WIRELESS COHERENT OFDM MODEM

A Design Project Report
Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial fulfillment of the requirements for the Degree of
Master of Engineering (Electrical and Computer)

By
Francisco Bastidas, Rahul Kopikare and Gaëlle Protat
Project advisor: Anna Scaglione
Degree date: May 2002

Abstract

Master of Electrical and Computer Engineering

Cornell University

Design Project Report

Project title: Wireless coherent OFDM modem.

Authors: Francisco Rafael Bastidas, Rahul Kopikare and Gaëlle Protat

Abstract:

This project aim was to build wireless software modem for data communication between two computers using an acoustic interface in the voice frequency range (20Hz–20,000Hz). The transmitting antenna is a speaker (frequency response of: 90Hz – 20,000Hz) and the receiving antenna is a microphone (frequency response of: 100Hz – 16,000Hz). The test files used as information files were text files.

This goal was attained both in an incoherent scheme and in a coherent scheme. Build under Matlab code, our modem uses OFDM (orthogonal frequency division multiplexing) modulation, synchronization by LMS sequence, channel estimation (no equalizer) via pilot tones. The symbols are either PSK or ASK for a constellation size of 2 or 4. To optimize the probability of error, these symbols were mapped using Gray mapping.

Report Approved by
Project Advisor: _____ Date: _____

Executive summaries

Rahul Kopikare

Rahul's main contribution was in the design and coding of the transmitter and the receiver of the modem. His main work was writing functions and testing them on a simulator also developed by him. He contributed with Rafael towards the initial simulations to find the QPSK and QAM error probability. He also contributed with Gaëlle towards experiments in finding the channel length.

Gaëlle Protat

Gaëlle main part was to find the correct process for the synchronization (see section 4.1) and the channel estimation (see section 4.3). Assumptions about the channel made and criteria defined, she was able to search for the most appropriate synchronization sequence and try different ways of estimating the channel. As part of a group project, she also helped in the brainstorming and choices for the other parts of the modem.

Francisco Bastidas

Francisco constantly performed simulations of both the Coherent and Incoherent schemes to verify their proper functionality. He also updated these systems with the appropriate parametric values and changed the designs until the results matched the theoretical derivations. He successfully normalized the energy of the transmitted signal to work with the proper Signal to Noise Ratio. He performed extensive experimentation on the Incoherent system by manipulating critical parameters and both derived an implemented the Optimum receiver for detection. He dedicated special attention and supported his colleagues during the design, experimentation and implementation of the Coherent Scheme. He performed simulations on this system, worked on several procedures to estimate the channel and anxiously pursued efficiency in the scheme.

Table of contents

1 Introduction

The aim of this project was to design and implement an OFDM based wireless acoustic software modem for local wireless communication between personal computers. OFDM is chosen because it is robust against inter symbol interference (ISI) and noise. Also it doesn't need perfect synchronization of the receiver with the received signal. These factors of OFDM are making it an emerging technology for future wireless communication. The communication frequency is chosen to be in the Audio band because it requires cheap hardware. Also this project is envisaged to have its future in underwater communication where audio frequencies (sonar waves) triumph over radio frequencies (RF).

1.1 Hardware description.

The hardware we used is:

- A speaker with specifications:
  - Input sensitivity: 82 dB at 1 W, 1 m.
  - Frequency range: 90 Hz to 20 kHz.
  - Amplifier distortion at rated power: 1% THD maximum.
  - Transducer: 3-inch full-range driver, magnetically shielded.
  - Maximum noise: –60 dB with no signal in.
  - Maximum acoustic output: >92 dB at 1 kHz, 1 m both speakers driven.
- A microphone with specifications:
  - Frequency range of 100Hz – 16,000Hz.
  - Sensitivity: -67dB/µBar -47dBV/Pascal +/- 4dB.

This hardware was chosen because it was cheap and off the shelves.

1.2 Software description

To convert from digital to analog (D/A) and then from analog to digital (A/D), we used the Matlab functions wavplay and wavrecord respectively.

Wavplay Matlab help file (excerpt)

Wavplay plays sound using Windows audio output device.

wavplay(Y,FS) sends the signal in vector Y with sample frequency of FS Hertz to the Windows WAVE audio device.

We used a sampling frequency of 8000 Hz in our experiment, because we found that the devices were optimized for this sampling frequency.

The range of the quantization is -1.0 <= y <= 1.0, and the values outside that range are clipped. The number of bits per samples is 16 in our case. Thus the quantization noise variance is:

$$q = \frac{2}{2^{16}}$$

$$s_{noise}^{2} = \frac{q^2}{12} = \frac{4}{12 \times 2^{16}} = \frac{1}{196608} \approx 5{,}1 \cdot 10^{-6}$$

$NSR \approx 6.02 \times b = 96.32$ (without the clipping)

Wavrecord Matlab help file (excerpt)

Wavrecord records sound using Windows audio input device.

wavrecord(N,FS,CH) records N audio samples at FS Hertz from CH number of input channels from the Windows WAVE audio device.

We used the same sampling frequency of 8000 Hz at the receiver. By default, the CH value is put to 1 (which is our case). The quantization has the same range and thus the same quantization noise variance.

1.3 Channel assumptions and properties.

With this information, our first job on this project was to identify the channel and its properties under some initial assumptions:

- The channel gave linear distortion,

- The channel was time invariant,
- The noise could be considered to be white Gaussian noise.

Thus, the first step was to estimate the variance of the noise to be able to simulate our modem in realistic conditions of noise and channel distortion before doing the actual experiments. Our estimation of the variance was 0.01, in a silent room.

We then tried to see if our channel was frequency selective by sending tones at different frequencies and viewing the possible distortion. From the analysis, we found that the channel is a frequency selective channel. We then estimated a minimum frequency below which transmission was too much distorted and thus not possible. We denoted such frequency fmin in the global_var file.

2 Background and multi-carrier transmission

Orthogonal Frequency Division Multiplexing (OFDM) allows the transmission of high data rates over extremely hostile channels. OFDM splits the available bandwidth into many narrow-band channels. Multi-Carrier (MC) modulation has several advantages over single-carrier communication. When the channel is frequency selective the distinctive advantage of OFDM is that, using a guard interval whose duration exceeds the channel's impulse response length, it allows to decompose the channel in a set of independent flat fading subchannels over which equalization amonts to a single phase compensation. As a consequence, the guard protection also diminishes the need for time-domain equalization at the receiver, i.e. the Inter-Symbol Interference(ISI) is neglible. Viewing the problem in another way, if a block of length M is chosen so that $T = MTs >> LTs$, where LTs is the length of the channel impulse response and Ts is the baud-rate , then the effect of the inter-symbol interference (ISI) is negligible on each sub-carrier. To eliminate the ISI altogether at the expense of a small decrease in capacity, the guard interval of length $PTs \geq LTs$ is inserted between successively modulated OFDM blocks. The reduction of Capacity is due to a loss of bandwidth efficiency that comes from the addition of the cyclic prefix (guard interval). In standard OFDM each OFDM block has length which is $M = (N + L)$ where N is the number of subcarriers and hence symbols that we transmit. Therfore if MTs is the duration of the OFDM block the symbol rate is:

$$Rs = \frac{N}{MTs} = \frac{N}{(N+L)Ts} = \frac{1}{(1+\frac{L}{N})Ts} \qquad (2.1)$$

which tends to 1 if $N \to \infty$.

Another advantageous characteristic of an OFDM system relates to its ease in implementation. This refers to the fact that modulation can be performed digitally by a simple Inverse Discrete Fourier Transform (IDFT) which can be implemented very efficiently as a Inverse Fast Fourier Transform (IFFT) (The complexity of the IFFT is $10N \log N$). Naturally, the FFT the correspondent demodulator at the receiver.

Unlike single carrier methods, OFDM distributes its power over multiple orthogonal carriers which makes the modulation power efficent and also gives the system the opportunity to exploit the frequency selectivity of the channel as form of diversity. The concept of orthogonality allows each carrier to be separated from the others at the receiver. By using an IFFT for modulation we implicitly chose the spacing of the subcarriers in such a way that at the frequency where we evaluate the received signal all other signals are zero (Inter-carrier interference (ICI) equal to zero). In our design, the essence of OFDM being intrinsically orthogonal relies on an environment of digital manupilation. When our digital sequence is converted to anolog transmision, the spacing between carriers, Δf , equals the inverse of the time durration of the transmitted symbols ($\frac{1}{NTs}$), causing the carriers to be orthogonal.

We will now proceed to describe the OFDM system and some of its advantages mathematically. In order to do this, however, we will first begin by explaining some of the digital modulations that we have chosen for our design. Specifically, we will discuss PSK and QAM.

2.1 PSK

Phase Shift Keying (PSK) modulators map input bits into output waveforms of the form $s(t) = A\cos(? t + ?)$, and the information bits are stuffed in the phase $?$. As a quick example, in BPSK, when bit 0 goes into the modulator, the modulator spits out the waveform $A\cos(? ct + 0°)$. If bit 1 struts into the modulator, it pops out $A\cos(? ct + 180°)$. In a similar manner, 4-PSK outputs a different waveform for every two bits that come into the modulator. Utilizing the same idea, higher PSK (8, 16, etc) constellations can be constructed.

2.2 QAM

Another modulation of interest to us is Quadrature Amplitude Modulation (QAM). QAM consists of two independently amplitude-modulated carriers in quadrature. The QAM signal representation is expressed as a sum of one sine and a cosine term. The correspondent QAM constellations (4, 16, etc) are obtained by projecting the orthonormal basis into the constellation plane.

In our digital scheme, these modulations basically generate the data symbols that are transformed by the IFFT transform. However, only half of the complex subcarriers will be used because we need to construct a real signal as will be clear in section 2.4. Take 4-PSK for instance. In this scenario, our digital modulator will map two bits into a symbol chosen from a constellation of size C. Each data symbol corresponds to $\log_2 C$ bits. Clearly, the previously described waveforms are now the data symbols.

2.3 FFT-based OFDM system

Let $\mathbf{x}_n$ be the data symbol block at each epoch n. Then:

$$\mathbf{x}_n = \{xn0, xn1, ..., xnN-1\} \tag{2.2}$$

where n is the block index.

The digital OFDM signal is:

$$\mathrm{X(n)} = \tilde{s}(nTs) = A\sum_{k=0}^{N-1} x(k) e^{\frac{j2\mathrm{p}kn}{N}}, n = 0,1,...,N-1 \tag{2.3}$$

From (2.3), we have the vector $\mathbf{X} = \{X0, X1, ..., XN-1\}$ which represents the Inverse Discrete Fourier Transform (IDFT) of the vector $A\mathbf{x} = A\{x0, x1, ..., xN-1\}$.

Once we have performed the IDFT, the sampled sequence $\{X0, X1, ..., XN-1\}$ is converted into an analog signal and transmitted.

As mentioned before, a guard of length P can be attached to the sequence $\mathbf{X}$. This guard protection is also referred as cyclic prefix and its length is assumed to equal or

exceed the channel length, L. Then, the transmitted sequence with a guard interval appended to it is:

$$X_n^p = X_{(n)N} \tag{2.4}$$

$$= A\sum_{k=0}^{N-1} x(k)e^{\frac{j2\pi kn}{N}}, n = 0,1,...,N+P-1 \tag{2.5}$$

where $(n)_N$ is the residue of n modulo N, thereby, the name cyclic prefix. The OFDM base-band modulator is shown in Fig. 2.1

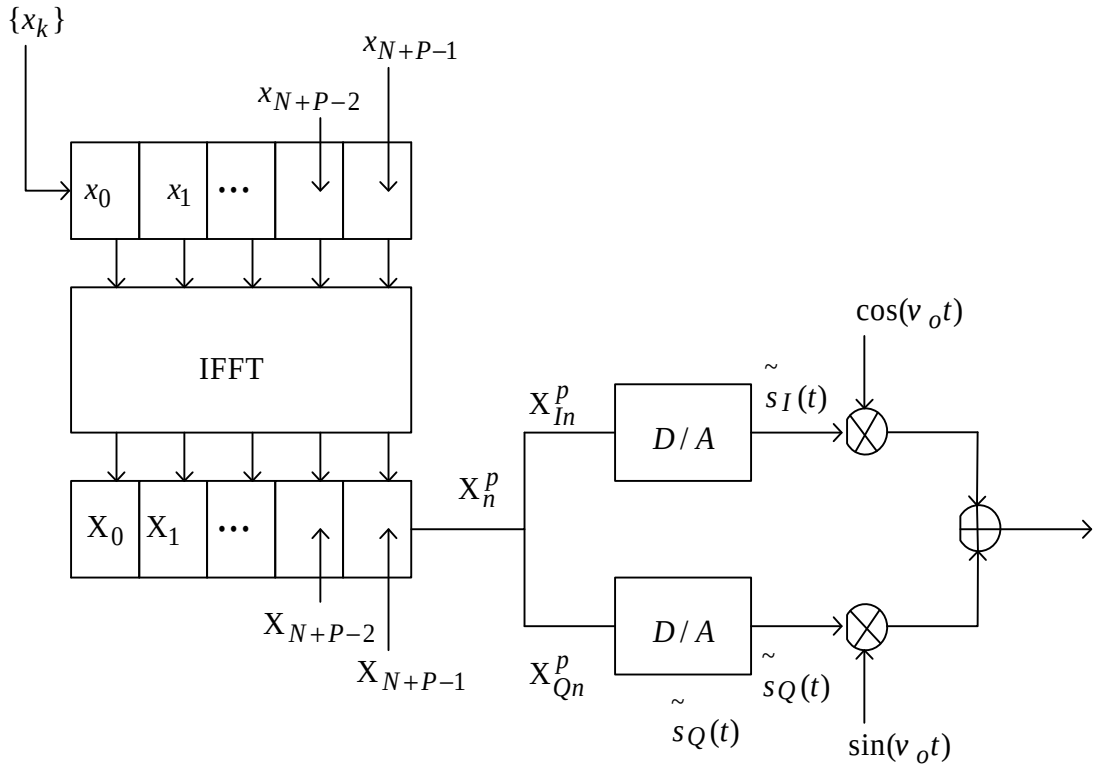


Figure 2.1

The discrete-time channel is given by $\{h_0, h_1, ..., h_L\}$ where LT_s is the duration of the channel impulse response. The discrete-time convolution of the transmitted sequence

$\{X_n^p\}_{n=0}^{N+P-1}$ with the discrete-time channel leads to the following received sequence $\{Y_n^p\}$ defined as:

$$Y_n^p = \sum_{m=0}^{L} p(m) X_{n-m}^p \tag{2.6}$$

At the receiver, the first $P \geq L$ samples of the block are damaged by ISI from the previous block. The ISI from the previous block affecting each set of IFFT coefficients is removed by removing the cyclic prefix. The remaining portion of data is:

$$\begin{aligned} Y_n &= Y_{P+(n-P)N}^p \\ &= \sum_{m=0}^{L} p(m) X((n-m)_N), 0 \leq n \leq N-1 \end{aligned} \tag{2.7}$$

