Spatiotemporal Processing and Time-reversal for Underwater Acoustic Communications

by

Daniel Y. Wang

B.S., Electrical Engineering
North Carolina State University, 1998

Submitted to the Department of Ocean Engineering and the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer
and
Master of Science in Electrical Engineering and Computer Science

at the
Massachusetts Institute of Technology
June 2005

Signature of Author [illegible]
Department of Ocean Engineering and the
Department of Electrical Engineering and Computer Science
May 6, 2005

Certified by [illegible]
Milica Stojanovic, Principle Scientist
MIT Sea Grant Program
Thesis Supervisor

Certified by Arthur B. Baggeroer
Arthur B. Baggeroer, Ford Professor of Engineering
Department of Electrical Engineering and Computer Science
Thesis Reader

Accepted by [illegible]
Michael Triantafyllou, Professor of Ocean Engineering
Chair, Departmental Committee on Graduate Students
Department of Ocean Engineering

Accepted by [illegible]
Arthur C. Smith, Professor of Electrical Engineering and Computer Science
Chair, Departmental Committee on Graduate Students
Department of Electrical Engineering and Computer Science

Spatiotemporal Processing and Time-reversal for Underwater Acoustic Communications

By

Daniel Wang

Submitted to the Departments of Ocean Engineering and
Electrical Engineering and Computer Science
On May 11, 2005, in partial fulfillment of the
Requirements for the degrees of
Naval Engineer
and
Master of Science in Electrical Engineering and Computer Science

Abstract

High-rate underwater acoustic communication can be achieved using transmitter/receiver arrays. Underwater acoustic channels can be characterized as rapidly time-varying systems that suffer severe Inter Symbol Interferences (ISI) caused by multi-path propagation. Multi-channel combining and equalization, as well as time-reversal techniques have been used over these channels to reduce the effect of ISI. As an alternative, a spatiotemporal focusing technique had been proposed. This technique is similar to time-reversal but it explicitly takes into account elimination of ISI. To do so, the system relies on the knowledge of channel responses. In practice, however, only channel estimates are available.

To assess the system performance for imperfectly estimated time-varying channels, a simulation analysis was conducted. Underwater acoustic channels were modeled using geometrical representations of a 3-path propagation model. Multi-path fading was incorporated using auto regressive models. Simulations were conducted with various estimator delay scenarios for both the spatiotemporal focusing and simple time-reversal. Results demonstrate performance dependence on the non-dimensional product of estimation delay and Doppler spread. In particular, it has been shown that when this product is low, the performance of spatiotemporal focusing remains superior to simple time-reversal.

Thesis Supervisor: Milica Stojanovic
Title: Principle Scientist, MIT Sea Grant Program

Thesis Reader: Arthur B. Baggerorer
Title: Ford Professor Engineering

TABLE OF CONTENTS

TABLE OF FIGURES

1.0 Introduction

Acoustic transmission in the ocean often has multiple paths due to interactions between sound waves and the sea surface and the bottom. This pattern of multiple-path transmission leads to significant time spread at the receiver. The pattern will also experience time-variation as the surface of the sea changes with waves. Such time-variant spreading along with dispersion makes underwater acoustic communications extremely difficult.

This paper is to document the simulation evaluation of an innovative optimization technique in underwater acoustic communications. The technique is derived from the general principles of time-reversal or phase-conjugate focusing [1], [2], but with the explicit requirement of eliminating the Inter Symbol Interference (ISI) while preserving constant power constraint. The technique is referred to as spatiotemporal focusing and can be further divided into unrestricted two-sided filter adjustment and restricted one-sided filter adjustment [3]. It can be regarded as an alternative to multi-channel equalization [4]. The basics of time-reversal and spatiotemporal focusing are explained in the next section.

1.1 Background

The majority of underwater acoustic channels can be characterized as rapidly time-varying channels that suffer from severe multi-path propagation and large Doppler fluctuations [3] . Inter Symbol Interferences (ISI) caused by multi-path, and Doppler effects caused by channel time-varying characteristics are much more pronounced than those of radio channels where much higher carrier frequencies are used. To combat ISI in underwater communication channels, sophisticated signal processing techniques must be utilized.

Techniques such as adaptive multi-channel combining and equalization are very effective in reducing ISI [4]. Spatial modulation over multi-input multi-output (MIMO) channels had also been investigated for reducing ISI in underwater acoustic systems [5]. However, to obtain the improved performance, computationally complex algorithms are utilized in these approaches.

To reduce the computational complexity of multi-channel equalization, time-reversal arrays have been investigated for use in underwater acoustic communications [6],[7],[8]. Time reversal, or phase conjugation in the frequency domain, refocuses the signal energy back to the source, thus reducing the effect of multi-path.

A highly idealized example can illustrate the time-reversal principle. When a signal $s_i(t)$ is transmitted through the i^{th} channel, its response can be expressed as:

$$r_i(t) = s_i(t) * h_i(t) \tag{1.1.1}$$

where $h_i(t)$ is the impulse response of the channel. The receiver observes $r_i(t)$ and retransmits the time-reversed version of the signal $r_i^*(-t)$ back into the channel. At the locale of the original source, the combined signal (with array size N) will be:

$$s_r(t) = \sum_{i=1}^{N} s_i^*(-t) * (h_i^*(-t) * h_i(t)) \tag{1.1.2}$$

The convolution $h_i^*(-t) * h_i(t)$ enhances the principal component while reducing the relative strength of multi-path components. It can be seen in equation (1.1.2) that the degree to which the signal is focused depends on the size N of the "focusing" array. It is also worth noting that, in the absence of noise, time-reversal is in fact the implementation of channel-matched filters.

Various methods have been proposed to take advantage of the time-reversal principle. These methods include passive phase-conjugation, active phase-conjugation and phase-conjugation with adaptive channel estimation [[6],[7],[8]]. Some of these methods reduce the signal processing at the expense of data rate while others utilize some signal processing (channel estimation) at the receiver to improve performance. They all suffer, however, the common problem of sub-optimal system design in that the ISI elimination is not completely achieved.

1.2 Problem Definition

A new approach has been proposed to combine time-reversal with a system design that is explicitly optimized with respect to ISI elimination and SNR maximization [3]. Referred to as spatiotemporal focusing, it utilizes both the receiver and transmitter filter design to achieve optimization goals. Such optimized design does not depend solely on array size to reduce ISI; instead, it leaves the freedom of adjusting array size vs. computational complexity to the system designer. In this aspect, it differs from typical time-reversal where the signal resolution depends exclusively on the array size. Spatiotemporal focusing is also different from standard receiver-side equalization in that it attempts to optimize both the receiver and transmitter ends (unrestricted two-side filter adjustment). However, if an application calls for limited complexity at one end of the system, retro-focusing optimization can be performed at one end only to reduce computational burden at the other end (restricted one-sided filter adjustment). In this work, we focus on the one-side adjustment system.

To implement this system in practice, it is necessary for the receiver/transmitter to know the channel response between the two ends of the system. One equipped with a simple element and another with an array of elements. When channel estimates are used in place of the true channel responses, estimation errors will affect system performance. The objective of this work is to assess the impact of the estimation errors on the system performance. Notable for acoustic channels is the long propagation delays, which may cause significant difference between the time when the channel was observed and the time of transmission.

As precursor to the experimental validation of spatiotemporal focusing techniques, simulations of a spatiotemporal focusing system in shallow water environment have been conducted. In order to facilitate comparison with simple time reversal, simulations of time-reversal in the same environment have also been completed. There were several steps in approaching the task of constructing the simulation model. First, we defined the system and its optimization scheme including analytical expressions that specify optimal filters according to time reversal and spatiotemporal criteria [3]. Second, we constructed shallow water communication channels using a simplified 3-path model. The channels were first approximated as time-invariant with all channel characteristics being calculated using channel geometry. Then the channel variations were modeled as Gaussian random processes. Auto regressive model of order one (AR1) and order two (AR2) were considered for modeling channel variations. In addition, two types of possible estimator schemes were offered for estimating time-varying channels. Finally, simulation runs were conducted using AR2 model with a simple delay estimator scheme for restricted one-side focusing and time reversal. The results were graphed and discussed.

