, pages 1–5, 2010.

[2] C.O. Arinze, V.E. Idigo, C.O. Ohaneme, and V.N. Agu. Simulation and evaluation of high density cognitive hotspot network based on IEEE 802.11 minipop access point series. *International Journal of Computers and Distributed Systems*, 4(2):1–10, 2014.

[3] S. Aviyente. Compressed sensing framework for EEG compression. *IEEE/SP Statistical Signal Processing 14th Workshop 2007*, pages 181–184, 2007.

[4] World Bank. Population ages 65 and above (% of total). `http://data.worldbank.org/indicator/SP.POP.65UP.TO.ZS/countries`. Accessed: 06/06/2014.

[5] E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[6] Jose Luis Cuevas-Ruiz Diana Alejandra Sanchez-Salas and Miguel

Gonzales-Mendoza. *MATLAB - A Fundamental Tool for Scientific Computing and Engineering Applications - Volume 2*. InTech, Morn Hill, 2012.

[7] S. Fauvel. Energy-efficient compressed sensing frameworks for the compression of electroencephalogram signals, 2013.

[8] S. Fauvel, A. Agarwal, and R. Ward. Compressed sensing and energy-aware independent component analysis for compression of EEG signals. *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 973–976, 2013.

[9] S. Fauvel and R.K. Ward. An energy efficient compressed sensing framework for the compression of electroencephalogram signals. *Sensors*, 14(1):1474–1496, 2014.

[10] Simon Haykin. *Communication Systems*. Wiley, Hoboken, New Jersey, 2001.

[11] F.J. Hermann. Randomized sampling and sparsity: getting more information from fewer samples. *Geophysics*, 75:173–187, 2010.

[12] H.H. Hung and L.J. Chen. An analytical study of wireless error models for bluetooth networks. *Advanced Information Networking and Applications*, pages 1317–1322, 2008.

[13] M. Khazi, A. Kumar, and V. M-J. Analysis of EEG using 10:20 electrode system. *International Journal of Innovative Research in Science, Engineering and Technology*, 1(2):185–191, 2012.

[14] D. Lechuga, O. Escandn, M. Corona, C. Montoya, R. G., A. Anguiano, C. Garca, and J. Moctezuma. Electroencephalographic abnormalities in patients with idiopathic insomnia. *Neuroscience and Medicine*, 2(3):178–184, 2011.

[15] H. Mahrous, R. Ward, and Z. Zhang. Multi-channel compression of EEG signals via compressed sensing. In the process of being published.

[16] H. Mamaghanian, N. Khaled, D. Atienza, and P. Vandergheynst. Compressed sensing for real-time energy-efficient ECG compression on wireless body sensor nodes. *IEEE Transactions on Biomedical Engineering*, 58(9):2456–2466, 2011.

[17] R. Mc Sweeney, C. Spagnol, E. Popoviv, and L. Giancardi. The impact of source and channel coding in the communications efficiency of wireless body area networks. *International Journal on Advances in Software*, 2(4):337–348, 2009.

[18] James-A.B. Milner, B.P. Analysis and compensation of packet loss in distributed speech recognition using interleaving. *Proceedings of INTERSPEECH*, pages 1–4, 2003.

[19] M. Moudgill, A. Iancu, and D. Iancu. Galois field instructions in the sandblaster 2.0 architecture. *International Journal of Digital Multimedia Broadcasting*, pages 1–5, 2009.

[20] University of Virginia. Switching technology. `http://www.cs.virginia.edu/~mngroup/projects/mpls/documents/thesis/node8.html`. Accessed: 14/08/2014.

[21] Charles L. Phillips and John M. Parr. *Signals, Systems, and Transforms.* Prentice Hall, Upper Saddle River, New Jersey, 2008.

[22] John Proakis. *Digital Communications.* McGraw-Hill, New York, 2000.

[23] P. Rmeily and H. Mahrous. Bayesian learning algorithm for compressive sensing of non-sparse EEG signals. -. Class project.

[24] N. Sireesha and V. Prasanth. Iterative multivariate interpolation for low complexity Reed-Solomon codes. *International Journal of Modern Engineering Research*, 2(5):3769–3772, 2012.

[25] Highland Communications Technologies. Introduction to Reed-Solomon codes. `http://www.highlandcomm.com/reed_solomon_codes.htm`. Accessed: 04/08/2014.

[26] M. Unser. Sampling—50 Years after Shannon. *Proceedings of the IEEE*, 88(4):569–587, 2000.

[27] Z. Zhang, T. Jung, S. Makeig, and B. Rao. Compressed sensing of EEG for wireless telemonitoring with low energy consumption and inexpensive hardware. *IEEE Transactions on Biomedical Engineering*, 60(1):221–224, 2013.

[28] Z. Zhang, T.P. Jung, S. Makeig, and B.D. Rao. Compressed sensing for energy-efficient wireless telemonitoring of non-invasive fetal ECG via block sparse Bayesian learning. *IEEE Transactions on Biomedical Engineering*, pages 1–4, 2014.

[29] Z. Zhang and B. D. Rao. Extension of SBL algorithms for the recovery of block sparse signals with intra-block correlation. *IEEE Transactions on Signal Processing*, 61(8):2009–2015, 2013.

[30] W. Zhou, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.