Successively, the OFDM demodulator performs an FFT on the vector $\mathbf{Y} = \{Y_n\}_{n=0}^{N-1}$, therefore the demodulated sequence is:

$$\begin{aligned} D(i) &= \frac{1}{N} \sum_{n=0}^{N-1} Y(n) e^{\frac{-j2\pi in}{N}} \\ &= A \sum_{m=0}^{L} h(m) \sum_{k=0}^{N-1} x(k) e^{\frac{-j2\pi m}{N}} \frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{j2\pi kn}{N}} e^{\frac{-j2\pi ni}{N}} \\ &= A \sum_{m=0}^{L} h(m) \sum_{k=0}^{N-1} x(k) e^{\frac{-j2\pi m}{N}} \frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{j2\pi n(k-i)}{N}} \end{aligned} \tag{2.8}$$

In order to proceed further, we need to realize that in (2.8), the last term $\frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{j2\pi n(k-i)}{N}}$ can be decomposed using a geometric series

$\sum_{n=0}^{N-1} r^n = \frac{1-r^{(N-1)+1}}{1-r}$ Applying this to the last term in (2.8) we have:

$$= \frac{1 - e^{j2\pi(k-i)}}{1 - e^{\frac{j2\pi(k-i)}{N}}} \qquad (2.9)$$

From (2.9) we notice that when $k \neq i$, the numerator will always be zero and therefore the whole expression becomes zero at these discrete values. However, when $k = i$, we have $\frac{0}{0}$, which, in this case, means that the limit of this function when $k = i$ is N. Thus:

$$\frac{1}{N}\sum_{n=0}^{N-1} e^{\frac{j2\pi n(k-i)}{N}} = \frac{1}{N} N\delta(k-i) = \delta(k-i) \qquad (2.10)$$

Then:

$$D(i) = A\sum_{m=0}^{L} h(m)\sum_{k=0}^{N-1} x(k) e^{\frac{-j2\pi m}{N}} \delta(k-i) \qquad (2.11)$$

It is important to note, that in (2.11), k has to equal i to obtain a non-zero result from the summation where the indexes belong to. Therefore:

$$D(i) = A\sum_{m=0}^{L} h(m)x(i)e^{\frac{-j2\pi mi}{N}}$$

$$= Ax(i)\sum_{m=0}^{L} h(m)e^{\frac{-j2\pi mi}{N}}, 0 \le i \le N-1 \qquad (2.12)$$

$$= Ax(i)H(e^{\frac{-j2\pi i}{N}}), 0 \le i \le N-1 \qquad (2.13)$$

Clearly, (2.12) demonstrates that the ISI has been successfully removed and (2.13) shows that the effect of the channel is multiplication of the symbol by the channel fourier coefficient $H(e^{\frac{-j2\pi i}{N}})$ at $\frac{i}{N}$. In the presence of noise, $D(i)$ is used to make data symbol decisions.

2.4 Making a real signal

As mentioned earlier, our design requires special attention as it relates to the data being modulated through he IFFT. This is because the output of the IFFT must be real since we are not going to modulate the signal with a carrier, since our channel is basically base-band. Furthermore, once the symbols are generated, a sequence of zeros is placed ahead of them. This gives us the advantage to be selective during transmission and completely eliminate the undesired low frequencies which microphone and speakers are filtering away. Naturally, this ability to cut off the low frequencies is fairly intuitive and comes from the fact that the number of zeros will multiply the same number of carries inside the IFFT transform. As a consequence, the signal power at those specific carriers is of zero value. A more interesting result however, is the issue of making the IFFT real. In order to eliminate the imaginary part, our design creates symmetry by concatenating a "special" sequence of data after the existing (zeros + symbols) information sequence. This "special" sequence is obtained by flipping the (zeros + symbols) data sequence and conjugating it. The following example, clearly demonstrates the mentioned approach.

Let $Symb$ be a set of symbols and let $Zero$ be the zeros at the undesired low frequencies:

$$Symb = [1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 2]$$

$$Zero = [0 \quad 0]$$

Then, the zeros are placed ahead of the symbols leading the following result:

$$ZeroSymb = [0 \quad 0 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 2]$$

Next, the $ZeroSymb$ sequence is flipped in the following manner:

$$FlipZeroSymb = [2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 0 \quad 0]$$

and, if the symbols are complex, the $FlipZeroSymb$ sequence is also conjugated. Finally, the $FlipZeroSymb$ sequence is attached immediately after the $ZeroSymb$ sequence as follows:

$$SymmSeq = [0 \quad 0 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 2 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 0 \quad 0]$$

The last thing to successfully achieve the goal of making the IFFT real is to add one more zero at the beginning of $SymmSeq$ which yields the following final sequence that enters the IFFT:

$SeqIIFT = [0 \quad 0 \quad 0 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 2 \quad 2 \quad 1 \quad 3 \quad 4 \quad 2 \quad 4 \quad 3 \quad 1 \quad 0 \quad 0]$

This zero represents the 0^{th} frequency and therefore should not be considered for purposes of creating symmetry. To prove that the IFFT transform is real given a symmetrical sequence, we use a brief example. Let A be a symmetrical sequence as follows:

$A = [0 \quad 1 \quad 2 \quad 2 \quad 1]$

Recalling the definition of the IFFT transform as:

$$\mathrm{x}(n) = \frac{1}{N}\sum_{k=0}^{N} \mathrm{X}(k) e^{\frac{j2pkn}{N}}, n = 0,1,2,..., N-1$$

we obtain the following calculation:

$$\mathrm{x}(0) = \frac{1}{5}\{A(0) + A(1) + A(2) + A(3) + A(4)\}$$

$$\mathrm{x}(1) = \frac{1}{5}\{A(0)e^{j0} + A(1)e^{j2\mathrm{p}/5} + A(2)e^{j4\mathrm{p}/5} + A(3)e^{j6\mathrm{p}/5} + A(4)e^{j8\mathrm{p}/5}\}$$

…..

…..

$$\mathrm{x}(4) = \frac{1}{5}\{A(0)e^{j0} + A(1)e^{j8\mathrm{p}/5} + A(2)e^{j16\mathrm{p}/5} + A(3)e^{j24\mathrm{p}/5} + A(4)e^{j32\mathrm{p}/5}\}$$

Where $X(k) = A(k)$, $\{A(0) = 0, A(1) = 1, A(2) = 2, A(3) = 2, A(4) = 1\}$ and $N = 5$. Decomposing the exponents, we can express $\{\mathrm{x}(0), \mathrm{x}(1),....., \mathrm{x}(4)\}$ in terms of $\cos(x)$ as the real part and $\sin(x)$ as the imaginary part. It turns out, that because of symmetry all the imaginary parts will cancel each other, leaving a pure real signal. Take $\mathrm{x}(4)$ for instance, we have:

$$\mathrm{x}(4) = \frac{1}{5}\left\{\cos\left(\frac{8\mathrm{p}}{5}\right) + j\sin\left(\frac{8\mathrm{p}}{5}\right) + 2\left(\cos\left(\frac{16\mathrm{p}}{5}\right) + j\sin\left(\frac{16\mathrm{p}}{5}\right)\right) + 2\left(\cos\left(\frac{24\mathrm{p}}{5}\right) + j\sin\left(\frac{24\mathrm{p}}{5}\right)\right) + \cos\left(\frac{32\mathrm{p}}{5}\right) + j\sin\left(\frac{32\mathrm{p}}{5}\right)\right\}$$

$$\mathrm{x}(4) = \frac{1}{5}\left\{2\cos\left(\frac{8\mathrm{p}}{5}\right) + 4\cos\left(\frac{16\mathrm{p}}{5}\right)\right\}$$ Similarly:

$$\mathrm{x}(0) = A(1) + A(2) + A(3) + A(4)$$

$$x(1) = \frac{1}{5}\left\{2\cos\left(\frac{2p}{5}\right) + 4\cos\left(\frac{4p}{5}\right)\right\}$$

$$x(2) = \frac{1}{5}\left\{2\cos\left(\frac{4p}{5}\right) + 4\cos\left(\frac{8p}{5}\right)\right\}$$

$$x(3) = \frac{1}{5}\left\{2\cos\left(\frac{6p}{5}\right) + 4\cos\left(\frac{12p}{5}\right)\right\}, \text{ thus:}$$

$\mathbf{x} = [x(0) \quad x(1) \quad x(2) \quad x(3) \quad x(4)]$ which is real.

For purposes of transmission, it is necessary to normalize the energy of the transmitted signal to one. This is accomplished by multiplying the IFFT transform by a constant as follows.

The IFFT transform is:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N} X(k)e^{\frac{j2pkn}{N}}, n = 0,1,2,..., N-1$$

and the corresponding FFT transform is:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{\frac{-j2pkn}{N}}, k = 0,1,2,..., N-1$$

We notice from the IFFT that our symbols are multiplied by a gain of 1/N, whereas the noise that is added during transmission does not have this characteristic. To control the energy per symbol, there exists the necessity to normalize our signal with respect to the noise by multiplying the IFFT transform by a constant. Fixing the energy per symbol to a value of 1, requires introducing a gain of $\sqrt{N}$ which leads to the following normalized IFFT transform:

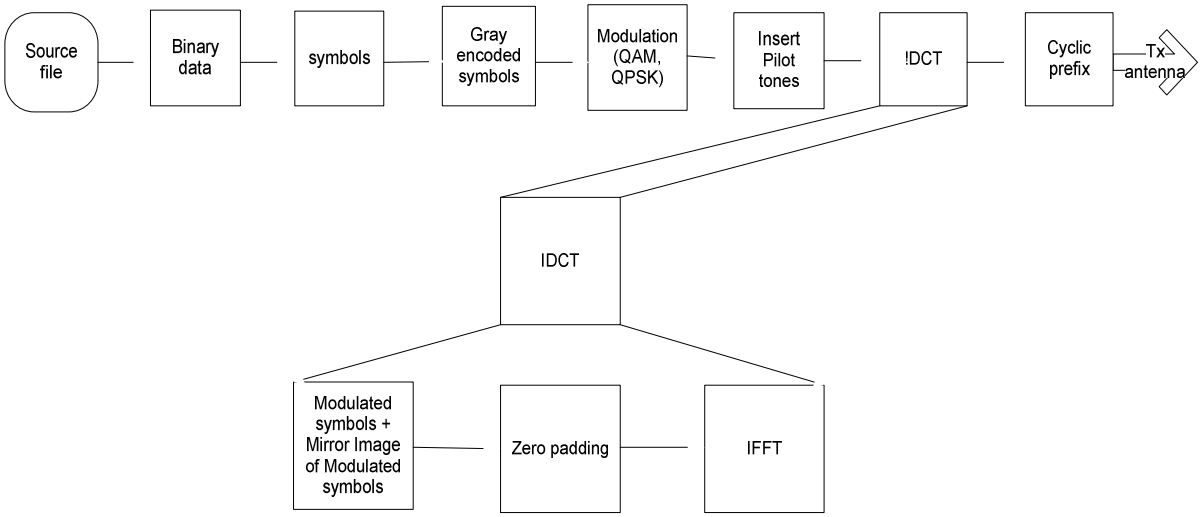
$$x(n) = \frac{1}{\sqrt{N}}\sum_{k=0}^{N} X(k)e^{\frac{j2pkn}{N}} \qquad (2.14)$$

2.5 The incoherent scheme

So far, we have been generating symbols according to a coherent scheme. This scheme requires the receiver to compensate for the phase of the transmitted signals in order to make the correct decisions on the received symbols. The incoherent system, however, does not. The incoherent scheme uses symbols that are 0 or 1. In other words, it transmits energy (1) or stays quiet (0), depending on the binary information to transmit. Therefore, our symbols are now 0's and 1's with the disadvantage of a reduced Euclidian distance between the symbols in the constellation.

3 Transmitter

3.1 Block Diagram of the Transmitter



3.3 Gray Coding-

Gray coding is an efficient way to reduce the probability of bit error for a given Symbol Error. It is based on the fact that the most likely symbol errors result in one bit error only. It is an important code and is used for its simplicity in almost all communication technologies. In pure binary coding or 8421 BCD, counting from 7 (0,1,1,1) to 8 (1,0,0,0) requires 4 bits to be changed simultaneously. If this does not happen then the number is decoded wrongly.

Gray coding avoids this since only one bit changes between subsequent numbers. To construct the code there are two simple rules. First start with all 0s and then proceed by changing the least significant bit (lsb) which will bring about a new state. So instead of representing 8 by 1000 we would represent it by 0110 or 0101 or 0011 etc, taking care to see that the transition from 7 to 8 results in only 1 bit change.

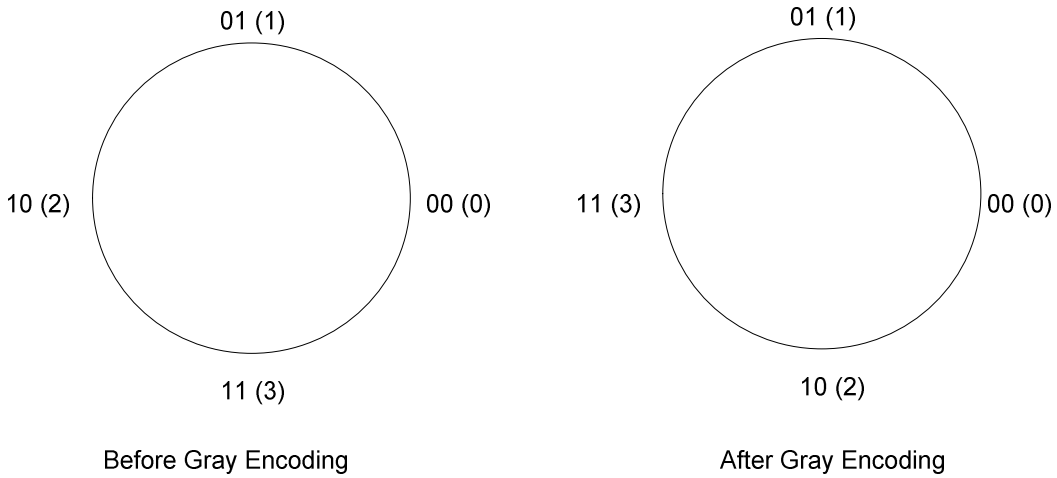
The first 4 Gray coded numbers are indicated below.

Decimal	**Gray Code**
0	00
1	01
2	11
3	10

So referring to figure 3.1 for as we travel along the circle the binary representation of the number changes by only one bit.

3.3.1 Pseudo code for gray encoding-

Bellow is one of the most efficient way to get the gray permutation of sequence of BCD numbers in MATLAB. The implementation is recursive in the sense that we start with a zero element array and go on appending it with its flipped version + successive powers of 2.