Chapter 2 describes the system optimization through filter calculations for both time reversal and spatiotemporal focusing. Chapter 3 specifies channel model. The simulation results, including time reversal performance results, are presented in Chapter 4 where the performance of one-side spatiotemporal focusing is compared with that of time-reversal/phase conjugation. The advantages and disadvantages of each technique are discussed. Some of the future research works are discussed in Chapter 5 along with the concluding remarks.

2.0 System Optimization

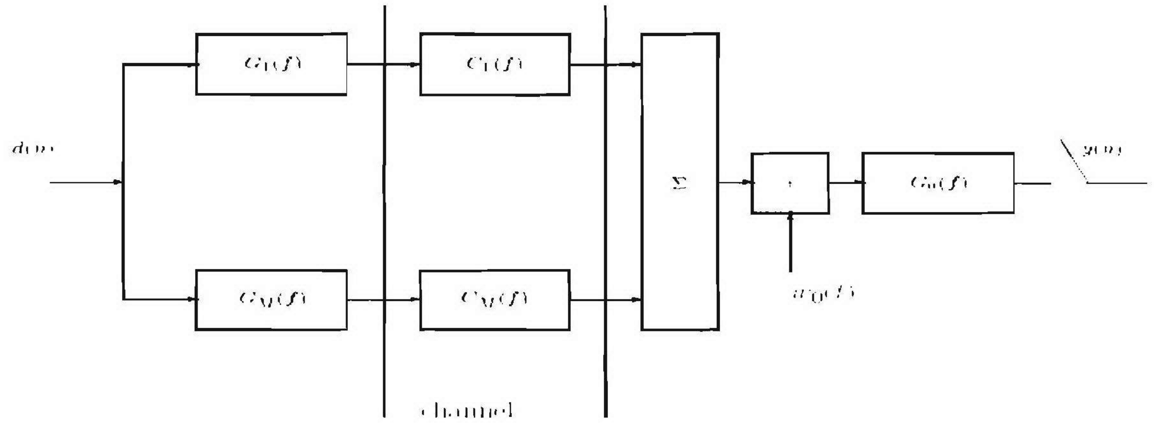

Figure 2-1 Downlink system schematic

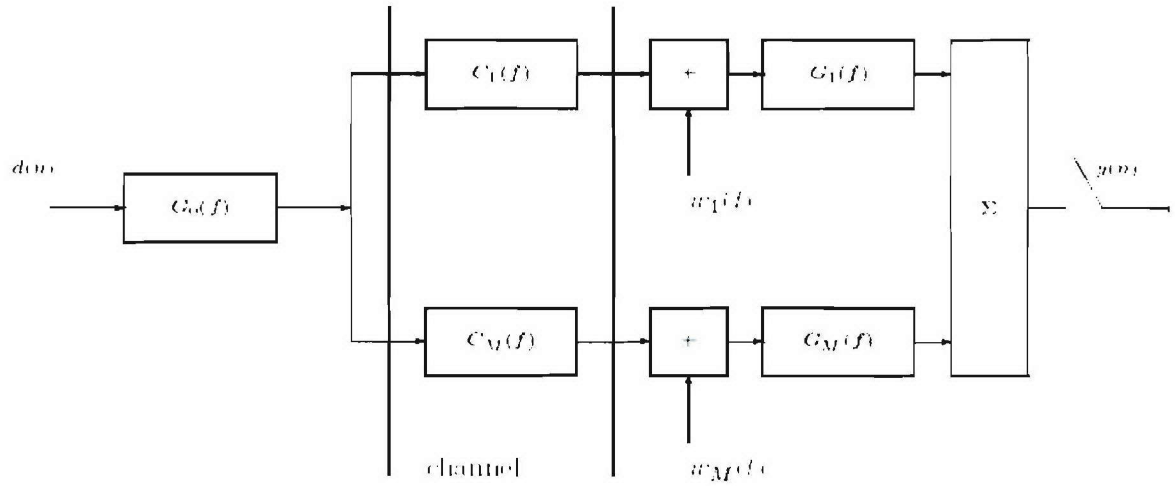

Figure 2-2 Uplink system schematic

In this chapter, system optimization is addressed for uplink and downlink communications. The schematics of downlink and uplink are shown in Figure 2-1 and Figure 2-2, respectively. The direction from multi-element array to single element is defined as downlink; the direction from single element to multi-element array is defined as uplink. Data stream is represented by $d[n]$. $y[n]$ is the sampled signal stream with the sampling rate of 1/T, where T is the symbol interval.

The figure of merit for system performance will be output Signal-to-Noise-Ratio (SNR) as measured in the variable $y[n]$. Performance comparison between time reversal and spatiotemporal focusing will be made using this SNR value. For time reversal, filters are to be calculated based on the underlining principles of phase-conjugation (time reversal in frequency domain); spatiotemporal filters are to be calculated using SNR maximization with no-ISI and constant energy constraints. We will first start with time-reversal filter calculations.

2.1 Time Reversal

Time reversal technique implements phase-conjugated channel impulse response to process the received signal. In frequency domain, the channel transfer function can be given as[3]

$$C_m(f) = \sum_{p=0}^{P-1} c_{m,p}(\mathrm{t}) e^{-j2\pi f \tau_p(t)} \tag{2.1.1}$$

The composite channel power spectral density $\gamma(f)$ derived as

$$\gamma(f) = \sum_{m=1}^{M} \left| C_m^2(f) \right| \tag{2.1.2}$$

This power spectral density in time domain exhibits certain properties that enable the time-reversal to work. Succinctly put, time reversal is the simplest form of focusing. As the number of channels increases, the center-lobe power

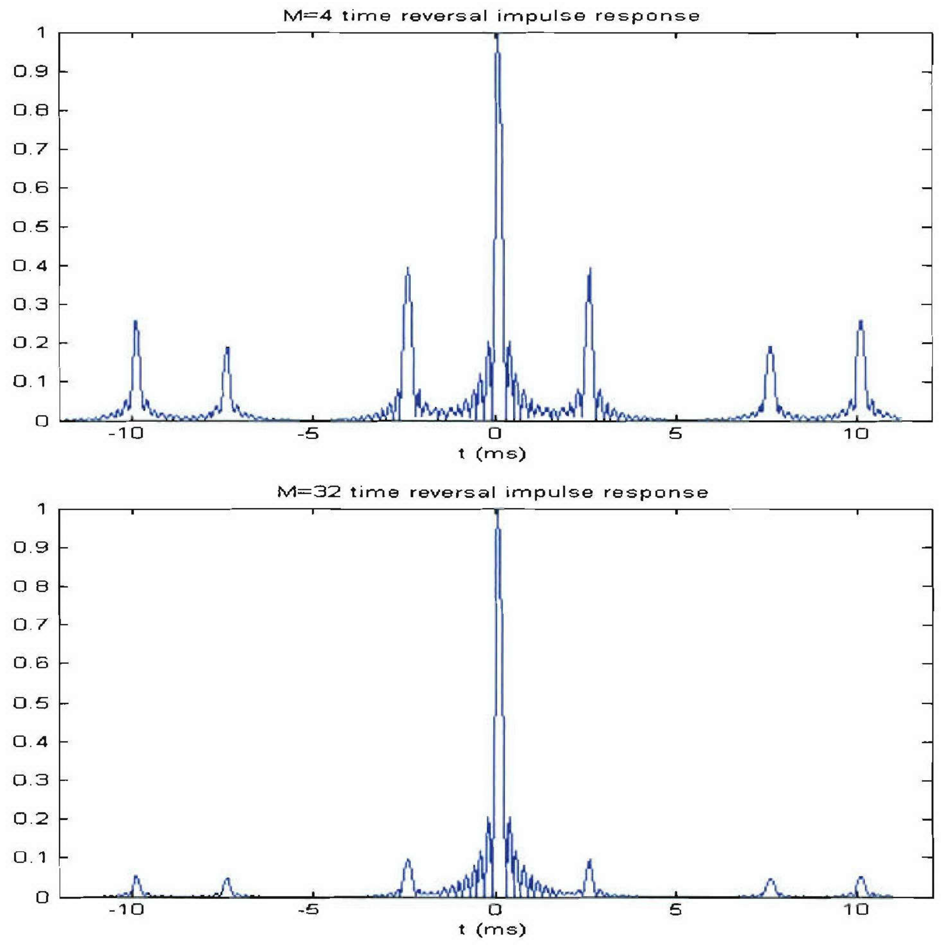

Figure 2-3 Time-reversal impulse response

increases and the side-lobe power decreases. Such impulse response convolved with the received signal will tend to focusing the main power in the zero-referenced signal component while suppressing the delayed components of the signal. Consequently, the ISI between zero-referenced component and delayed components is suppressed. In our model of 3-path communication channel, such phenomenon can be readily seen in Figure 2-3, where low-pass filtered versions of 4-channel composite p.s.d ($X(f)\gamma(f)$) impulse response and 32-channel composite p.s.d impulse response are shown. We can clearly see that the center lobe magnitude increases with the increasing of channel numbers while the side-lobe magnitude decreases at the same time.

To implement time reversal, we will compute the transmit filter and receive filter according to the principle of phase conjugation. For the uplink case, the transmitter filter is the standard square-root raised-cosine filter $\sqrt{X(f)}$ with gain factor of K_u, $G_0(f) = K_u\sqrt{X(f)}$. The receiving filter is given by

$$G_m(f) = G_0^*(f)C_m^*(f), m = 1,...M \tag{2.1.3}$$

The gain constant can be calculated using the transmit energy constraint as

$$K_u = \sqrt{\frac{E/\sigma_d^2}{\int_{-\infty}^{\infty} X(f)df}} \tag{2.1.4}$$

where E is the transmit energy and $\int_{-\infty}^{\infty} X(f)df = x_0$. This filter calculation gives results similar to passive phase conjugation.

For downlink case, the transmitter uses filter

$$G_m(f) = K_d\sqrt{X(f)}C_m^*(f), m = 1,...M \tag{2.1.5}$$

Where K_d can be calculated using energy constraint equation (2.2.23)

$$K_d = \sqrt{\frac{E/\sigma_d^2}{\int_{-\infty}^{\infty} X(f)\gamma(f)df}} \tag{2.1.6}$$

This filter produces result analogous to that of active phase-conjugation.

2.2 Spatiotemporal Focusing

Spatiotemporal focusing explicitly enforces no-ISI condition while maximizing SNR through implementation of temporal focusing in addition to spatial focusing achieved by simple time reversal. There are two implementation variations for spatiotemporal focusing. Unrestricted two-side focusing implementation is a pure form of optimization in that it does not put additional restriction on SNR optimization other than no-ISI and energy constraints. It achieves optimal performance results. Restricted one-side focusing, on the other hand, will sacrifice some performance comparing to two-side focusing in exchange for reduced complexity on one end of the system. As presented by Stojanovic[3] , filters can be calculated analytically with known channel transfer functions and channel noise power spectrum. With that assumption, we will start with one-side restricted filter calculations first.

From now on, we will focus on the case where channel noise is assumed to be white. i.e. $S_w(f) = N_0$.

2.2.1 One-sided Restricted Filters

Some applications require that one side of the communication link can only have minimal complexity, such situation may rise in Autonomous Underwater Vehicles (AUV) applications where volume and power constraint limit the processing complexity inside the AUV. In these situations, we constrain the single-element side to use only the standard fixed filter.

The downlink case is shown below. The no-ISI condition must hold in one-sided restricted case, i.e. $F(f) = X(f)$. To maximize SNR, the transfer function should be divided between the transmitter and receiver so that

$$G_0(f) = \beta \frac{\sqrt{X(f)}}{\sqrt{S_w(f)}} \tag{2.2.1}$$

where β is a constant and $S_w(f) = N_0$ is the power spectral density of the channel noise. SNR can be expressed as

$$SNR = \frac{\sigma_d^2 \left| \int_{-\infty}^{\infty} G_0(f) \sum_{m=1}^{M} G_m(f) C_m(f) \right|^2}{\int_{-\infty}^{\infty} S_w(f) \left| G_0^2(f) \right| df} \tag{2.2.2}$$

where $\sigma_d^2 = E\{d^2[n]\}$ is the power of transmitted signals.

Applying the Shwarz inequality to nominator

$$\left| \int_{-\infty}^{\infty} G_0(f) \sum_{m=1}^{M} G_m(f) C_m(f) \right|^2 \leq \int_{-\infty}^{\infty} \left| G_0(f) \right|^2 \left| \sum_{m=1}^{M} G_m(f) C_m(f) \right|^2 \tag{2.2.3}$$

It follows that

$$SNR \leq \sigma_d^2 \int_{-\infty}^{\infty} \frac{1}{S_w(f)} \left| \sum_{m=1}^{M} G_m(f) C_m(f) \right|^2 df \tag{2.2.4}$$

Applying the Shwarz inequality again to the integrand and substituting

$$\gamma(f) = \sum_{m=1}^{M} \left| C_m(f) \right|^2$$

$$SNR \leq \sigma_d^2 \int_{-\infty}^{\infty} \frac{1}{S_w(f)} \gamma(f) \sum_{m=1}^{M} \left| G_m^2(f) \right| \tag{2.2.5}$$

Where maximum SNR is achieved when $G_m(f) = \alpha(f) C_m^*(f), m = 1, \ldots M$.

Combining this condition with (2.2.1) and no-ISI constraint, we obtain receiving filter for the downlink restricted case

$$G_0(f) = K^{-1}(f)\sqrt{X(f)} \tag{2.2.6}$$

The transmission filter is

$$G_m(f) = K(f)\sqrt{X(f)}\gamma^{-1}(f)C_m^*(f), m = 1,...M \tag{2.2.7}$$

Where the gain factor

$$K(f) = \sqrt{\frac{E/\sigma_d^2}{\int_{-\infty}^{\infty} S_w(f)\frac{X(f)}{\gamma(f)}df}} S_w^{1/2}(f) \tag{2.2.8}$$

With white noise assumption, $S_w(f) = N_0$, the factor $K(f)$ becomes constant independent of frequency. Consequently, the same set of filters can be used for uplink and downlink transmissions. The filters then can be expressed as

$$G_0(f) = K^{-1}\sqrt{X(f)} \tag{2.2.9}$$

$$G_m(f) = K\sqrt{X(f)}\gamma^{-1}(f)C_m^*(f), m = 1,...M \tag{2.2.10}$$

The uplink case can be derived similarly. The filter and gain factor are listed below

$$\begin{aligned} &G_0(f) = K\sqrt{X(f)} \\ &G_m(f) = K^{-1}\sqrt{X(f)}\gamma(f)C_m^*(f) \\ &where K = \sqrt{\frac{E/\sigma_d^2}{x_0}} \end{aligned} \tag{2.2.11}$$

2.2.2 Two-sided Unrestricted Filters

As presented in [3], two-side filter analytical expression can be derived similarly. Assume that the power spectral density *S_w(f)* of the uncorrelated noises *$w_m(t)$, m=0,1...N* are known. Channel responses *$C_m(f)$, m=1,...M* are also assumed known (either *a priori* or through estimation). *$G_0(f)$* and *$G_m(f)$,m=1....N,* are the filters at the transmitter/receiver. The goal is to design the filters *G₀(f)* and *Gₘ(f),m=1...M,* so as to maximize the SNR with constraints of no ISI and finite transmitted energy per symbol. For uplink transmission, the constraints can be expressed as

$$y(nT) = d(n)x_0 + z(nT) \tag{2.2.12}$$

$$E = \sigma_d^2 \times \int_{-\infty}^{+\infty} \left| G_0^2(f) \right| df \tag{2.2.13}$$