Gray Encoding for 4 QPSK

4 Receiver

The receiver side undoes what the transmitter did to the data, but also takes care of the distortion induced by the channel. The signal we received is shown on figure 4.1. There are five major steps in the receiver:

1. The synchronization,
2. The acknowledgement of the header information,
3. The demodulation and the channel estimation,
4. The decision,
5. The decoder.

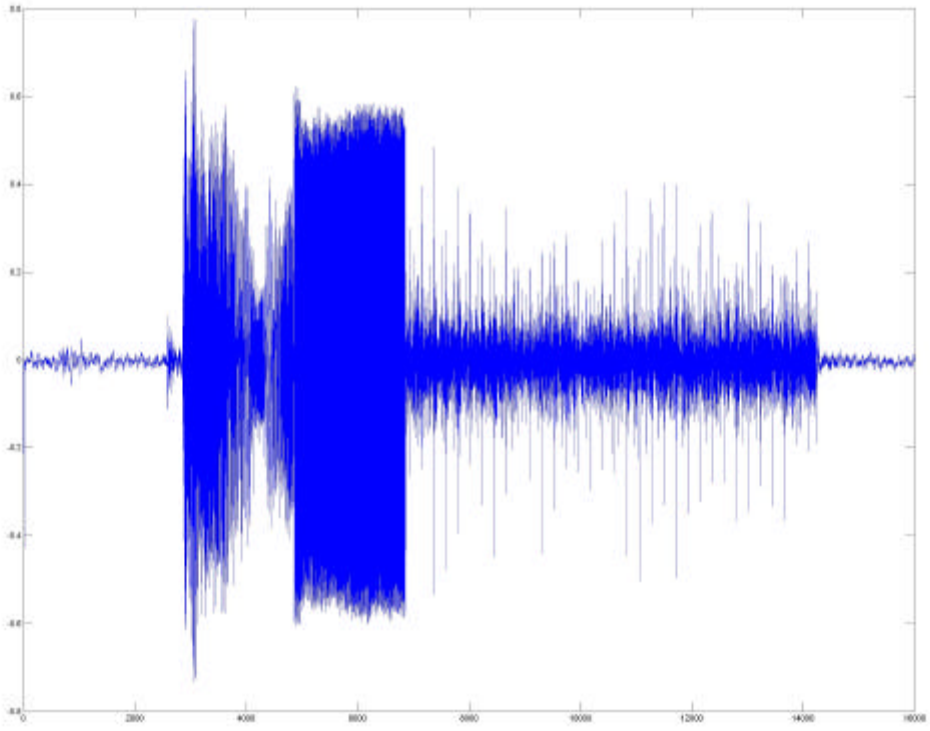
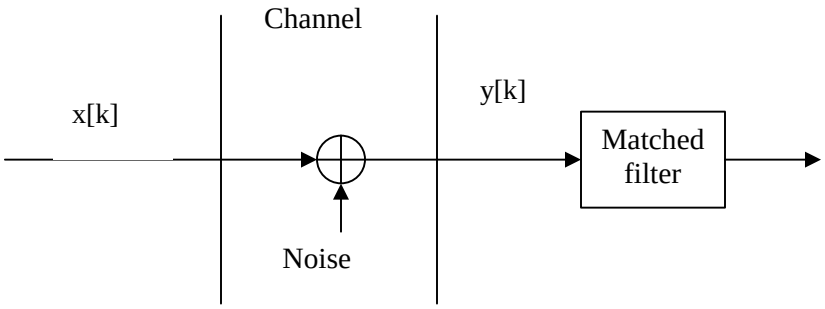


Figure 4.1: Received signal. We can see the three different parts of our signal, appearing in the following order: the synchronizing sequence, the header (approximately flat part) and the actual information data.

4.1 The synchronization.

Since we are recording the data using a microphone, we do not know what delay the channel generated, it is to say where the signal starts in our received file. Thus we need to synchronize our file in order to be able to then apply our different steps. The synchronization is basically a correlation of our signal with a known sequence. The receiver knows that this sequence is in the signal and knows its position in the message; in our case, it is situated at the start of the message relative to the information data. Since we are correlating two similar sequences, we are dealing with autocorrelation more than just correlation.

Consider the simplified communication system:



The correlation formula is:

$$x[k]*x[-k]=\sum_{n=0}^{N}x[n].x[n+k]\approx NE\{x[n].x[n+k]\}=Nd[k]$$

with the variance being var(c[k])=1 and c[k] being the uncorrelated synchronization sequence.

Then, in case of an unknown delay d, we have:

$$y[k]=x[k-d]+w[k]$$

Thus the autocorrelation formula leads to:

$$z[k]=y[k]*x[N-k]=\sum_{n=0}^{N}y[k-n].x[N-n]=\sum_{n=0}^{N}x[k-n+d].x[N-n]+\sum_{n=0}^{N}w[k-n].x[N-n]$$

The matched filter is of finite length and causal. Hence we have: *x*[N-k].

Changing the variable n in n’ = N-n and k-n+d = k+n’-N+d, we have:

$$z[k] = \sum_{n'=0}^{N} x[n'].x[k+n'-N+d] + \sum_{n=0}^{N} w[k-n].x[N-n]$$

$$z[k] = Rxx[k-N+d] + \sum_{n=0}^{N} w[k-n].x[N-n] \approx Nd[k-N+d] + z[k]$$

where:

$Rxx[k] = \sum_{n=0}^{N} x[n].x[n+k]$ is the autocorrelation of the synchronizing sequence.

We will assume that w[n] is an independent identically distributed Gaussian random variable with mean zero and variance σ^2. Thus ζ[k] is also Gaussian but with mean zero and variance σ_ζ^2.

$$s_z^2 = E\{|z[k]|^2\} = E\{\left|\sum_{n=0}^{N} w[k-n].x[N-n]\right|^2\} = \sum_{n,n'=0}^{N} x[N-n].x[N-n']E\{w[k-n].w[k-n']\}$$

$$s_z^2 = \sum_{n,n'=0}^{N} x[N-n].x[N-n'].d[n-n']s^2 = \sum_{n=0}^{N} x[N-n]^2 s^2 = Ns^2$$

Thus z[k] has two components of O(N). To detect correctly the message, we are considering the probability of false alarm: P_{fa} = P(x>T) = Q(T/σ) where σ is the square root of the noise variance, and T is a threshold which also works for the probability of detection: $P_{detection}$ = Q((T-N)/ σ). We are willing to use the constant false alarm rate loop algorithm (CFAR): we set $P_{fa} = 10^{-9}$ and we adapt the threshold considering an adaptive σ:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=0}^{N} x_i^2} = \sqrt{s_N^2 \frac{N-1}{N} + \frac{x_N^2}{N}}$$

We can then get the threshold using the following:

$$P_{fa} = 10^{-m} = Q\left(\frac{T-\max}{s}\right) < e^{-\frac{(T-\max)^2}{s^2}}$$

$$-\log P_{fa} < \frac{(T-\max)^2}{s^2}$$

$$T \geq \max + s\sqrt{-\log P_{fa}}$$

Thus we can choose T equal to $\max + s\sqrt{-\log P_{fa}}$.

$-\log P_{fa}$ is a positive number since the probability of false error is less than 1.

We also have to check that the probability of detection is closed to one:

$$P_{\det ection} = Q\left(\frac{T-N}{s}\right)$$

To maximize the detection of the start of the synchronizing sequence, we choose to use a CHIRP sequence. It generates a sine wave with linearly increasing frequency:

```
% Matlab generation of the synchronizing sequence we use:
sync_seq_len= 2000;
a = 0.2;
mu = 0.1;
t = [0:sync_seq_len-1]';
sync_seq=0.45*sin(2*pi*mu/sync_seq_len*t.^2);
```

This sequence is represented in figure 4.2.

The autocorrelation of this sequence is particularly interesting: see figure 4.3a and 4.3b.

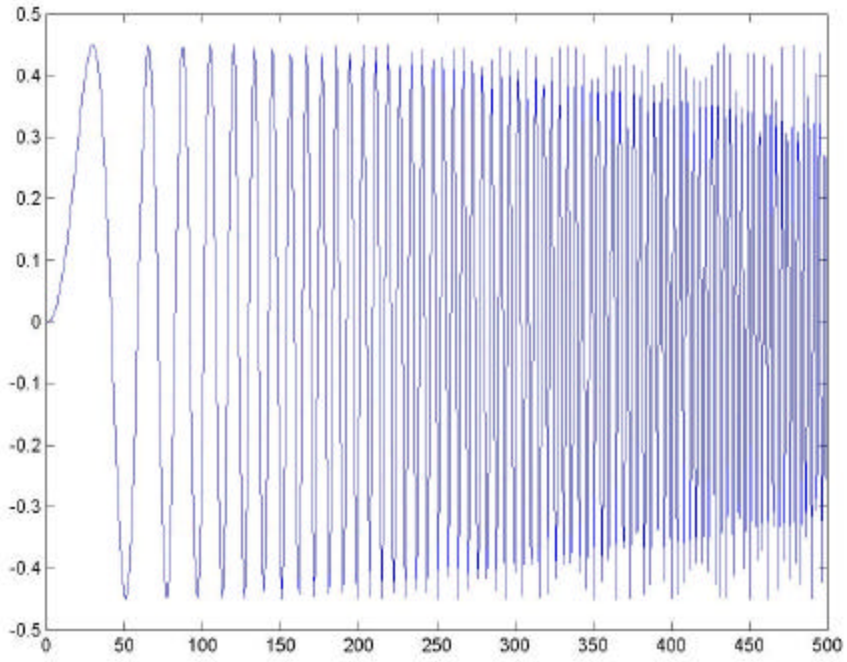


Figure 4.2: Example of synchronizing sequence

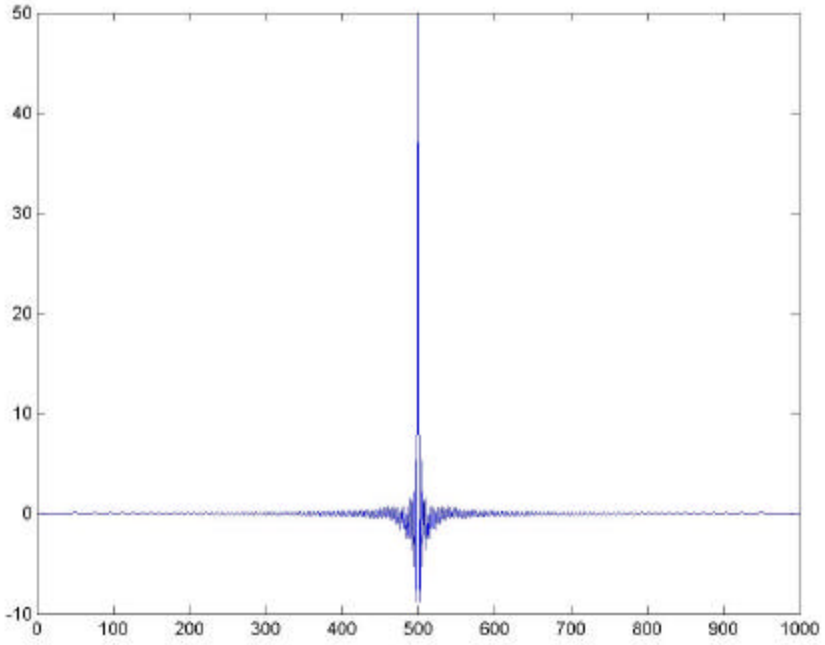


Figure 4.3a: autocorrelation of the chirp synchronizing sequence

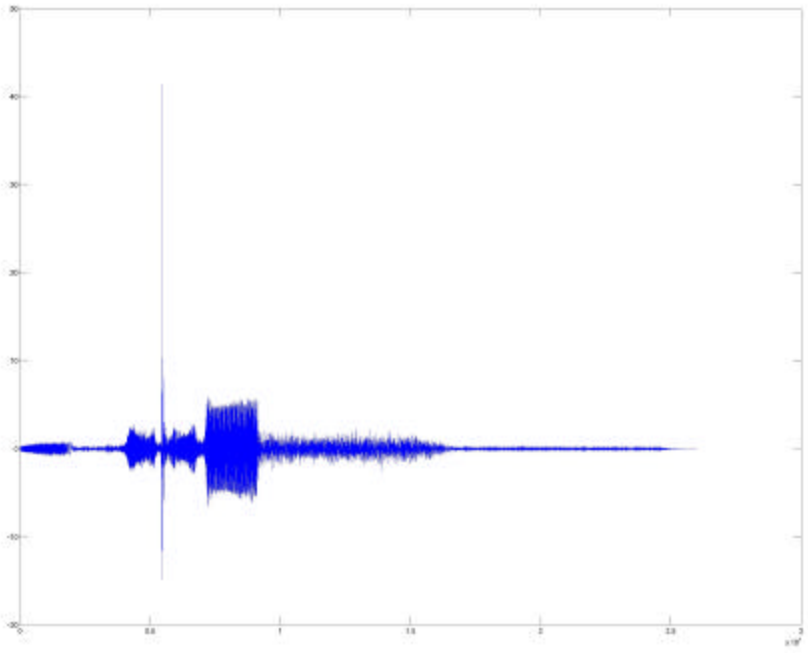


Figure 4.3b Example of correlation for the synchronization in the experiment.

We can see that the peak is very clear: the starting point of the sequence cannot be missed much. It might not be perfect due to the channel distortion: if the channel has two close peaks, then the correlation may give us also two peaks, thus mistaking the start of the sequence. However, in our experiments, we were never off by more than ten samples, which is in the order of the channel length.

4.2 The header specification and information

Once we know the start of the signal in the received file, we can continue our processing.

We also introduced a header to adapt the transmission parameters and notify their values to the receiver, so that the receiver can adapt itself to different rates. Specifically, a header contained the following parameters: Ns (packet size) and M (constellation size). The headers are simple sine waves at different frequencies. At the receiver side, we correlate the received information with the different sine waves that can be in our header. The one that gives the maximum value is the correct one (this is the Maximum Likelihood detection in additive white Gaussian noise).

The correlation of two sine waves of different frequencies gives us:

$$A = \int_0^K \cos(2pf_i t)\cdot\cos(2pf_j t)dt = \int_0^K \frac{1}{2}(\cos(2p(f_i+f_j)t)+\cos(2p(f_i-f_j)t))dt$$

$$= \frac{1}{4p(f_i+f_j)}\sin(2p(f_i+f_j)K)+\frac{1}{4p(f_i-f_j)}\sin(2p(f_i-f_j)K)$$

$$|A| \le \frac{1}{4p}\left(\frac{1}{f_i+f_j}+\frac{1}{f_i-f_j}\right) = \frac{f_i}{2(f_i+f_j)(f_i-f_j)}$$

K is the length of the signal.