The received noise *z(t)* is the result of white Gaussian noise *w(t)* going through LTI filters. For the uplink transmission, if we denote the white noise process as *wₘ(t)* and its power spectral density as $S_w(f)$, then the received noise power spectral density is

$$S_z(f) = S_w(f) \times \sum_{m=1}^{M} \left| G_m^2(f) \right| \tag{2.2.14}$$

where σ_d^2 is the variance of $d(n)$, $\sigma_d^2 = E\{d^2(n)\}$. If $\sigma_z^2 = \int_{-\infty}^{\infty} S_z(f)df$ is the variance of $z(nT)$, then the SNR can be expressed as:

$$SNR = \frac{\sigma_d^2 x_0^2}{\sigma_z^2} \tag{2.2.15}$$

Let $X(f)$ be a raised cosine spectrum, and $X(f) = |X(f)|$. Its time domain waveform satisfies the following condition for all $n \neq 0$:

$$x(nT) = 0 \tag{2.2.16}$$

Let the baseband transfer function of uplink be $F(f)$:

$$F(f) = G_0(f) \sum_{m=1}^{M} G_m(f) C_m(f) \tag{2.2.17}$$

The no-ISI condition means that the baseband transfer function $F(f)$ must satisfy

$$F(f) = X(f) \tag{2.2.18}$$

From(2.2.13), the variance σ_d^2 is:

$$\sigma_d^2 = \frac{E}{\int_{-\infty}^{+\infty} \left|G_0^2(f)\right| df} \tag{2.2.19}$$

From(2.2.17) and (2.2.18), it follows that:

$$\sigma_d^2 = \frac{E}{\int_{-\infty}^{+\infty} \frac{X^2(f)}{\left|\sum_{m=1}^{M} G_m(f)C_m(f)\right|^2}} \tag{2.2.20}$$

From the definition of noise variance σ_z^2 and equation (2.2.14), σ_z^2 can be expressed as:

$$\sigma_z^2 = \int_{-\infty}^{+\infty} S_w(f) \sum_{m=1}^{M} \left|G_m^2(f)\right| df \tag{2.2.21}$$

Combining the equations(2.2.15), (2.2.20), and (2.2.21) , we have

$$SNR = \frac{Ex_0^2}{\int_{-\infty}^{+\infty} \frac{X^2(f)}{\left|\sum_{m=1}^{M} G_m(f)C_m(f)\right|^2} \int_{-\infty}^{+\infty} S_w(f) \sum_{m=1}^{M} \left|G_m^2(f)\right| df} \tag{2.2.22}$$

This function can be maximized with respect to the filters $G_m(f)$, m=1,2...M.

The downlink no ISI constraints is the same as the uplink case as shown in(2.2.12). The energy constraints on the downlink is

$$E = \sigma_d^2 \times \sum_{m=1}^{M} \int_{-\infty}^{\infty} \left|G_m^2(f)\right| df \tag{2.2.23}$$

It follows that in downlink case

$$\sigma_d^2 = \frac{E}{\sum_{m=1}^{M}\int_{-\infty}^{\infty}\left|G_m^2(f)\right|df} \tag{2.2.24}$$

The goal of the optimization is to maximize SNR with respect to filter parameters. In the uplink case, the SNR is expressed in equation(2.2.22). This expression of SNR already incorporates no-ISI condition, therefore, if it is maximized with respect to the filters $G_m(f), m = 1,...M$, the received filter will have maximized SNR with no ISI. As it is shown in [3], the optimization involves two-step procedure that uses Schwarz inequality in each step. Specifically, from Schwarz inequality

$$\left|\sum_{m=1}^{M} G_m(f)C_m(f)\right|^2 \le \sum_{m=1}^{M}\left|G_m^2(f)\right|\sum_{m=1}^{M}\left|C_m^2(f)\right| \tag{2.2.25}$$

where the equality holds for

$$G_m(f) = \alpha(f)C_m^*(f) \tag{2.2.26}$$

where factor $\alpha(f)$ is to be optimized. Using the inequality, we have

$$SNR \le \frac{Ex_0^2}{\int_{-\infty}^{\infty}\frac{X^2(f)}{\gamma(f)\sum_{m=1}^{M}\left|G_m^2(f)\right|}df\int_{-\infty}^{\infty}S_w(f)\sum_{m=1}^{M}\left|G_m^2(f)\right|} \tag{2.2.27}$$

where $\gamma(f) = \sum_{m=1}^{M}\left|C_m^2(f)\right|$ is the composite channel power spectral density.

Applying a second Schwarz inequality to the denominator in equation(2.2.27), we have

$$\int_{-\infty}^{\infty}\frac{X^2(f)}{\gamma(f)\sum_{m=1}^{M}\left|G_m^2(f)\right|}df\int_{-\infty}^{\infty}S_w(f)\sum_{m=1}^{M}\left|G_m^2(f)\right| \ge \left[\int_{-\infty}^{\infty}\frac{X(f)}{\sqrt{\gamma(f)}}\sqrt{S_w(f)}df\right]^2 \tag{2.2.28}$$

where the equality holds for

$$\frac{X(f)}{\sqrt{\gamma(f)S_w(f)}} = \beta\sum\left|G_m^2(f)\right| \tag{2.2.29}$$

where β is a constant.

Combining equations (2.2.26) and (2.2.29) we obtain the optimal value for $\alpha(f)$

$$\alpha(f) = \frac{1}{\sqrt{\beta}}\frac{1}{S_w^{1/4}(f)}\frac{\sqrt{X(f)}}{\gamma^{3/4}(f)} \tag{2.2.30}$$

From the no ISI constraint (2.2.18)

$$G_0(f) = \frac{X(f)}{\alpha(f)\gamma(f)} \tag{2.2.31}$$

and the constant β then follows from the energy constraint(2.2.13)

$$G_0(f) = K(f)\sqrt{X(f)}\gamma^{1/4}(f) \tag{2.2.32}$$

The uplink receiver filter is given below

$$G_m(f) = K^{-1}(f)\sqrt{X(f)}\gamma^{-3/4}(f)C_m^*(f), m = 1,\ldots M \tag{2.2.33}$$

where in both cases, the gain of the filter is given as

$$K(f) = \sqrt{\frac{E/\sigma_d^2}{\int_{-\infty}^{\infty}\sqrt{S_w(f)\frac{X(f)}{\sqrt{\gamma(f)}}}df}}S_w^{1/4}(f) \tag{2.2.34}$$

Again assuming white noise, $S_w(f) = N_0$, the factor $K(f)$ becomes constant and the same set of filters can be used for uplink and downlink transmissions. The factors K then become constant independent of frequency

$$K(f) = \sqrt{\frac{E/\sigma_d^2}{\int_{-\infty}^{\infty}\sqrt{N_0\frac{X(f)}{\sqrt{\gamma(f)}}}df}}N_0^{1/4} \tag{2.2.35}$$

And filters become

$$G_0(f) = K\sqrt{X(f)}\gamma^{1/4}(f) \tag{2.2.36}$$

$$G_m(f) = K^{-1}\sqrt{X(f)}\gamma^{-3/4}(f)C_m^*(f), m = 1, \ldots M \qquad (2.2.37)$$

In downlink case, the filter is calculated similarly by a double application of the Schwarz inequality. The resulting filter is given below.

For the downlink transmission filter

$$G_m(f) = K^{-1}\sqrt{X(f)}\gamma^{-3/4}(f)C_m(f), m = 1, \ldots M \qquad (2.2.38)$$

The downlink receiving filter is given as

$$G_0(f) = K\sqrt{X(f)}\gamma^{-1/4}(f) \qquad (2.2.39)$$

In both cases, the gain of the filter is

$$K = \sqrt{\frac{\int_{-\infty}^{\infty}\sqrt{N_0}\frac{X(f)}{\sqrt{\gamma(f)}}df}{E/\sigma_d^2}N_0^{-1/4}} \qquad (2.2.40)$$

3.0 Channel Model

The optimal filters in chapter 2 are derived with the assumption of known channel frequency response $C_m(f)$ and composite power spectrum $\gamma(f)$. In practice, this knowledge is not available. Measures must be taken to estimate channel characteristics with accuracy and speed. An estimated $\hat{C}_m(f)$ has to replace the assumed-known channel frequency response. The difference between the true and the estimated channel responses will cause performance degradation. To quantify this degradation, a channel model, and the corresponding estimator model was implemented in simulation. In this chapter, we discuss the channel model that is used in simulation. Some basic understanding of a multi-path channel will be needed to develop such model. Specifically, the baseband channel characterization will be considered next.