When the two frequencies are equal, we have:

$$A = \int_0^K \cos(2pf_i t)^2 dt = \int_0^K \left(\frac{1}{2}+\frac{1}{2}\cos(4pf_i t)\right)dt$$

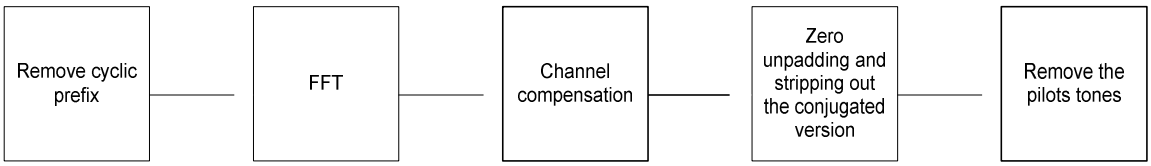
$$= \frac{K}{2}+\frac{1}{8pf_i}\sin(4pf_i K) \ge \frac{K-1}{2}$$

Thus, we clearly see that we have a maximum when the two frequencies are equal. This is how we can decide at the receiver which frequency was send by the transmitter, and thus which parameters are used to code the information.

4.3 The demodulation

4.3.1 Received signal

At this point, we need to get to the start of the actual message. Knowing the start of the synchronizing sequence, its length and the length of the header, we know where our message starts. However, to be sure we get all the data, we go back by a few samples. Since we are going back by a few samples in the time domain, after the Fourier transform, there will be no loss of carriers. At this point, we have the following steps:



$$u[n] = x[n] * h[n] = \sum_{l=0}^{M-1} x[n-l]h[l]$$

$$= \sum_{l=0}^{M-1}\left[h[l]\sum_{k=0}^{N-1} S[k]e^{j\frac{2p(n-l)k}{N}} \right]$$

$$= \sum_{k=0}^{N-1} S[k]e^{j\frac{2pnk}{N}} \sum_{l=0}^{M-1} h[l]e^{-j\frac{2pkl}{N}}$$

$$= \sum_{k=0}^{N-1} S[k]e^{j\frac{2pnk}{N}} H\left(e^{j\frac{2pk}{N}}\right)$$

$$y[n] = \sum_{k=0}^{N-1} S[k]e^{j\frac{2pnk}{N}} H\left(e^{j\frac{2pk}{N}}\right) + v[n]$$

$$R[k'] = \sum_{n=0}^{N-1} y[n]e^{-j\frac{2pk'n}{N}} = \sum_{n=0}^{N-1}\left[\sum_{k=0}^{N-1} S[k]e^{j\frac{2pkn}{N}} H\left(e^{j\frac{2pk}{N}}\right) + v[n]\right]e^{-j\frac{2pk'n}{N}}$$

$$= \sum_{k=0}^{N-1} S[k]H\left(e^{j\frac{2pk}{N}}\right)\sum_{n=0}^{N-1}\left(e^{j\frac{2p(k-k')n}{N}} + v[n]e^{-j\frac{2pk'n}{N}}\right)$$

$$= \sum_{k=0}^{N-1} S[k]H\left(e^{j\frac{2pk}{N}}\right)d(k-k') + \sum_{n=0}^{N-1} v[n]e^{-j\frac{2pk'n}{N}}$$

If the delay induces by the channel is not equal to 0, we have:

$$R[k'] = \left(S[k']H\left(e^{j\frac{2pk'}{N}}\right) + \sum_{n=0}^{N-1} v[n]e^{-j\frac{2pk'n}{N}}\right)e^{-j\frac{2pd}{N}}$$

Thus, when we compensate for the phase shift due to d:

$$R'[k'] = S[k']e^{-j\frac{2p(d+Lk')}{N}} H\left(e^{j\frac{2pk'}{N}}\right) + noise\ \ component \qquad \text{Equation. 4.1}$$

Since theoretically d+Lk' = 0, we are left, with just the Fourier transform of the channel taken at frequency k'/N. This is why we need this expression to compensate for the channel distortion. Once we have it, we can divide R' by H at the appropriate frequency and get back S corrupted with some noise: the estimated message is $\hat{S}$.

$$\hat{S}[k'] = S[k'] + noise\ \ component$$

This proved the need of channel estimation, in the case of a coherent detection.

4.3.2 Channel estimation

We want through several ways to estimate the channel. Our first assumptions were that the channel gave linear distortion, was time invariant and that the noise could be considered to be white Gaussian noise.

The different criteria for the choice of the estimation scheme were:

- Efficiency in the computation,
- Efficiency in the estimation,
- Accuracy in the results (scattering plots talked about later on).

4.3.2.1 Using the synchronizing sequence

We assumed that the channel was linear time invariant. Since we already had the synchronizing sequence, we used it to estimate the channel as explained below.

Let us have the following:

- y is the received signal,
- h is the channel impulse response of length L,
- x is the original message of length N,
- w is the noise.

$$y(0) = h(0)x(0) + w(0)$$
$$y(1) = h(0)x(1) + h(1)x(0) + w(1)$$
$$y(2) = h(0)x(2) + h(1)x(1) + h(2)x(0) + w(2)$$

And so on until:

$$y(N-1) = \sum_{i=0}^{L-1} h(N-1-i)x(i) + w(N-1)$$

Thus we can see that we can put this in matrix form:

$$[y(0).....y(N)] = [h(0).....h(L-1)]\begin{bmatrix} x(0) & x(1) & x(2)....x(N-1) \\ 0 & x(0) & x(1).....x(N-2) \\ . & & \\ 0.....0 & & x(0) \end{bmatrix}$$

$$\underline{y} = \underline{h}\underline{X}$$

where y is a row vector of length N, h is a row vector of length L and X Is a matrix of size L x N.

The formula gives us:

$$\underline{y}\underline{X}' = \underline{h}\underline{X}\,\underline{X}'$$

$$\underline{y}\underline{X}'(\underline{X}\,\underline{X}')^{-1} = \underline{h}$$

So we can have h using the pseudo-inverse of X from y. And if we have a delay induce by the channel, the vector y and X are just shifted by d, the delay amount, for their first index. Then, using a minimum mean square error (MMSE) scheme, we can get the best estimation for h in term of length and values.

$$\hat{\underline{h}} = \arg\max_{\underline{h}} \left\| \underline{y} - \underline{h}\underline{X} \right\|^2$$

This scheme gave us good values, but only for L > 250 (see figure 4.4). Thus the computation was long: it was not an efficient scheme for computation. We also noticed that it was not accurate in the presence of noise. The reason of this problem explained by using the properties of the synchronizing sequence:

$$\underline{R_c} \hat{=} \underline{T(c)}^T \underline{T(c)}$$

$$\underline{T(c)} = \begin{bmatrix} c(0) & 0 \ldots\ldots\ldots\ldots\ldots 0 \\ c(1) & c(0) \quad 0 \ldots\ldots 0 \\ | & \ldots\ldots\ldots \backslash \\ | & \ldots\ldots\ldots\ldots\ldots c(0) \\ | & \ldots\ldots\ldots\ldots\ldots\ldots\ldots | \\ c(N-1) & \ldots\ldots\ldots\ldots | \\ 0 & c(N-1) \ldots\ldots\ldots | \\ | & \ldots\ldots\ldots \backslash \ldots\ldots\ldots | \\ | & \ldots\ldots\ldots\ldots\ldots c(N-1) \end{bmatrix}$$

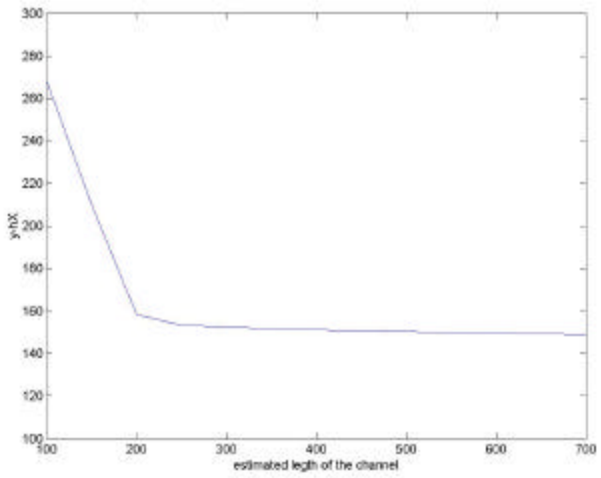


Figure 4.4: error of estimation versus the estimated length.

where

$$\underline{y} = \underline{T(c)}\underline{h} + \underline{w}$$

$\underline{T(c)}$ is a Sylvester matrix:

$$\left(1 \;\; e^{jw} \;.....\; e^{jw(N+L-1)}\right)\underline{T(c)} = C\left(e^{jw}\right)\left(1 \;\; e^{jw} \;.....\; e^{jLw}\right)$$

where $C\left(e^{jw}\right)$ is the Fourier series of c[n] for e^{jw}.

Since T($\underline{c}$) is a Sylvester matrix, for L large (length of the impulse response $\underline{h}$ of the channel):

$$\underline{R} \approx U.S.U^{H}$$

with U being the Fourier transform matrix of size L x L, S being the diagonal matrix with main diagonal $\left[S_{cc}(1).....S_{cc}\left(e^{-j\frac{2p(L-1)}{L}}\right)\right]$. Since the training sequence does not have low frequency components or high frequency components, the corresponding eigenvalues are zero and the matrix is singular. If we had to reconstruct the channel exactly, that would have been a problem. However, we want to evaluate the frequency response of the channel only at the subcarrier frequencies used (where we actually transmit). Suppose we have the frequencies m/M for $m = m_1,\ldots,m_2$.

Let us define $\left\{U_{NpxN}\right\}_{m,n} = \left\{e^{-j\frac{2pmn}{N}}\right\}$ for $m = m_1,\ldots,m_2$ and n = 0,…,N+L-1. Then:

$$\underline{Y} \hat{=} U_{NpxN}.\underline{y} = U_{NpxN}\underline{T(c)}\underline{h} + U_{NpxN}.\underline{w}$$
$$= diag\left(C\left(e^{-j\frac{2pm_1}{M}}\right),.....,C\left(e^{-j\frac{2pm_2}{M}}\right)\right)U_{NpxN}.\underline{h} + \underline{W}$$

What we are really interested in is:

$$\underline{H} = \left(H\left(e^{-j\frac{2pm_1}{M}}\right),.....,H\left(e^{-j\frac{2pm_2}{M}}\right)\right) = U_{NpxN}.\underline{h}$$

Thus:

$$\underline{Y} = diag\left(C\left(e^{-j\frac{2pm_1}{M}}\right),.....,C\left(e^{-j\frac{2pm_2}{M}}\right)\right).\underline{H} + \underline{W}$$
$$= diag(\underline{C}).\underline{H} + \underline{W} = \underline{C} \bullet \underline{H} + \underline{W}$$

where $\bullet$ is the dot product and $\underline{C} = U_{NpxN}.\underline{c}$

If the noise $\underline{w}$ has variance s^2 then the entries of $\underline{W}$ have variance $s^2.\left\{U_{NpxN}U_{NpxN}^{H}\right\} = s^2 N$.

Hence:

$$\hat{H}_{mmse} = G \bullet Y \quad \text{with} \, G = \frac{\underline{C}^*}{\left(\underline{C}^* \bullet \underline{C} + s^2 N\right)}$$

With this scheme, we had an efficient process in term of computation on and in term of estimation.

However, the results were not as good as expected when looking at the scattering diagrams. These diagrams are the plots of the symbols before and after the channel compensation (to see the results of this compensation). Before the channel compensation, we can see a phase shift in the symbols: they appear in a circle (see figure 4.5a).

After compensation, the eye is not open: see figure 4.5b.

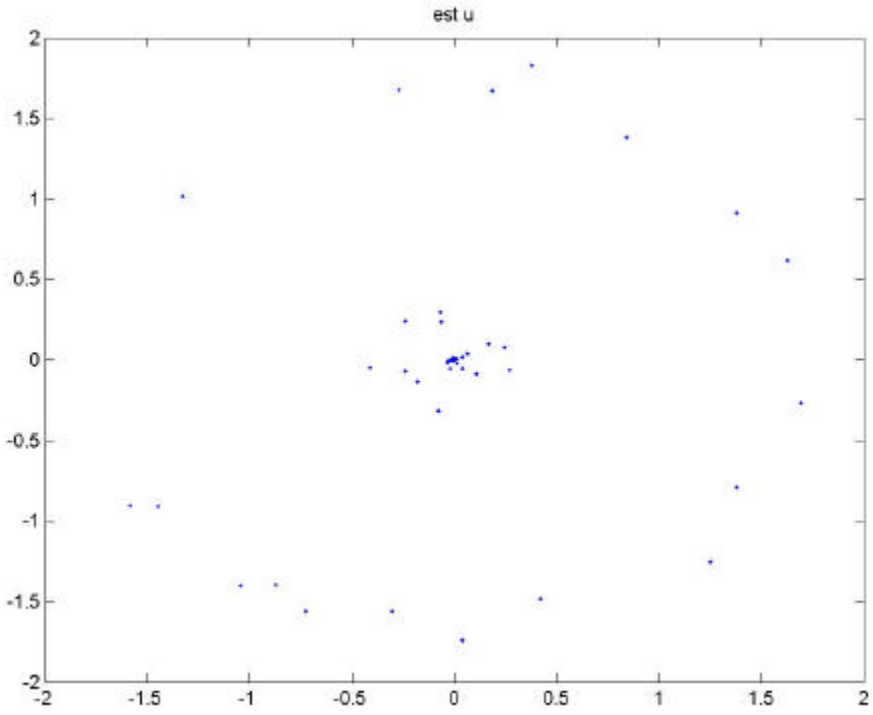


Figure 4.5a: Demodulated signal for one carrier before compensation by the channel estimation

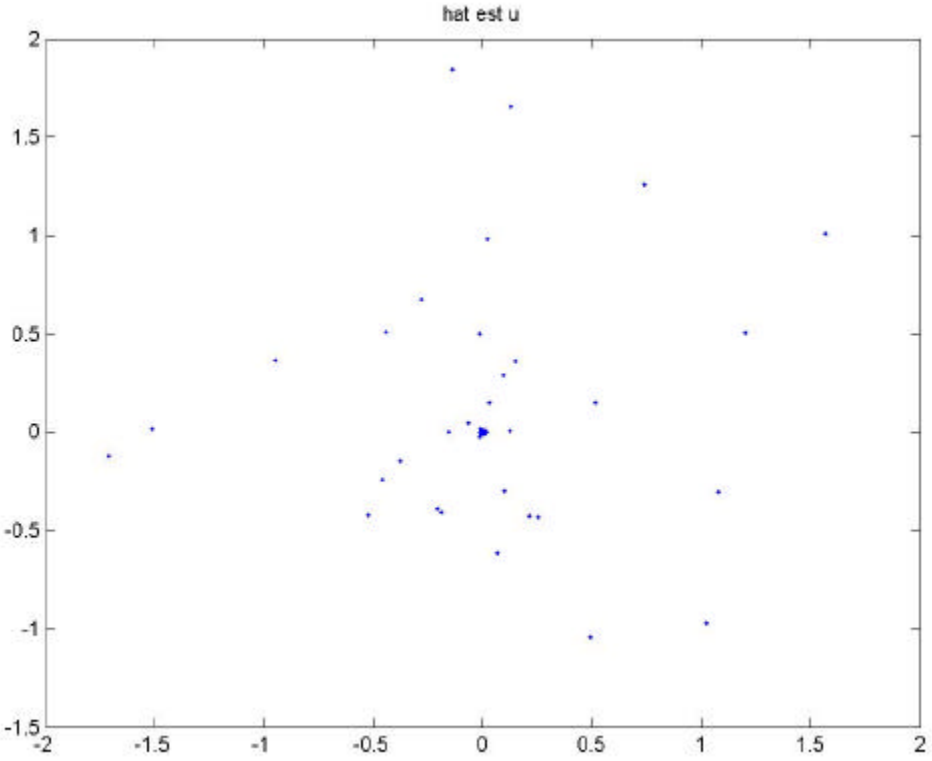


Figure 4.5b: Demodulated and compensated signal for the same carrier: the eye is closed.

4.3.2.2 Using pilot tones

As a result of this, we looked back at our assumptions and thought about the possibility of a time varying channel. However, if the blocks are small enough, for one block, we can assume that the channel is time invariant. This is why we oriented our design towards one channel estimation per block using pilot tones. The scheme when using pilot tones is the following: the pilot tones are symbols of value one added before the ifft at the transmitter. Thus, at the receiver, after the fft, we have at the index of the pilot tones an estimation of the frequency response of the channel at the corresponding frequencies (recall equation 4.1 of the received signal and take it at the pilot tones indexes). Mathematically, we have:

$$H(f_i) = \sum_{n=0}^{L-1} h[n] e^{j\frac{2pn}{L} f_i}$$

$$\underline{H} = \underline{A}.\underline{h}$$

$$\underline{A} = \left[e^{-j\frac{2pn}{L} f_i} \right]_{i,n} \quad and \quad \underline{h} = [h(0),....,h(L-1)]^T$$

for $\underline{A}$, it goes from 1 to the number of pilots we put per block; n = 0,…,L-1, where L is the estimated length of the channel.

Thus we can estimate $\underline{h}$ at these frequencies by $\underline{h} = pinv(\underline{A}).\underline{H}$ where pinv($\underline{A}$) is the pseudo-inverse of the matrix $\underline{A}$, which may not be square. Then, by taking the fft of this estimation of h for all the Np frequencies we used to transmit (interpolating the values between the pilot tones), we have a good estimation of the frequency response of the channel for each block and we can compensate it.

At the transmitter, the pilot tones are placed as uniformly as possible so when we interpolate at the receiver, the interpolation is the best possible everywhere. Even if we are not transmitting at very low frequencies or very high frequencies, pilots are placed there so that the interpolation at the edge of the transmitting frequencies is also the best possible. We also took care of the middle of the blocks (since we flip the data to have a real signal).

This scheme gave us excellent results in the estimation of the channel, in the efficiency of the computation and in the results observed in the scattering diagrams (see figures 4.6a and 4.6b). It is the one we choose to implement in the final version of the coherent modem (the incoherent modem does not need such a thorough estimation of the channel).

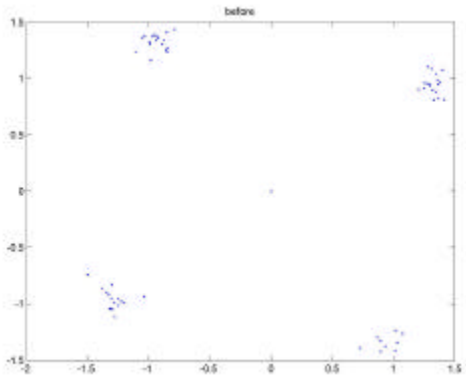


Figure 4.6a: Scatter plot for one carrier before the compensation by the channel (simulation)

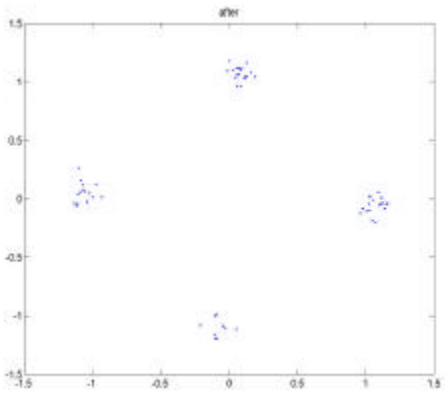


Figure 4.6b: Same carrier but after the channel compensation. We can see that the clusters are centered around the correct point: (1,0), (0,j), (-1,0), (0,-j).

4.3.2.3 Conclusion and future work for the channel estimation

We noticed that the results were decreasing in accuracy at the end of the message. Also, with a longer message, the last blocks were completely off. This lead to the conclusion that the channel is not intrinsically time variant, but really time invariant with a loss of samples that makes the blocks shift and look like a time varying channel. This loss of samples was observed by comparing the channel impulse responses estimates via the pilot tones plotting the channel estimation for each block (see figure 4.7). There is a clear shift of the peaks of the impulse response towards the beginning of the block. But these peaks, regardless of their position in the block are similar in amplitude. Hence the impulse response looks less time varying then shifting in the block by a loss of samples.

This loss of samples could be explained by a difference of sampling frequency between our two audio functions wavrecord and wavplay. Since we are using two different computers, the audio cards and software may be slightly different, but enough so that the effect is evident in our experiments. The D/A converter is slower than the A/D converter. Thus, the signal lasts for T+e and is sampled with a rate of Fs at the transmitter. However, the receiver samples a message lasting T at Fs. Hence, when we sample at the receiver, we take more sample than we should, giving the impression of a time varying channel. This sampling problem can be resolved by first experimentally determining how much longer the message is lasting (what is the value for e). We should then sample for T+ e at Fs at the receiver then interpolating back at T before processing the block by the channel estimation. This could be verified using pilot tones at first then the LTI scheme described before. Another solution would be to sample at Fs for T at the receiver, but taking the blocks by more than T'>T+prefix_length each time. T' would be estimated experimentally and linearly increasing with the number of blocks.

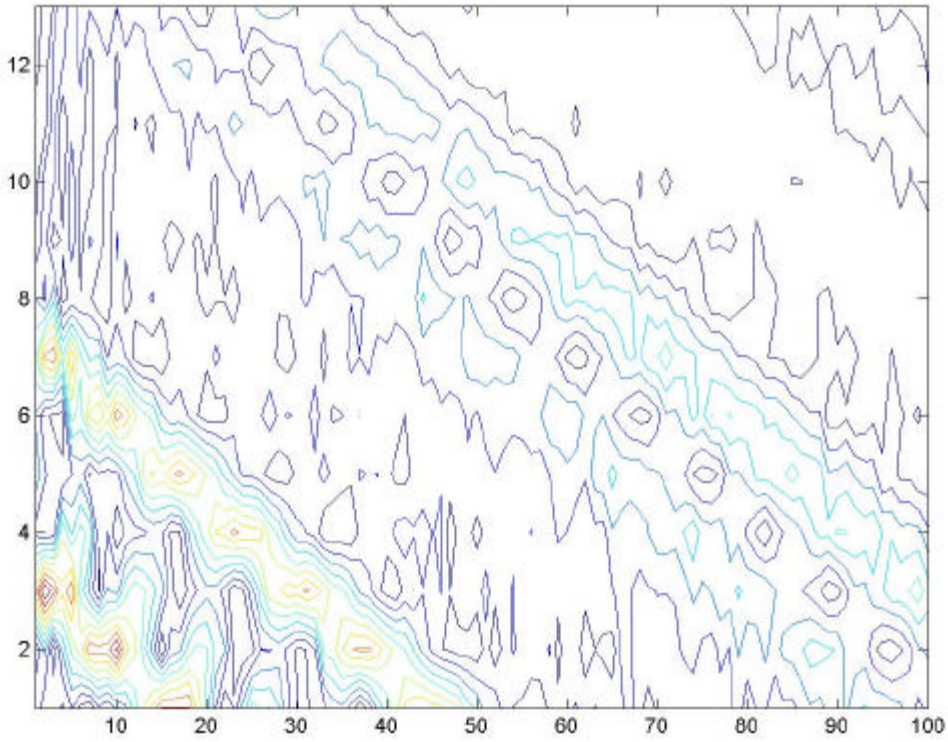


Figure 4.7: plot of the channel estimation impulse. This plot shows the peaks and different levels of the impulse response (y axis) for each block (x axis).

4.4 Optimum detection and performance of the systems

As described earlier, the transmitter sends digital information by the use of C signal waveforms $\{\tilde{s}_m(t), m = 1,2,....,C\}$. Under white Gaussian noise, the received signal is corrupted uncorrelated samples from a normal distribution. Thus, the received signal is:

$\tilde{r}(t) = h\tilde{s}_m(t) + n(t)$ where $h = \mathrm{a}e^{j\mathrm{f}}$ is the gain due to fading and $n(t)$ denotes a sample function of the additive white Gaussian noise. The fading phenomenon appears as changes in magnitude and phase of the transmitted signal. This is due to changes in the scattering environment. The vector representation of the previous is the following:

$$\tilde{\mathbf{r}} = (\tilde{r}\,0, \tilde{r}\,1, ..., \tilde{r}\,N-1)) \qquad (4.1)$$

$$\tilde{\mathbf{s}}i = (\tilde{s}\,i_0, \tilde{s}\,i1,..., \tilde{s}\,iN-1) \quad (4.2)$$

$$\tilde{\mathbf{n}} = (\tilde{n}\,0, \tilde{n}\,1,..., \tilde{n}\,N-1) \quad (4.3)$$

Let us now describe the optimum decision rule based on the observation vector **r**. The decision is made towards the vector message $\tilde{\mathbf{s}}_i$ in such a way that maximizes the a posteriori probability $\mathrm{P}(h\tilde{\mathbf{s}}_i/\tilde{\mathbf{r}})$. Then if the system decides in favor of another vector $\tilde{\mathbf{s}}_m$, it is clear that the probability of erroneous decision is:

$$\mathrm{P}_{e/\tilde{\mathbf{r}}} = 1-\mathrm{P}(h\tilde{\mathbf{s}}_m/\tilde{\mathbf{r}}) \quad (4.4)[1]$$

This is called a MAP (maximum a posteriori probability) because its nature minimizes the decision error probability. Then, by the use of Bayes theorem we have:

$$\mathrm{P}(h\tilde{\mathbf{s}}_m/\tilde{\mathbf{r}}) = \frac{p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_m)\mathrm{P}_m}{p(\tilde{\mathbf{r}})}, m = 0,....,M-1 \quad (4.5)[1]$$

In the equation above, $p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_m)$ is the joint conditional probability density function of $\tilde{\mathbf{r}}$ given the $\tilde{\mathbf{s}}_m$. P_m is the prior message probability which are equal most of the times and therefore are of no interest to us. This in turn, leads us to the Maximum Likelihood (ML) receiver rule as follows:

choose $\tilde{\mathbf{s}}_m$ if $p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_m) \geq p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_k) \forall k \neq m$ (4.6)[1]

In order to define $p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_m)$, we need to recall that $\tilde{\mathbf{r}} = h\tilde{\mathbf{s}}_m + \tilde{\mathbf{n}}$ where $\tilde{\mathbf{n}}$ has the multivariate Gaussian probability density function:

$$p(\tilde{\mathbf{n}}) = \frac{1}{(2\mathrm{p}No)^N} e^{\left(\frac{1}{2No}\|\tilde{\mathbf{n}}\|^2\right)} \quad (4.7)[1]$$

Then, we replace $\tilde{\mathbf{n}} = \tilde{\mathbf{r}} - h\tilde{\mathbf{s}}_m$ into the equation above, leading to:

$$p(\tilde{\mathbf{r}}/h\tilde{\mathbf{s}}_m) = \frac{1}{(2\mathrm{p}No)^N} e^{\left(\frac{1}{2No}\|\tilde{\mathbf{r}} - h\tilde{\mathbf{s}}_m\|^2\right)} \quad (4.8)$$

We recognize however, that $\tilde{\mathbf{s}}_m$ that maximizes the previous also maximizes the metric $metric = -\|\tilde{\mathbf{r}} - h\tilde{\mathbf{s}}_m\|^2$. This means that the ML chooses the vector that is closest in squared Euclidean distance to the received vector $\tilde{\mathbf{r}}$.

To compute this we need to define:

$$R_m = \{\tilde{\mathbf{r}} : \|\tilde{\mathbf{r}} - h\tilde{\mathbf{s}}_m\|^2 \le \|\tilde{\mathbf{r}} - h\tilde{\mathbf{s}}_k\|^2, \forall k \neq m\} \quad (4.9)[1]$$

which is the decision region around each of the signal points $h\tilde{\mathbf{s}}_m$. The ML decision rule becomes:

Choose $\tilde{\mathbf{s}}_m$ *if* $\tilde{\mathbf{r}} \in R_m$. (4.10)

4.4.1 Pair wise error probability

A special case of consideration is the pair wise error probability. In this situation, we consider two of the M signal vectors. These two vectors $\tilde{\mathbf{s}}_j$ and $\tilde{\mathbf{s}}_i$ are separated at the receiver by $\|h\tilde{\mathbf{s}}_j - h\tilde{\mathbf{s}}_i\|^2$ which is the squared Euclidian Distance. Since we have assumed AWGN noise, we know that it this noise has zero mean and variance No. The decision boundary established between these two vectors is pictured in Fig. 4.1.

In the case where vector $\tilde{\mathbf{s}}_i$ is sent, the probability of error is the event that the noise is big enough to force the received vector $\tilde{\mathbf{r}} = \tilde{\mathbf{s}}_i + \tilde{\mathbf{n}}$ to cross the decision boundary. Thus, this probability of error is given by:

$$\mathrm{P}(e/\tilde{\mathbf{s}}\,i) = \mathrm{P}(e/\tilde{\mathbf{s}}\,j) = Q\left(\sqrt{\frac{a^2 d_{ji}^2}{4No}}\right) \quad (4.11)$$

and $d_{ji}^2 = \| \tilde{\mathbf{s}}\, i - \tilde{\mathbf{s}}\, j \|^2$ is the squared Euclidian distance between the two vectors $\tilde{\mathbf{s}}\, i$ and $\tilde{\mathbf{s}}\, j$. With this in mind, we can proceed to evaluate the performance of a PSK and QAM system.

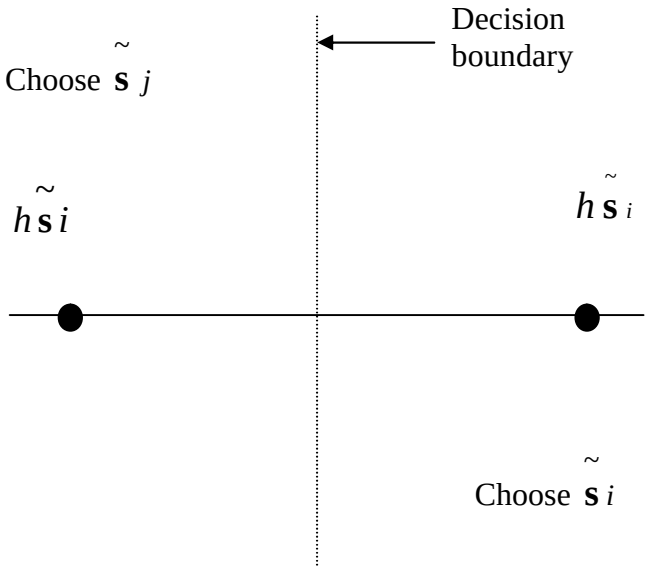


Figure 4.1.