3.1 Baseband Characterization of a Multi-Path Channel

Communication signals have to be modulated onto a carrier frequency before passing through the channel. In simulation and analysis, however, this modulation step is normally not done. Instead, a complex-valued baseband equivalent channel and system model will be used.

Time-domain impulse response of the channel can be derived using statistical characterization of the channel [9]. The transmitted signal is represented in general as:

$$s(t) = \mathrm{Re}[s_l(t)e^{j2\pi f_c t}] \tag{3.1.1}$$

Where $s_l(t)$ is the baseband equivalent signal and f_c is the carrier frequency. With multiple paths, both the propagation delays and the attenuation factors are time-variant as a result of changes in the structure of the medium. Thus, the received bandpass signal may be expressed in the form

$$r(t) = \sum_p \alpha_p(t) s[t - \tau_{\mathrm{p}}(t)] \tag{3.1.2}$$

Where $\alpha_{\mathrm{p}}(t)$ is the attenuation factor for the signal received on the p^{th} path and $\tau_{\mathrm{p}}(t)$ is the associated path delay. Substitution (3.1.1) yield

$$r(t) = \mathrm{Re}(\{\sum_p \alpha_{\mathrm{p}}(t) e^{-j2\pi f_c \tau_{\mathrm{p}}(t)} s_l[t - \tau_{\mathrm{p}}(t)]\} e^{j2\pi f_c t}) \tag{3.1.3}$$

If we denote the quantity $\alpha_{\mathrm{p}}(t) e^{-j2\pi f_c \tau_{\mathrm{p}}(t)}$ as $c_p(t)$, then the above equation becomes

$$r(t) = \mathrm{Re}(\{\sum_p c_p(t) s_l[t - \tau_{\mathrm{p}}(t)]\} e^{j2\pi f_c t}) \tag{3.1.4}$$

Therefore, the low-pass equivalent received signal is

$$r_l(t) = \sum_p c_p(t) s_l[t - \tau_{\mathrm{p}}(t)] \tag{3.1.5}$$

Since $r_l(t)$ is the response of an equivalent low-pass channel to he equivalent low-pass signal $s_l(t)$, it follows that the low pass channel impulse response is

$$c(\tau, t) = \sum_p c_p(t) \delta[\tau - \tau_{\mathrm{p}}(t)] \tag{3.1.6}$$

$c(\tau, t)$ represents the impulse response of the channel at time t. This is the base-band equivalent impulse response for channels that contains discreet multi-path components.

The channel base-band complex gain $c_p(t)$ is a phasor having real-valued amplitude $|c_p(t)|$ and phase $2\pi f_c \tau_p(t)$. In practical channels, the amplitude $|c_p(t)|$ and $\tau_p(t)$ does not change very fast. However, the phase term $2\pi f_c \tau_p(t)$ will change by 2π whenever τ_p changes by $1/f_c$, which is a very small number. Consequently, the phase term $2\pi f_c \tau_p(t)$ will go through very rapid change during the transmission of signals through acoustic channels. This rapid change of phase causes the received signal to behave almost randomly. To properly model this random variation, the channel complex gain $c_p(t)$ can be modeled as random processes in t. Before we delve into developing random process representations of $c_p(t)$, let us first construct a basic multi-path channel where channel characteristics are time-invariant.

3.2 Time-Invariant Channel Model

As first approximation, we will develop a model of multi-path channels that are time-invariant. Hence, all channel characteristics (complex gains, transfer functions, etc.) will be based on geometry alone.

A geometric model of multi-path formation was used. Figure 3-1 shows the model schematic (not to scale). In shallow water transmissions, both surface and bottom reflections must be considered. Each channel has a direct path, and

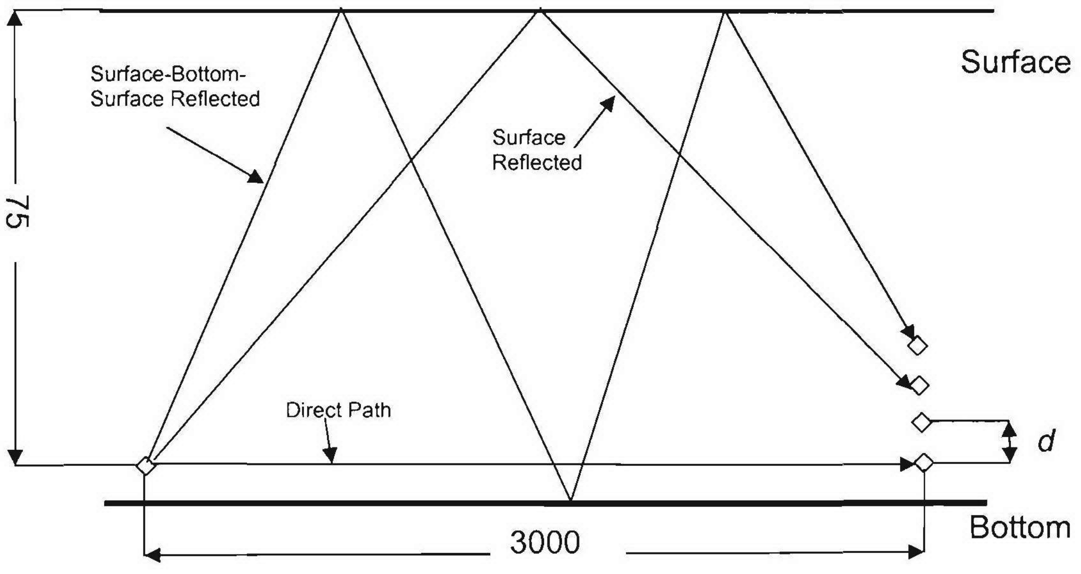

Figure 3-1 Channel geometrical model (not to scale)

a number of surface reflected and surface-bottom-surface reflected paths. The single element and multiple elements are located near the bottom in our example. If the distance between the receiving elements is large enough comparing to water depth, a planar wave assumption can be made. We can justify the planar wave assumption by the following arguments.

In our model, the range is 3000m; the depth is 75m. The distance between array elements is set to half wavelength in our model. For our model, we will be using carrier frequency of 15kHz. At nominal sound speed of 1500m/s, the half-wave length is $\lambda/2 = 0.05m$. With distance of 3000m, the target angle difference from one array element to the other is ~0.001 degrees. If we have an array size of 30, the target angle difference between the top element and the bottom element at half-wavelength spacing is only ~0.03 degrees. With this angle difference, the multiple-element array practically appears as a point when looking from the single element side. Therefore, the main axes of waves arriving to the array will be almost parallel to each other. Consequently, we can view those waves as planar if we assume two-dimensional acoustic wave propagation.

Assuming planar wave pattern, θ_p is the wave angle of arrival as shown in Figure 3-2. One major component of path delay is due to the path length differences, which we designated as delay τ_p. The other component is due to non-zero angle of arrival for paths other than the direct path. This delay will cause a phase shift between array elements. Let us denote the distance between array elements as d, which

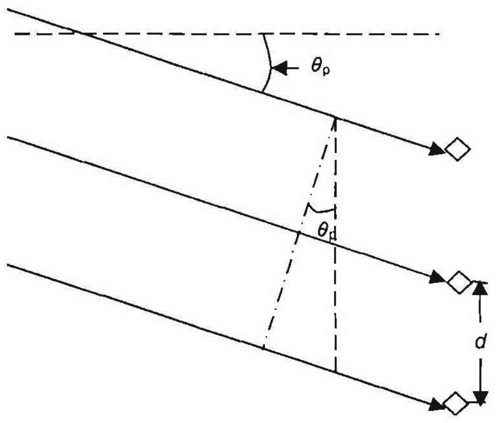

Figure 3-2 Angle of arrival

is normally set at half wavelength of the sound wave. The difference in arrival time is $\frac{d\sin(\theta_p)}{\lambda_c}$ as shown in

Figure 3-2.