4.4.2 Performance of PSK

The PSK complex signal vectors are:

$$\tilde{sm}(t) = \sqrt{2Ehe}^{jqm}, m = 0,...,C-1 \qquad (4.12)[1]$$

where q is the carrier phase defined as:

$qn = \frac{2p}{M} x_n$ and the data symbols are $x_n = n, n \in \{0,1,...,C-1\}$, with C being the alphabet size. E_g is the energy in the amplitude shaping pulse $Ag(t)$:

$$E_g = \frac{A^2}{2} \int_{-\infty}^{\infty} g^2(t)dt \qquad (4.13)$$

From these definitions, the BPSK complex vectors are:

$$\tilde{so} = \tilde{s1} = \sqrt{2E_g} \tag{4.14}$$

and therefore the Euclidian distance between the vectors is $\tilde{d}_{01} = 2\sqrt{2E_g}$. It is important to recognize that the BPSK system transmits 1 bit/symbol which implies that the symbol energy equals the bit energy. Thus, using (4.11), the probability of bit error for BPSK is:

$$\begin{aligned} Pb(SNRb) &= Q\left(\sqrt{\frac{a^2 \tilde{d}_{01}^2}{4No}}\right) \\ &= Q\left(\sqrt{\frac{a^2 4(2Eg)}{4N_o}}\right) \\ &= Q\left(\sqrt{\frac{2a^2 E_g}{No}}\right) \end{aligned}$$

$$Pb(SNRb) = Q(\sqrt{2SNRb}) \tag{4.15}$$

where $SNRb$ represents the bit energy-to-noise ratio:

$$SNRb = \frac{a^2 E_g}{No} \tag{4.16}$$

Similarly, the probability of error for QPSK is derived as follows:

The signals with $?_0 = \frac{p}{4}$ are mapped into the constellations as:

$$\tilde{\mathbf{s}}_0 = -\tilde{\mathbf{s}}_2 = \sqrt{E_g}(1+j) \tag{4.17}$$

$$\tilde{\mathbf{s}}_1 = -\tilde{\mathbf{s}}_3 = \sqrt{E_g}(-1+j)) \tag{4.18}$$

Figure 4.2 illustrates this mapping.

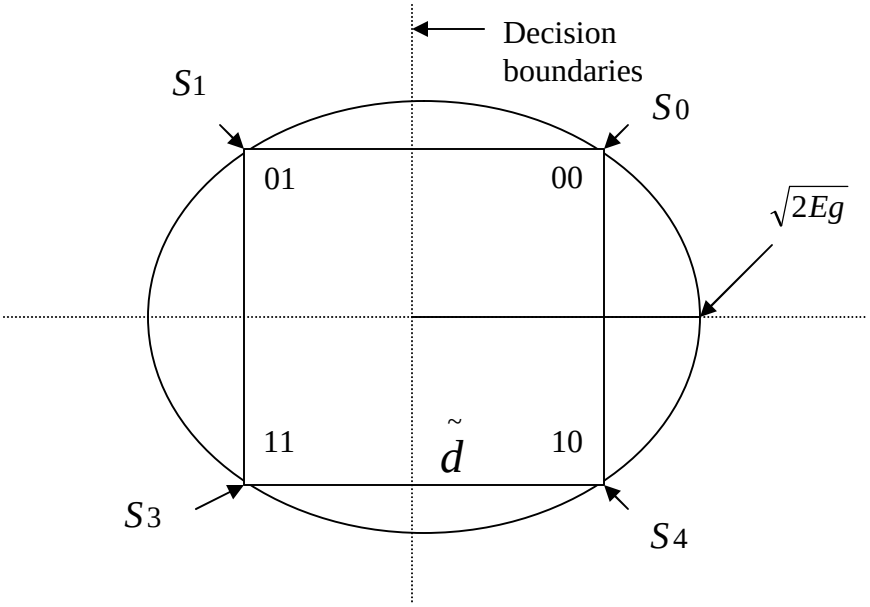


Figure 4.2.

Looking at the Fig 4.2, we realize that the noise added is two dimensional which means that it has an in phase and a Quadrature component. It turns out, that these $\tilde{n}_I$ and $\tilde{n}_Q$ components are independent zero-mean Gaussian random variables with variance No. Using the ML receiver, the probability of symbol error is:

Define Pc as the probability of correct decision for the 2-bit symbol as

$$P_c = (1 - P_b)(1 - P_b) = (1 - P_b)^2 \qquad (4.19)$$

where P_b is the probability of bit error. Since there is no interference between the signals on the two Quadrature carriers, the bit error probability for QPSK is the same as the one for BPSK. Thus:

$$P_c = \left[1 - Q\left(\sqrt{\frac{a^2 2Eb}{No}}\right)\right]^2 \qquad (4.20)$$

and the symbol error probability for QPSK is:

$$P_4 = 1 - P_c$$

$$= 1 - \left[1 - 2Q\left(\sqrt{\frac{a^2 2Eb}{No}}\right) + Q\left(\sqrt{\frac{a^2 2Eb}{No}}\right)^2\right]$$

$$= 2Q\left(\sqrt{\frac{2a^2 Eb}{No}}\right) - Q\left(\sqrt{\frac{2a^2 Eb}{No}}\right)^2$$

$$P_4 = 2Q\left(\sqrt{\frac{2a^2 Eb}{No}}\right)\left[1 - \frac{1}{2}Q\left(\sqrt{\frac{2a^2 Eb}{No}}\right)\right]$$

$$P_4(SNRb) = 2Q(\sqrt{2SNRb})\left[1 - \frac{1}{2}Q(\sqrt{2SNRb})\right] \quad (4.21)$$

and $$SNRb = \frac{a^2 E_g}{No} \quad (4.22)$$

4.4.3 Performance of QAM

In this section, we will consider the Error Probability of QAM. To explain this, we will use the probability of error of PAM signals as a reference. We'll start by looking at Fig 4.3.

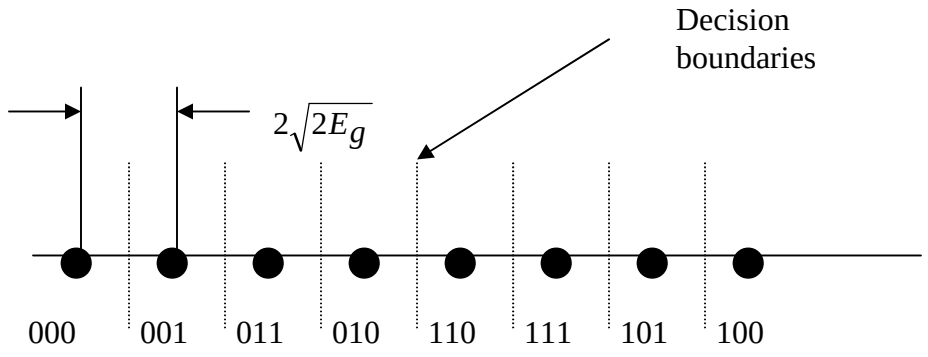


Figure 4.3.

The signal constellation in the figure has $C-2$ points in the inside that have the following probability of symbol error:

$$P_{inner} = Q\left(\sqrt{\frac{2a^2E_g}{No}}\right) + Q\left(\sqrt{\frac{2a^2E_g}{No}}\right) = 2Q\left(\sqrt{\frac{2a^2E_g}{No}}\right) \qquad (4.23)$$

This result comes from the fact that these points in the constellation have two boundaries limiting the symbols and therefore the probability of crossing the boundary due to error is multiplied by two. On the other hand, the 2 outer points have only one boundary and thus, the probability of error in these cases is the following:

$$P_{outer} = Q\left(\sqrt{\frac{2a^2Eg}{No}}\right) \qquad (4.24)$$

Combining these two outcomes, the probability of symbol error is:

$$P_C = \frac{C-2}{C}P_i + \frac{2}{C}P_o$$

$$= \frac{C-2}{C}2Q\left(\sqrt{\frac{2a^2Eg}{No}}\right) + \frac{2}{C}Q\left(\sqrt{\frac{2a^2E_g}{No}}\right)$$

$$P_C = 2\left(\frac{C-1}{C}\right)Q\left(\sqrt{\frac{2a^2E_g}{No}}\right) \qquad (4.25)$$

The next step is to place the average symbol energy into the equation above.

We have $\tilde{s}_i = \sqrt{2E_g}(2i-1-C), i = 1,...,C$

and of course, the energy in the signal is:

$$E_i = \frac{1}{2}(\tilde{s}_m)^2 = E_g(2i-1-C)^2 \qquad (4.26)[1]$$

We can now express the average energy as:

$$E_{average} = Eg\frac{1}{C}\sum_{i=1}^{C}(2i-1-C)^2$$

$$= E_g\frac{1}{C}(4\sum_{i=1}^{C}i^2 - 4(C+1)\sum_{j=1}^{C}j + C(C+1)^2)$$

Using the following identities: $\sum_{i=1}^{C} i^2 = \frac{C(C+1)(2C+1)}{6}$ and $\sum_{j=1}^{C} j = \frac{C(C+1)}{2}$ gives

$$E_{average} = E_g \frac{1}{C}\left(\frac{4C(C+1)(2C+1)}{6} - \frac{4C(C+1)^2}{2} + C(C+1)^2\right)$$

$$= Eg \frac{1}{C}\left(\frac{C(C+1)(4(2C+1) - 12(C+1) + 6(C+1))}{6}\right)$$

$$= E_g \frac{1}{C}\left(\frac{C(C+1)(8C+4-6C-6)}{6}\right)$$

$$= Eg \frac{1}{C}\frac{(C^2+C)(2C-2)}{6}$$

$$E_{average} = E_g \frac{C^2-1}{3} \qquad (4.27)$$

Then, replacing the result above into the probability of symbol error we have:

$$\mathrm{P}_C(SNRs) = 2\left(\frac{C-1}{C}\right)Q\left(\sqrt{\frac{6}{C^2-1}SNRs}\right) \qquad (4.28)[1]$$

and $$SNRs = \frac{a^2 E_{average}}{No} \qquad (4.29)$$

is the average symbol energy-to-noise ratio.

Therefore, for an *C*-QAM constellation of size $C = 4^k$, where *k* is an integer chosen from the examples shown in Fig 4.4, they can be analyzed as two $\sqrt{C}$-PAM systems in Quadrature.

Each of these systems is allocated half the power of the *C*-QAM system. Recalling the previous results for PAM, then the probability of symbol error for each $\sqrt{C}$-PAM system is:

$$P_{\sqrt{C}} = 2\left(\frac{\sqrt{C}-1}{\sqrt{C}}\right)Q\left(\sqrt{\frac{6SNRs}{(C-1)2}}\right) \qquad (4.30)[1]$$

The probability of correctness in the *C*-QAM system is:

$$\mathrm{P}_C = (1 - P_{\sqrt{C}})^2 \qquad (4.31)$$

which leads to the following probability of symbol error:

$$P_C(SNRs) = 1 - (1 - P_{\sqrt{C}})^2 \qquad (4.32)$$

Fig. 4.5 illustrates the bit error probability of the coherent systems.

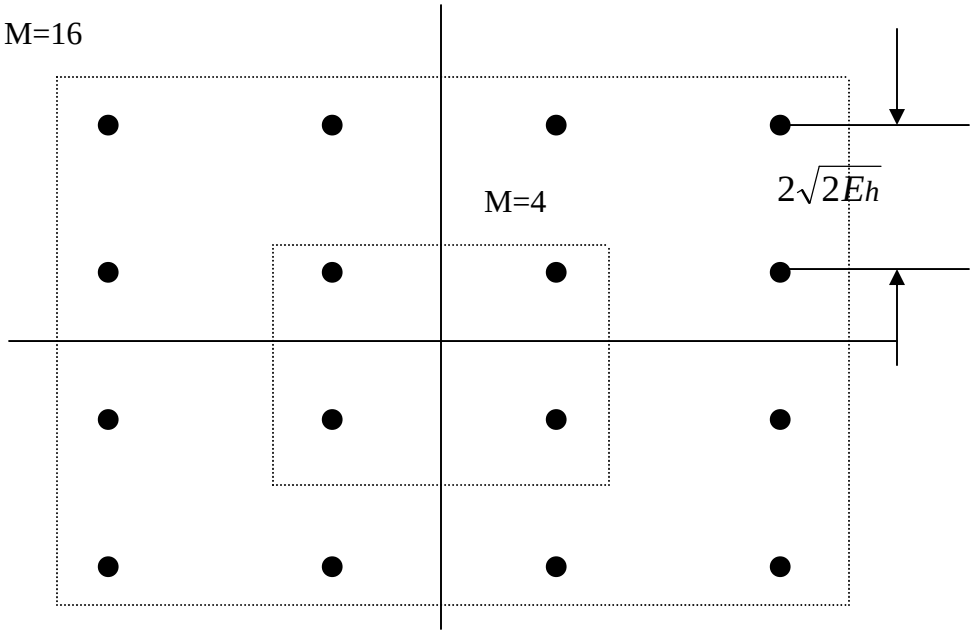


Figure 4.4.

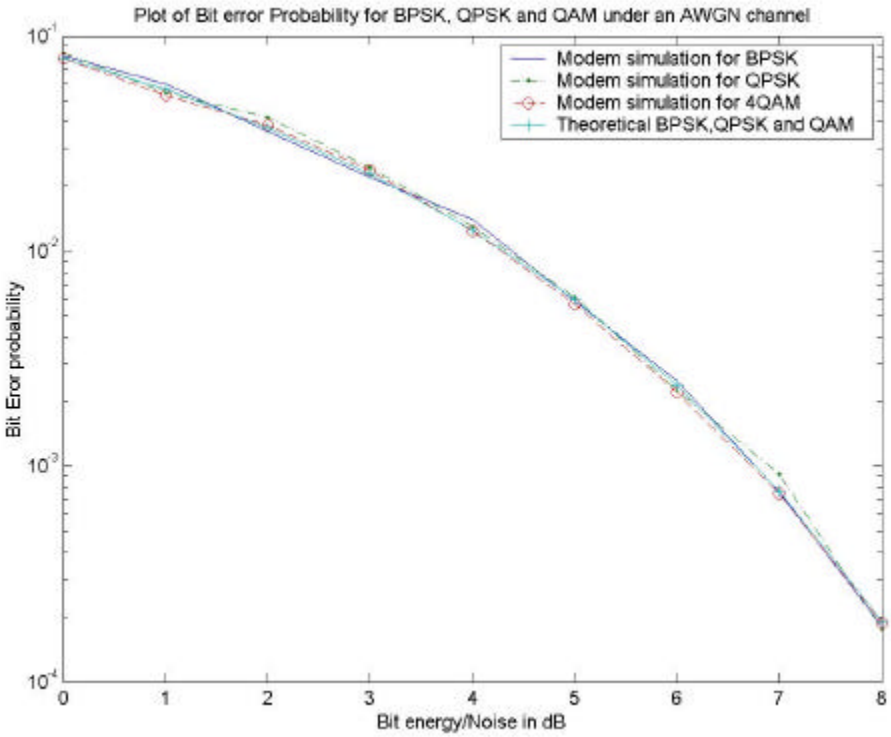


Figure 4.5

4.4.4 Incoherent detection

Since the incoherent receiver does not perform any phase compensations, its objective is to make a correct decision based on the energy received. This is illustrated in Fig 4.6.