This phase delay can be computed using

$$\phi_p = 2\pi(\frac{d\sin\theta_p}{\lambda_c}) \tag{3.2.1}$$

where $\lambda_c = \frac{c}{f_c}$. *C=1500 m/s* is the nominal sound speed used in our calculation.

Since the extra distance traveled will be directly related to the number of array element in between, the phase delay at mth array elements is $(m-1)\phi_p$, where $m = 1,...M$.

Attenuation in acoustic channel includes both refection attenuation and water-absorption attenuation. Consequently, the magnitude of path gain c_p is related to the attenuations caused by reflection and absorption. The magnitude of this path gain can be computed using path length l_p as

$$|c_p| = \frac{\Gamma_p}{\sqrt{A(l_p)}} \tag{3.2.2}$$

where Γ_p is the loss factor due to reflection. It is chosen as $\sqrt{2}$ for a single bottom reflection. *$A(l_p)$* is the nominal acoustic propagation loss, $A(l_p) = l_p^k(a(f_c))^{l_p}$. Assuming practical spreading, *k=1.5*. *$a(l_p)$* can be calculated using Thorp equation

$$10\log a(f_c) = \frac{0.11f_c^2}{(1+f_c^2)} + \frac{44f_c^2}{(4100+f_c^2)} + 2.75\times10^{-4}f_c^2 + 0.003 \tag{3.2.3}$$

where f_c is in kHz and *$a(f_c)$* is given in dB/km. This equation is valid at or around *f_c=15 kHz,* which is the carrier frequency chosen for the model simulation.

An impulse transmitted over a time-varying multi-path channel will appear at the receiver as a train of pulses due to multi-path. For our model, the multi-path channel impulse response is shown in the top graph of Figure 3-3. The impulse response shows path strength as function of delay. The first impulse arrival time is taken as the reference time of zero. The second and third arrivals are at 2.5ms and 10ms, respectively. Thus, the channel multi-path spread is 10ms. The bottom graph shows the angles of arrival of the three paths.

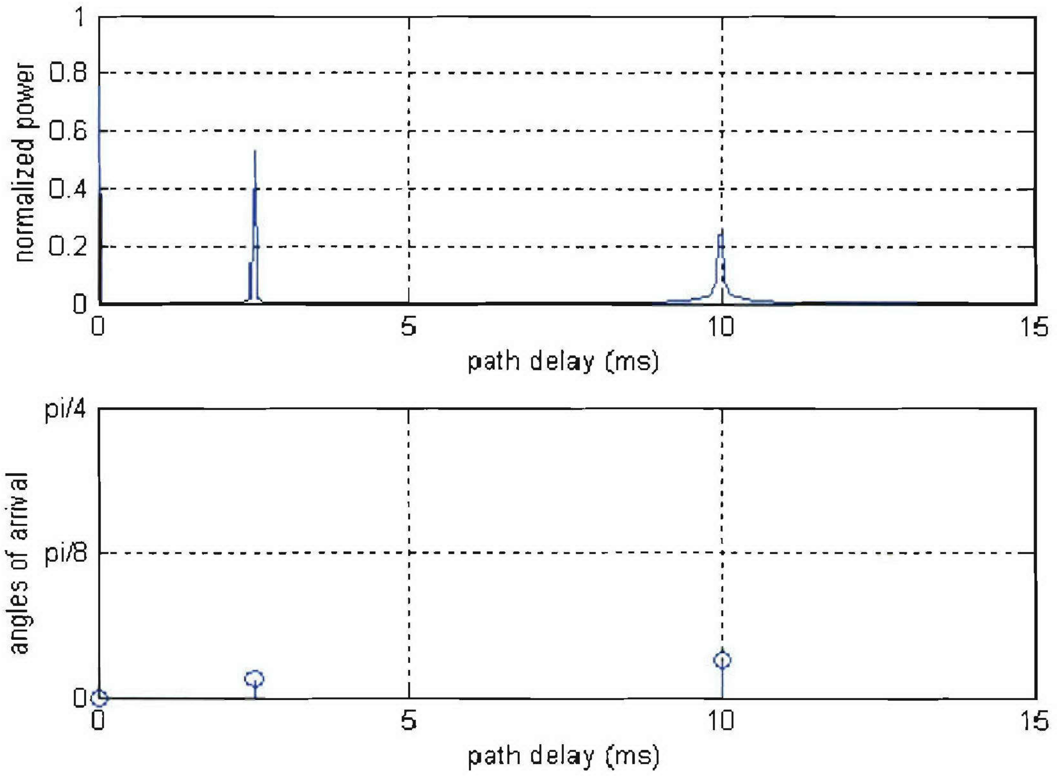

Figure 3-3 Multi-path properties of the channel

Top graph in Figure 3-3shows the magnitude of the path gain $|c_p|^2$. The path gain c_p is complex valued with phase $\angle c_p = -2\pi f_c \tau_p$, where τ_p is the path delay. We can express the path gain in complex form as $c_p = |c_p| e^{-2\pi f_c \tau_p}$.

For an M-element array, the channel transfer function can be expressed as

$$C_m(f) = \sum_{p=0}^{P-1} c_{m,p} e^{-j2\pi f \tau_p} \qquad (3.2.4)$$

where the path gains $c_{m,p}$ are obtained from the original path gains c_p by adding a phase shift

$$c_{m,p} = c_p e^{-j(m-1)\phi_p}, m = 1, \ldots M, p = 0, \ldots P-1 \qquad (3.2.5)$$

With time-invariant channel characterization in place, we can proceed to develop time-varying model which will be used in simulation. The time-varying behavior is modeled as a random process due to unpredictable nature of practical channels.

3.3 Time-Varying Channel

The time-varying channel transfer functions are given by

$$C_m(f,t) = \sum_{p=0}^{P-1} c_{m,p}(\mathrm{t}) e^{-j2\pi f \tau_{\mathrm{p}}(t)} \qquad (3.3.1)$$

where $c_{m.p}(t)$ is the time-varying channel complex gain corresponding to that given in equation (3.2.5), $c_{m,p}(t) = c_p(t) e^{-j(m-1)\phi_p} = | c_p(t) | e^{-j2\pi f_c \tau_p(t)} e^{-j(m-1)\phi_p}$.

In the time domain, the impulse responses of the channels are

$$c_m(\tau,t) = \sum_{p=0}^{P} c_{m.p}(t) \delta(\tau - \tau_p(t)) \qquad (3.3.2)$$

The complex gain $c_p(t) = |c_p(t)|e^{-j2\pi f_c \tau_p(t)}$ varies rapidly due to phase change. The magnitude $|c_p(t)|$ does not change significantly during the packet transmission interval; the delay $\tau_p(t)$ changes slowly as well and we can assume that $\tau_p(t) \approx \tau_p$. Thus $\tau_p(t)$ can be approximated by a constant delay calculated from geometry. The equation (3.3.2) becomes

$$c_m(\tau,t) \cong \sum_{p=0}^{P-1} c_{m,p}(t)\delta(\tau - \tau_p) \tag{3.3.3}$$

Although delay can be approximated as a constant, the phase term $2\pi f_c \tau_p$ can change significantly with very small change of delay. This is because large f_c will make even the smallest variation in delay a significant phase change. Therefore, the complex gain $|c_p(t)|e^{-j2\pi f_c \tau_p(t)}$ will undergo rapid changes even with the slow changing $|c_p(t)|$ and $\tau_p(t)$.

The complex gains $c_p(t)$ varies rapidly in an unpredictable manner, and it is consequently modeled as a random process. Furthermore, in Rayleigh fading model, $c_p(t)$ is modeled as complex-valued Gaussian random process in the t variable. i.e. $c_p(t) \sim N(0, \sigma_p^2)$. From equation (3.2.5), it follows that $c_{m,p}(t)$ is also a Gaussian random process.

For a given transmission medium with multi-path multiple channels, these complex gains $c_{m,p}(t)$ will form an *m*p* matrix. This matrix along with the path delay will completely represent the communication channels in our model. Let us call the *m by p* matrix the transmission matrix.

$$C(t)=\begin{pmatrix} c_{11}(t) & \dots & c_{1p}(t) \\ \vdots & \ddots & \vdots \\ c_{m1}(t) & \cdots & c_{mp}(t) \end{pmatrix} \tag{3.3.4}$$

This transmission matrix represents the complete knowledge of multi-path-multiple-channel communication channels. In discreet time, the transmission matrix can be written as

$$C(n\delta t)=\begin{pmatrix} c_{11}(n\delta t) & \dots & c_{1p}(n\delta t) \\ \vdots & \ddots & \vdots \\ c_{m1}(n\delta t) & \cdots & c_{mp}(n\delta t) \end{pmatrix} \tag{3.3.5}$$

where δt is the sampling interval in time.

3.3.1 Model of the Channel Variation

When implementing channels from the geometric model, $c_{m,p}$ is constructed from physical values such as delay time, arrival angle, attenuation etc. With the value of $c_{m,p}$ known, all channels can be simulated and all filter values can be calculated. Filters that are calculated based on perfectly known channel $c_{m,p}$ will produce received signal SNR that agrees very closely to the theory predicted values. However, in practical communications scenarios, the channels are not perfectly known. The channel coefficients have to be estimated using various means. The resulting $\hat{c}_{m,p}(t)$ is used to calculate filter values. Filters that are calculated based on estimated channel coefficients $\hat{c}_{m,p}(t)$ will generate errors. The goal of the simulation is to simulate system performance int eh presence of channel estimation errors, and to quantify the performance degradations from the ideal.

3.3.2 Auto Regressive Model

The channel variations can be modeled in a variety of ways. Auto regressive model represents one-way of obtaining the randomly varying $c_{m,p}(t)$. Auto regressive model recursively generates the process at discrete intervals in time starting with an initial value. Two types of auto regressive models, AR1 and AR2 will be considered.

In AR1 model, instead of deterministic c_p, we generate recursively a sequence of $c_p(n\delta\mathrm{t})$ as sampled points of $c_p(t)$. The value of $\delta\mathrm{t}$ defines the resolution of $c_p(t)$: the smaller the value of $\delta\mathrm{t}$, the finer the resolution. In order to capture the variability of the channel, $\delta\mathrm{t}$ must be much less than the channel coherence time. In most underwater acoustic channels, the coherence time is on the order of 1 second or more. Therefore, time resolution must be less than 0.1 second (In our model, we choose $\delta\mathrm{t}$ equaling symbol time interval, which is $1/5000 = 0.0002s$). The recursive equation of AR1 model is

$$c_p(n\delta\mathrm{t}+\delta\mathrm{t})=\mathrm{a}_0 c_p(n\delta\mathrm{t})+\xi_p(\mathrm{n}\delta\mathrm{t}),\ \mathrm{n}=0,1,2... \qquad (3.3.6)$$

where a_0 is a real constant that determines how fast the channel changes and $\xi_p(n\delta\mathrm{t})$ is the sequence of sampled points of a complex-valued, zero-mean white Gaussian random process that is independent of $c_p(n\delta\mathrm{t})$.

a_0 is related to the Doppler spread of the channel which determines the rate of time variation. If 3dB bandwidth of Doppler Spectrum is $2f_d$, then

$$a_0 = e^{-2\pi f_d |\delta t|} \tag{3.3.7}$$

$\xi_p(n\delta t)$ variance is calculated as

$$\sigma_{\xi_p}^2 = (1 - a_0^2)\sigma_p^2 \tag{3.3.8}$$

Where $\sigma_p^2 = E\left\{|c_p(t)|^2\right\}$ is the power of the path channel coefficients.

The initial term of the $c_p(n\delta t)$ is the same that we used in the deterministic model. It is calculated using geometric relationships. With that initial value, we can recursively generate the rest of sequence. The variance is also determined from this value as $\sigma_p^2 = |c_p(0)|^2$.

With random sequence $\{c_p(n\delta t)\}$ generated, the next step is to generate random matrices **C**. In the deterministic model, **C** is an *m by p* matrix with elements consisted of complex gain $c_{m,p}$. (3.2.5) defines the relationship between $c_{m,p}$ and c_p. In a time-varying model, the matrix $C(n\delta t)$ is time-dependant, with elements values $c_{m,p}(n\delta t)$

$$c_{m,p}(0) = c_p(0)e^{-j(m-1)\phi_p}$$
$$c_{m,p}(\delta t) = c_p(\delta t)e^{-j(m-1)\phi_p} = (a_0 c_p(0) + \xi(0))e^{-j(m-1)\phi_p} = a_0 c_{m,p}(0) + \xi(0)e^{-j(m-1)\phi_p}$$
$$c_{m,p}(2\delta t) = c_p(2\delta t)e^{-j(m-1)\phi_p} = (a_0 c_p(\delta t) + \xi(\delta t))e^{-j(m-1)\phi_p} = a_0 c_{m,p}(\delta t) + \xi(\delta t)e^{-j(m-1)\phi_p}$$

.

.

.

Auto regressive model of order one (AR1) is conceptually simple, but the resulting time variation maybe overly pessimistic for a practical channel. It tends to exaggerate the changes that occur in the channel. Auto regressive model of order two generates a better-behaved channel which maybe closer to a realistic acoustic channel.

The AR2 model can be generated in discrete time using the recursion

$$c_p[(n+1)\delta t] = a_0 c_p[n\delta t] + a_1 c_p[(n-1)\delta t] + \xi_p[n\delta t] \tag{3.3.9}$$

The parameters a_0 and a_1 are real constants that can be calculated from the continuous time parameters ξ and f_n, with ξ being the damping coefficient and f_n the natural frequency which is related to Doppler frequency through

$$f_{3dB} = f_n \sqrt{(1-2\xi^2) + \sqrt{1+(1-2\xi^2)^2}} \tag{3.3.10}$$

Damping coefficient is dependant on the physical channel and can be either over-damped (>0.5) or under-damped (<0.5). The relationship between continuous-time parameters and discreet-time parameters are

$$\begin{aligned} a_0 &= 2e^{-\xi\omega_n T_s} \cos(\sqrt{1-\xi^2}\,\omega_n T_s) \\ a_1 &= -e^{-2\xi\omega_n T_s} \end{aligned} \tag{3.3.11}$$

In the expression (3.3.9), $\xi_p(n)$ is a white process with variance $\sigma_w^2 = \sigma_p^2(1 - a_0^2 - a_1^2 - 2a_0 a_1 \rho)$, where $\rho = a_0/(1-a_1)$ is the one-step correlation coefficient. And P is the power of output fading process

$$\sigma_p^2 = E\{|c_p(t)|^2\} \tag{3.3.12}$$

Thus, the sequence $\{c_p(n\delta t)\}$ can be generated from the recursive relation(3.3.9). We can then follow the same steps as in AR1 model to generate the 3-D matrix using the following

$$c_{m,p}(0) = c_p(0)e^{-j(m-1)\phi_p}$$
$$c_{m,p}(\delta t) = c_p(\delta t)e^{-j(m-1)\phi_p}$$
$$c_{m,p}(2\delta t) = (a_0 c_p(\delta t) + a_1 c_p(0) + \xi(\delta t))e^{-j(m-1)\phi_p} = a_0 c_{m,p}(\delta t) + a_1 c_{m,p}(0) + \xi(\delta t)e^{-j(m-1)\phi_p}$$
$$c_{m,p}(3\delta t) = (a_0 c_p(2\delta \mathrm{t}) + a_1 c_p(\delta t) + \xi(2\delta t))e^{-j(m-1)\phi_p} = a_0 c_{m,p}(2\delta t) + a_1 c_{m,p}(\delta t) + \xi(2\delta t)e^{-j(m-1)\phi_p}$$

.

.

.

3.4 Channel Estimation

There are two types of errors that arise in channel estimation. One is caused by measurement noises and the other is caused by the lag between the time of measurement and the time of transmission. Due to the low speed of the sound propagation underwater, time-lag induced error may be the dominant component at high SNR.