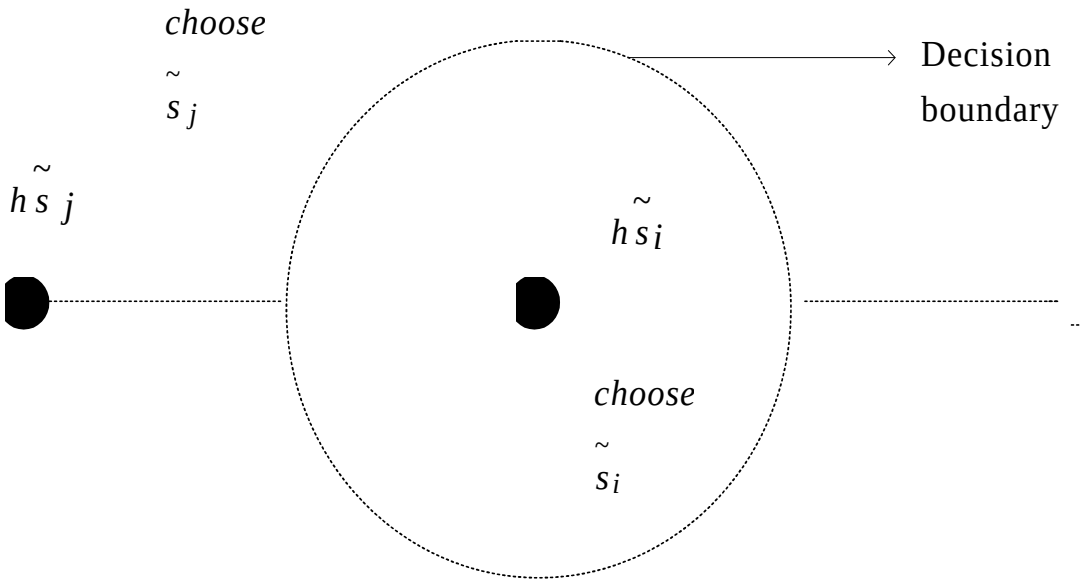


Figure 4.6.

Since we only have two symbols and one of them is the only possibility for energy, we have:

$$\tilde{s}_j = h\sqrt{2E_g} \qquad (4.33)$$

$$\tilde{s}_i = 0 \qquad (4.34)$$

Therefore, the decision boundary is just a radius of length $\frac{h\sqrt{2E_g}}{2}$ and the Euclidian distance between the vectors is $\tilde{d}_{ij} = h\sqrt{2E_g}$. It is important to recognize that the Incoherent system transmits 1 bit/symbol which implies that the symbol energy equals the bit energy. Thus, using (4.11), the probability of bit error for the incoherent scheme is:

$$\mathrm{P}b(SNRb) = Q\left(\sqrt{\frac{a^2(\tilde{d}\,i_j)^2}{4No}}\right)$$

$$= Q\left(\sqrt{\frac{a^2(2E_g)}{4N_o}}\right)$$

$$= Q\left(\sqrt{\frac{a^2 E_g}{2No}}\right)$$

$$\mathrm{P}b(SNRb) = Q\left(\sqrt{\frac{SNRb}{2}}\right) \qquad (4.35)$$

where $SNRb$ represents the bit energy-to-noise ratio

$$SNRb = \frac{a^2 Eg}{No} \qquad (4.36)$$

5 Further improvements and applications

The major improvement for this modem is the channel estimation and the sampling jitter we observed (see section 4.3.2.3). Once this problem is taken care of, the modem should be much more robust to different channels and noise variances.

A good simulation would be to use the same medium as the Dial-up modem since the wireless coherent OFDM modem we built uses the same band. The major obstacle would be the interface between Matlab and the software to send using that port of the computer. Their respective results could then be compared.

This modem could be used under water. The Cornell Robocop team also showed interest in it for the communication between their different robots. However, they are not using as of now the same range of frequencies.

Broadband communications are needed for shallow water acoustic transmission. For these applications, the respective channel induces co-channel interference, high Doppler frequency offsets and multipath propagation effects. All these represent complications for appropriate data reception. The use of a coded OFDM system (COFM) is a good scheme for under water acoustic communication due to its strong ability to withstand a multipath fading channel [2]. The coded data allows the system to be elastic in frequency-selective and time-varying systems which is an advantage generated from the use of time and frequency diversity [2]. For efficiency purposes, the guard interval is not used in this system. When the system is heavily affected by noise, each OFDM block is retransmitted four times [2].

To estimate the channel and frame synchronization, pilot tones are included in the transmitted signal. In this case, however, this process is complemented by a feedback system that must maintain an accurate estimate of the channel transfer function between the relatively infrequent channel sounding signals [2].

6 References

[1] Gordon L. Stuber, Principles of Mobile Communications, Kluwer Academic Publishers, 2001

[2] Khaled Fazel and Stefan Kaiser, Multi-Carrier Spread-Spectrum & Related Topics, Kluwer Academic Publishers, 2000

7 Appendix-

7.1 List of MATLAB functions used in the code

FOPEN Open file.
FID = FOPEN(FILENAME) opens the file FILENAME for read access.

FREAD Read binary data from file.
[A, COUNT] = FREAD(FID) reads binary data from the
Specified file and writes it into matrix A. It also returns the number of data elements read in COUNT

FWRITE Write binary data to file.
COUNT = FWRITE(FID,A) writes the elements of matrix A
to the file specified by the file descriptor (FID). It returns the number elements written in COUNT

FCLOSE Close file.
FCLOSE(FID) closes the file with file identifier FID.

RESHAPE Change size.
RESHAPE(X,M,N) returns the M-by-N matrix whose elements
are taken column wise from X.

FFT Discrete Fourier transforms.
FFT(X,N) is the N point discrete Fourier transform (DFT) of vector X.

IFFT Inverse discrete Fourier transform.
IFFT(X,N) is the N point inverse discrete Fourier transform of X.

RANDN Normally distributed random numbers.
RANDN(N) is an N-by-N matrix with random entries, chosen from a normal distribution with mean zero, variance one and standard deviation one.

SYMERR Compute number of symbol errors and symbol error rate.
[NUMBER,RATIO] = SYMERR(X,Y) compares the elements in the two matrices X and Y. The number of the differences is output in NUMBER. The ratio of NUMBER to the number of elements is output in RATIO (i.e. the error rate).

WAVPLAY Play sound using Windows audio output device.
WAVPLAY(Y,FS) sends the signal in vector Y with sample frequency of FS Hertz to the Windows WAVE audio device.

WAVRECORD Record sound using Windows audio input device.

WAVRECORD(N,FS,CH) records N audio samples at FS Hertz from
CH number of input channels from the Windows WAVE audio device.

C = CONV(A, B) convolves vectors A and B.
The resulting vector is length LENGTH(A)+LENGTH(B)-1.

PINV Pseudo inverse.
X = PINV(A) produces a matrix X of the same dimensions as A' so that A*X*A = A, X*A*X = X and A*X and X*A are Hermitian.

7.2 A list of functions and their pseudo code written by us

BIN = readfile(filename)
Converts ASCII characters given in a TEXT file (filename) to a binary string of data and returns it in 'BIN' array.

```
{
    open file;
    read data from the file to get the ASCII number representation
    (E.g.- 'a' = 97);
    call 'num2binmap' function to convert the ASCII values to binary
    representation (E.g.- 97 = 1100001);
    close file;
}
```

A = num2binmap(Y)
Finds the binary equivalent of an array of numbers and returns it in array A

```
{
    call 'dec2bin' function of matlab to convert to binary representation;
    call str2num function of matlab to convert the binary representation into a
    binary string representation. (E.g.- 1100001 = [1 1 0 0 0 0 1];
}
```

[grayperm] = grayperm(d);
Returns the gray permutation for a given constellation M (d = log2m) in the vector 'grayperm'

```
{
    Find the gray permutation by XORing each element with its previous
    element;
}
```

[block] = make_blocks(tx_sig,N)
Returns a matrix with N columns.

```
{
    Find the number of rows for the final matrix (R)
    If the length of 'tx_sig' is not an integer multiple of N then pad with zeros;
```

Call *'reshape'* function to make blocks of size RxN.
}

[rescaled_data] = rescale_data(data, limits)
Rescales the data between -limits to +limits and returns it in rescaled_data matrix
{
Find the minimum and maximum of the entire matrix;
max_val = limits; min_val = - limits;
Linearly rescale the data in between these 2 limits;
}

block = add_sync_header_seq (data,constellation_size,block_size)
Adds the synchronization sequence and the header sequence in front of the data.
{
Generate the synchronization sequence (chirp signal) ('sync_seq');
Modulate it ('sync_seq_mod');
Generate a header sequence of a particular frequency depending on the constellation size and size of each block (Np) ('header');
Append 'sync_seq_mod' and 'header' in front of the data;
}

[M,Ns,Nh] = find_header(rx_signal)
Returns the constellation size (M), the block size (Ns) and the header length (Nh) by convolving the received header sequence with a bank of corelators and picking the one for which the correlation is maximum.
{
Convolve the received signal with a bank of corelators;
Choose the correlate with the maximum correlation value;
Set M and Ns pertaining to that correlate;
}

writefile(array)
This function converts the given array of symbols to its ASCII number representation and writes it into a file
{
call *'num2binmap'* to convert the symbols to binary form. (E.g.- 3 = 11);
convert the binary represent to a string representation (E.g.- [11] = [1 1]);
reshape the binary string in columns of 7 (Note- ASCII representation is 7 bits long);
call *'bin2dec'* to convert the bit string to ASCII number
(E.g. – [1100001] = 97);
open file for writing;
write to file;
close file;
}

7.3 Pseudo code: Incoherent and Coherent modems

7.3.1 Transmitter

{

Load the global file to get the needed variables like costellation size, packet size;
Calculate the number of zeros (Nz) and the total packet size (Np);
Call function *'readfile'* to get binary data from the text file;
Calculate the number of bits per symbol (d) where d = logM; *****
Call function *'grayperm'* to get the gray permuting vector (v); *****
Convert the bits to symbols (symb); *****
Convert the symbols to gray permuted symbols (symb_g); *****
Modulate the data using either PSK/ASK modulation; *****
Make blocks of size Nsymb (the number of symbols transmitted per block);
Pump power into the various frequencies depending channel transfer function;
Insert pilot tones with even spacing; *
Flip data and zero pad it;
Take Np point IFFT of data;
Rescale data so that it has a dynamic range >1;
Add cyclic prefix to each block;
Call function *'add_sync_header_seq'* to add synchronization sequence and header sequence;
Call function *'wavplay'* to transmit data through the speakers;

}

*** - Not done for incoherent modem**

7.3.2 Receiver

{

Call function *'wavrecord'* to receive data through the microphone;
Load the global file to get the needed variables;
Generate local synchronization sequence (sync_seq);
Get a baseband copy of the received signal (rx_baseband);
Convolve 'rx_baseband' with 'sync_seq' to get the start of the message;
Calculate MMSE estimate of the channel
Call function *'find_header'* to find the constellation (M) and the block size (Ns);
Strip off the synchronization sequence and the header sequence;
Make blocks of size Np+prefix_length;
Remove the cyclic prefix;
Take Np point FFT;
Remove the zero padding and the flipped component of the signal;
Calculate the estimated channel transfer function (H_estim) with the help of pilot tones; *
Normalize the data with 'H_estim'; *
Remove pilot tones; *
De-Modulate the data using either PSK/ASK demodulation; *

Call function *'writefile'* to write data into a file;
}

*** - Not done for incoherent modem**

7.4 Actual code

7.4.1 Common functions

7.4.1.1 add_sync_header_seq

```
function block = add_sync_header_seq(data,const_size,pkt_size)

global_var = load('globals2.txt');
sync_seq_len= global_var(6);
a = global_var(7);
mu = global_var(8);
t = [0:sync_seq_len-1]';

sync_seq=0.45*sin(2*pi*mu/sync_seq_len*t.^2);
L_estim = global_var(4);
sync_seq_mod = [sync_seq.*sin(2*pi*a*t); zeros(L_estim,1)];% Modulated
sync seq
% Make header of size Nh
Nh = global_var(9);
Nc = 20;
T = 10;
time = [0:Nh-1]';

if ((const_size == global_var(3)) & (pkt_size == global_var(1))) %--
>modifying pkt_size is Ns
    header = 0.45*cos((pi/T)*(Nc+3)*time);% 3 since the frequency
should be > 0.1
elseif ((const_size == 4) & (pkt_size == 100))
    header = 0.45*cos((pi/T)*(Nc+4)*time);
elseif ((const_size == 4) & (pkt_size == 1000))
    header = 0.45*cos((pi/T)*(Nc+5)*time);
elseif ((const_size == 16) & (pkt_size == 10))
    header = 0.45*cos((pi/T)*(Nc+6)*time);
elseif ((const_size == 16) & (pkt_size == 100))
    header = 0.45*cos((pi/T)*(Nc+7)*time);
elseif ((const_size == 16) & (pkt_size == 1000))
    header = 0.45*cos((pi/T)*(Nc+8)*time);
else
    disp('There is some error in the values of const_size and
pkt_size');
end


%Add the sync and header to the data
block = [sync_seq_mod;header;data(:)];
```

7.4.1.2 find_header

```
function [M,Ns,Nh] = find_header(rx_signal)
%rx_signal- column vector

global_var = load('globals2.txt');
Nh = global_var(9);
Nc = 20;
T = 10;
t = [0:Nh-1]';
data = [];

% Make a bank of correlators
for j = 1:6 % There are 6 header sequences
    s(1:length(t),j) = cos((pi/T)*(Nc+(j+2))*t);
end

% Convolve the rx signal with the corellators
for j = 1:6
    c(:,j) = conv(rx_signal(1:Nh+100),flipud(s(:,j)));% I guess 200
points of rx_signal is quite high. Can take 150
end
% Find the maximum
[m n] = max(max(c));
[tmp max_val] = max(c(:,n)) % Just for display. Remove later

switch n
case 1
     M = global_var(3);
     Ns = global_var(1);
case 2
    M = 4;
    Ns = 100;
case 3
    M = 4;
    Ns = 1000;
case 4
    M = 16;
    Ns = 10;
case 5
    M = 16;
    Ns = 100;
case 6
    M = 16;
    Ns = 1000;
otherwise
    disp('Constellation size and no of symbols per block cound not be
found');
end
```

7.4.1.3 grayperm

```
function [grayperm]=grayperm(d);
%Gives the gray permutation for a given constalltion size
%Usage- [grayperm]=grayperm(d)
%Inputs- d = symbolsize in bits = log2(M) where M is the constallation
size
%Outputs- grayperm= the permuted array

 grayperm=[0];
if d<1
    grayperm=[0];
end
if d==1
    grayperm=[0 1];
else
    grayperm=[0 1 3 2];
end

ind=2;
a=grayperm;
while d-ind>0
     ind=ind+1;
     b=fliplr(a);
     a=[a,(b+2^(ind-1))];
end
grayperm=a;
```

7.4.1.4 make_blocks

```
function [block] = make_blocks(tx_sig,N)
% make blocks of size N
%Usage- [block] = make_blocks(tx_sig,N)
%Input-
%    tx_sig = the signal whose blocks have to be made
%    N = size of each block
%Output-
%    blocks = Matrix of blocks

no_of_blocks = ceil(length(tx_sig)/N);

% If the size of the block is not an integer multiple of size of eack
block then padd with zeros
if (mod(length(tx_sig),N) ~= 0)
   no_of_zeros = no_of_blocks*N - length(tx_sig); % Find the number of
zeros to be inserted
   tx_sig = [tx_sig;zeros(no_of_zeros,1)]; % Do zero padding
end
block = reshape(tx_sig,N,no_of_blocks);
```

7.4.1.5 num2binmap

```
function A = num2binmap(Y)
% Usage: A = num2binmap(Y)
% Finds the binary equivalent of a number
% Inputs: Y - numbers
% Output: A- bin data

    X = dec2bin(Y);
    Z = X';
    A = str2num(Z(:))';
```

7.4.1.6 readfile

```
function bin = readfile(filename)
% Converts the ascii characters given in a TEXT file to a binary string
of data.
%Usage- bin = readfile(filename)
%Iputs-
%    filename- name the file (Eg- 'try.txt')
%    constellation- constellation size
%    no_symb- symbols per packet
%Outputs
%    bin- the binary string of data

fdr = fopen(filename,'r'); % Open the file to read
if (fdr == -1)
    disp('File cannot be opened or found');
else
    input = fread(fdr);
end
fclose(fdr);
bin = num2binmap(input); % Convert the ascii characters(numbers) into
binary
```

7.4.1.7 rescale_data

```
function rescaled_data = rescale_data(data,limits)
% This function rescales the data between -limits to +limits

data_min = min(min(data));
data_max = max(max(data));
slim = [data_min data_max];
dx=diff(slim);
rescaled_data = limits*((data - slim(1))/dx*2-1);
```

7.4.1.8 writefile

```
function writefile(array)
% This function converts the given array of symbols to its ascii number
% and writes it into a file
%Usage- writefile(array_of_bits)
%Inputs-
%    array- array of symbols
%Outputs- None
% Global Parameters-
%   train_seq= training sequence which should be known by the
transmitter and the receiver

array_of_bits = num2binmap(array); %COnver the symbols to binary digits

x = num2str(array_of_bits(:)).';  % Done for proper arrangement
len = length(x) - mod(length(x),7);
x = x(:,1:len); % Truncate to the nearest integer divisible by 7

rows = ceil(length(x)/7); % find the number of rows required
array = reshape(x,7,rows); % arrange into columns of 7 to be converted
to ascii
num = bin2dec(array.');    % COnver into decimal
fdw = fopen('write.txt','w'); % Open file to write
count = fwrite(fdw,num);
fclose(fdw);
```

7.5 Incoherent modem

7.5.1 Globals text file

% 1. Ns -> block size
 40

% 2. prefix length (L_estim < prefix_length < Np)
 80

% 3. M -> constellation size
 2

% 4. L_estim -> Estimated channel length
 16

% 5. fmin -> minimum frequency
 0.2

% 6. sync_seq_len -> synchronization sequence length
 2000

% 7. a -> MOdulation frequency for the synchronization sequence
 0.2

% 8. mu -> frequency of the synchronization sequence
 0.15

% 9. Nh -> header length
 2000

% 10. Fs -> wavplay and wavreceive Transmission/Receiving frequency
 8000

7.5.2 Transmitter

```
filename = 'try1.txt';
global_var = load('globals.txt');
Ns= global_var(1);%--> size of the block of symbols transmitted through
the ifft
fmin=global_var(5);%--> minimum frequency
Nz=ceil(fmin/(1-2*fmin)*(2*Ns+1)); %--> integer number of zeros added
Np=2*(Ns+Nz)+1;
prefix_length = global_var(2);

symb = readfile(filename); % Read the text file
M = global_var(3);
d = log2(M);
sig_ifft = make_blocks(symb',Ns-2);
[m n] = size(sig_ifft);
sig_ifft = [sig_ifft;zeros(2,n)];
gain= [ones(1,3) ones(1,8)+.1.*[1:8] ones(1,2) ones(1,14)+0.1.*[1:14]
ones(1,13)]';
[m,n] = size(sig_ifft);
a = [zeros(11,n); 0.8*zeros(2,n) ;zeros(14,n)+.3; zeros(13,n)+1.5];
sig_ifft1= diag(gain)*sig_ifft+a;
sig_ifft = [zeros(Nz+1,n); sig_ifft;
flipud(conj(sig_ifft));zeros(Nz,n)];
u=ifft(sig_ifft,Np);
u=Np*real(u); %Energy of 1
u = rescale_data(u,0.45); % Rescale the data to the rance -0.5 to 0.5
u = [u((Np-prefix_length+1):Np,:);u]; % add cyclic prefix to each block
% Add synchronization sequence of size N
u = add_sync_header_seq(u,M,Ns);
u = make_blocks(u,Np+prefix_length);

save -ascii tx_file.txt u; % save the file as ascii characters
Fs = global_var(10);
wavplay(u(:),Fs);
```

7.5.3 Receiver

```
global_var = load('globals.txt');
prefix_length = global_var(2);
fmin=global_var(5);%--> minimum frequency

Fs = global_var(10);
rx = wavrecord(4*Fs,Fs);
save rx.txt rx -ascii;
plot(rx);

%---> matched filter
sync_seq_len = global_var(6);
a = global_var(7);
mu = global_var(8); %--> mu<=0.1
t = [0:sync_seq_len-1]';
sync_seq=0.45*sin(2*pi*mu/sync_seq_len*t.^2);
sync_seq_mod = sync_seq.*sin(2*pi*a*t);% Modulated sync seq
rx_baseband = rx.*sin(2*pi*a*[0:length(rx)-1]');% bring the signal to
baseband to do matched filtering
z=conv(rx_baseband,flipud(sync_seq)); %convolve with only part of the
signal to make fast
[tmp start] = max(abs(z))
plot(z);
L_estim = global_var(4);%10; % estimated channel length

%--->Calculating the header
[M Ns header_len] = find_header(rx(start+L_estim+1:end)); % Calculate
the header and get data with header stripped off
Nz=ceil(fmin/(1-2*fmin)*(2*Ns+1));
Np=2*(Ns+Nz)+1;
d = log2(M);

go_back = 65;
rx = rx(start-go_back+L_estim+header_len+1:end); % strip off the sync
sequence and the header

rx = make_blocks(rx,Np+prefix_length);
rx = rx(prefix_length+1:Np+prefix_length,:);%remove_cyclic_prefix

est_u = fft(rx,Np);
[l k] = size(est_u);
est_u = est_u(Nz+2:(Ns+Nz+1),:); % remove the zeros and flipped stuff
hat_est_u = abs(est_u);
No = 0.01;
[m,n] = size(hat_est_u);
T = sqrt(2)*abs(Hmmse)/2; % threshold vector
mtx_T = diag(T)*ones(m,n);
a = find(hat_est_u<=mtx_T);
hat_symb = ones(m,n);
hat_symb(a) = 0;
hat_symb = hat_symb(1:Ns-2,:);
dec = hat_symb(:);
writefile(dec);
```

7.6 Coherent modem

7.6.1 Globals text file

% 1. Ns -> block size to change when change Q
 39

% 2. prefix length (L_estim < prefix_length < Np)
 35

% 3. M -> constellation size
 4

% 4. L_estim -> Estimated channel length
 13

% 5. fmin -> minimum frequency
 0.05

% 6. sync_seq_len -> synchronization sequence length
 2000

% 7. a -> Modulation frequency for the synchronization sequence
 0.2

% 8. mu -> frequency of the synchronization sequence
 0.15

% 9. Nh -> header length
 2000

% 10. Fs -> wavplay and wavreceive Transmission/Receiving frequency
 8000

% 11. Q -> number of symbols per pilot tone, needs to be odd
 3

7.6.2 Transmitter

```
filename = 'try1.txt';

global_var = load('globals2.txt');
L_estim = global_var(4);
Q=global_var(11);
Nsymb=(Q-1)*L_estim;
```

```
Ns= Nsymb+L_estim;%--> size of the block of symbols transmitted through
the ifft
fmin=global_var(5);%--> minimum frequency

Nz=ceil(fmin/(1-2*fmin)*(2*Ns+1)); %--> integer number of zeros added
Np=2*(Ns+Nz)+1 ; % # of pilot tones is L_estim

prefix_length = global_var(2);

bits = readfile(filename); % Read the text file
M = global_var(3);%2;
d = log2(M);
bits=reshape(bits,d,length(bits)/d);
[vec]=grayperm(d);
symb= (bits'*2.^(d-1:-1:0)')'; %--> convert to decimal
save symb.txt symb -ascii;
symb_g=vec(symb+1);   %--> gray mapping
symb_g=symb_g(:);

%*********** Modulation

val = 1;
if val == 1 % method is 'PSK'
    sig=exp(j*2*pi/M*symb_g);
elseif val == 2
    sig = (dmodce(symb,1,1,'qask')).'; % method = QASK
else
    disp('Wrong modulation specified');
end
sig_ifft = make_blocks(sig,Nsymb);
fglobal_var = load('globals2.txt');
L_estim = global_var(4);
Q=global_var(11);
q = floor(Q/2);
[m,n]=size(sig_ifft);
app=[sig_ifft(1:q,:)];
for k=0:L_estim-2
    app=[app;ones(1,n);sig_ifft(q+1+k*(Q-1):q+(k+1)*(Q-1),:)];
end
app = [app;ones(1,n);sig_ifft(q+1+(L_estim-1)*(Q-1):end,:)];
sig_ifft =  app;
% flip zero pad
sig_ifft = [zeros(Nz+1,n); sig_ifft;
flipud(conj(sig_ifft));zeros(Nz,n)];
[m,n]=size(sig_ifft);
indx = [fliplr([Nz+2+q-Q:-Q:2]) [Nz+1+2*Ns-q+Q:Q:m]];
sig_ifft(indx)=1;
u=ifft(sig_ifft,Np);
u= Np*real(u); %Energy of 1
u = rescale_data(u,0.45); % Rescale the data to the range -0.5 to 0.5
u = [u((Np-prefix_length+1):Np,:);u]; % add cyclic prefix to each block
u = add_sync_header_seq(u,M,Ns);

Fs = global_var(10);
wavplay(u(:),Fs);
```

7.6.3 Receiver

```
% Coherent receiver
global_var = load('globals2.txt');
L_estim = global_var(4);
Q=global_var(11);
fmin=global_var(5);%--> minimum frequency
Fs = global_var(10);
rx0 = wavrecord(4*Fs,Fs);
rx = rx0;
plot(rx)

sync_seq_len = global_var(6)
a = global_var(7);
mu = global_var(8);%--> mu<=0.1
t = [0:sync_seq_len-1]';
sync_seq=0.45*sin(2*pi*mu/sync_seq_len*t.^2);
sync_seq_mod = sync_seq.*sin(2*pi*a*t);% Modulated sync seq
rx_baseband = rx.*sin(2*pi*a*[0:length(rx)-1]');% bring the signal to
baseband to do matched filtering
z=conv(rx_baseband,flipud(sync_seq)); %convolve with only part of the
signal to make fast
[tmp start] = max(abs(z))

[M Ns header_len] = find_header(rx(start+L_estim+1:end)); % Calculate
the header and get data with header stripped off
go_back = 5;
rx = rx(start-go_back+L_estim+header_len+1:end); % strip off the sync
sequence and the header
Nz=ceil(fmin/(1-2*fmin)*(2*Ns+1)); %--> integer number of zeros added
Np=2*(Ns+Nz)+1 ; % # of pilot tones is L_estim
prefix_length = global_var(2);%20;
q = floor(Q/2);

rx = make_blocks(rx,Np+prefix_length);
rx = rx(prefix_length+1:Np+prefix_length,:);%remove_cyclic_prefix
est_u = fft(rx,Np);
ind_p=[fliplr([Nz+2+q-Q:-Q:2]) [Nz+2+q:Q:Np]];
A = (ind_p(:)*(0:L_estim-1))/Np;
Mat = exp(-j*2*pi*A);
cond(Mat)
h_estm=pinv(Mat)*[est_u(ind_p,:)];
H_estim = fft(real(h_estm),Np); % imaginary part due to noise
[m,n]=size(est_u);
hat_est_u = est_u.*conj(H_estim);
hat_est_u = hat_est_u(Nz+2:Nz+1+Ns,:); % strips zeros and flipped
version
app = [hat_est_u(1:q,:)];
for k = 0:L_estim-2
   app = [app;hat_est_u(q+2+k*Q:q+(k+1)*Q,:)];
end
app = [app;hat_est_u(q+2+(L_estim-1)*Q:end,:)];

sig_fft = app;
d = log2(M);
[vec]=grayperm(d);
```

```
val  = 1;
if val == 1 % method is 'PSK'
        %****** demodulate PSK
        dec=exp(-j*2*pi/M*vec).'*sig_fft(:).'; %--> deciding is
combined with the gray demapping
        [value,dec]=max(real(dec));
        dec=dec-1;
elseif val == 2
        % put ASK modulation
        dec = ddemodce(sig_fft,1,1,'qask',M); % demodulation ASK
else
        disp('Wrong modulation specified');
end

filename = 'try1.txt';
bits = readfile(filename); % Read the text file
M = global_var(3);
d = log2(M);
bits=reshape(bits,d,length(bits)/d);
[vec]=grayperm(d);
symb= (bits'*2.^(d-1:-1:0)')'; %--> convert to decimal

err = length(find(dec(1:length(symb))~=symb))
ber = err/length(symb)

writefile(dec);

fd = fopen('try1.txt');
in = fread(fd);
fd2 = fopen('write.txt');
out = fread(fd2);
err1 = length(find(out(1:length(in))~=in))
cer = err1/length(in)
```