There are two main approaches to channel estimation. One is to assume a model for the channel and then use the model to derive an adaptive estimator. The other is not to assume any model and adaptively estimate channel from the received signal only. The first approach requires a reasonably correct model to be effective, while the second approach does not rely on any assumptions about the channel. However, the first approach will give us means to predict the channel, while the second approach will only allow estimation based strictly on observations. In what follows, we will concentrate on the model based predictor.

3.4.1 Model Based Predictive Estimator

Assuming that the dominant source of estimation error is time lag, we neglect the measurement noise. Therefore the main source of the error is channel variation over the time difference $\mathrm{t_d} = N_d \delta \mathrm{t}$, between the time when the signal pass through the channel and the time the estimation is performed. The MMSE channel gain estimate, given the previous observations, is

$$\hat{c}_p(n\delta t) = E\left\{c_p(n\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\right\} \tag{3.4.1}$$

If we assume an AR1 model of the channel variation and use discrete sample points $n\delta t$ to replace continuous time t, we have

$$\hat{c}_p(n\delta t) = E\left\{a_0 c_p(n\delta t) + \xi(n\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\right\} \tag{3.4.2}$$

Using the expected value equivalency equation, it follows

$$\hat{c}_p(n\delta t) = E\left\{a_0 c_p(n\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\right\} + E\left\{\xi_p(n\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\right\}$$

The second term in above equation is zero based on our noise assumption (zero mean white noise). Substitute (3.3.6) and treat mean of ξ_p as zero, we have

$$\hat{c}_p(n\delta t) = a_0 E\{a_0 c_p((n-1)\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\} \tag{3.4.3}$$

Using recursion, we have the delay estimation equation with $N_d \delta t$ delay

$$\hat{c}_p(n\delta t) = a_0^{N_d} E\left\{c_p((n-N_d)\delta t) \mid c_p((n-N_d)\delta t), \ldots c_p(\delta t), c_p(0)\right\} \tag{3.4.4}$$

The expected value of $c_p((n-N_d)\delta t)$ conditioned on *a priori* observed $c_p((n-N_d)\delta t)$ is itself. If follows then

$$\hat{c}_p(n\delta t) = a_0^{N_d} c_p((n-N_d)\delta \mathrm{t}) \tag{3.4.5}$$

From above equation, a sequence of $\{\hat{c}_p(n\delta \mathrm{t})\}$ can be generated based on sequence of $\{c_p(n\delta \mathrm{t})\}$.

In order to simulate the performance with the presence of the estimate delay, the signal is passed through channel with each possible $c_{m.p}(n\delta t)$ and the received signal is processed using filters calculated based on estimated $\hat{c}_{m.p}(n\delta t)$. The resulting SNR values are then averaged over all possible channel realizations. The final mean value is the expected SNR in the presence of estimation error caused by estimator delays.

If the AR2 model is used and we neglect the measurement noise, the estimator is

$$\hat{c}_p(n\delta t) = a_0\hat{c}_p((n-1)\delta t) + a_1 c_p((n-2)\delta t) \tag{3.4.6}$$

given perfectly delayed observations

$$\hat{c}_p((n-N_d)\delta t) = c_p((n-N_d)\delta t) \tag{3.4.7}$$

and

$$\hat{c}_p((n-N_d+1)\delta t) = c_p((n-N_d+1)\delta t) \tag{3.4.8}$$

3.4.2 Delayed Estimator

The second approach is not based on any particular model. The simplest form of such estimation is to estimate by using the previously observed channel complex gains

$$\hat{c}_p(n\delta t) = c_p((n-N_d)\delta t) \tag{3.4.9}$$

This is the simple time-delayed estimator which we will be using in our simulation.

The special case for this type of estimation is when the observation is instantaneous, i.e. there is no delay in estimated values and the true values. This is the ideal case where each estimate is

$$\hat{c}_p(n\delta t) = c_p(n\delta t) \tag{3.4.10}$$

This ideal case can be simulated simply by equating the true channel complex gains with the estimated value. For other delayed cases, the estimation delay will cause some errors.

4.0 Performance Analysis

4.1 Simulation Preliminaries

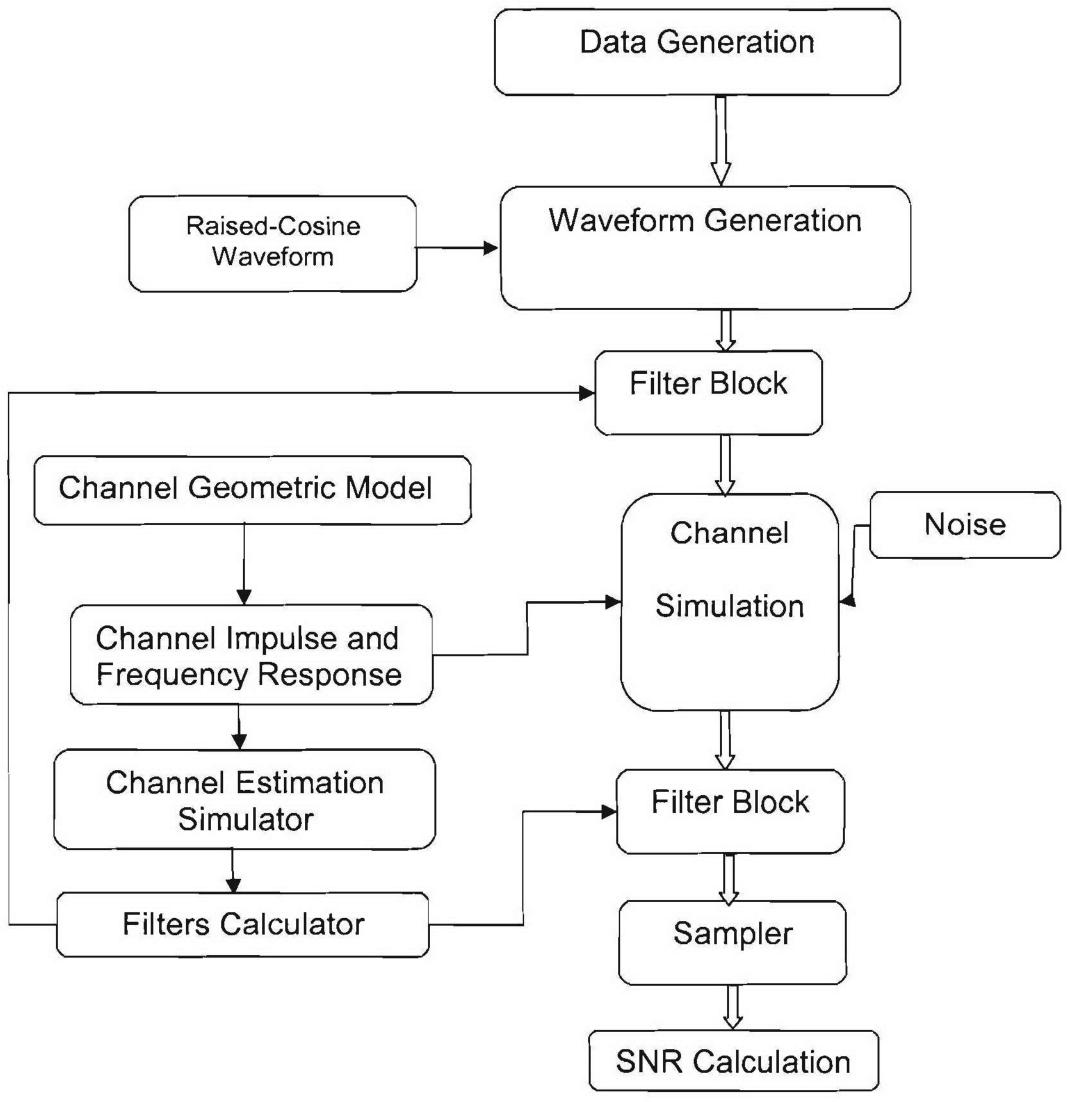

Figure 4-1 System Block Diagram

The system was implemented in MATLAB basic code. The MATLAB software block diagram is shown in Figure 4-1.

The data stream is randomly generated sequence of 0's and 1's. This data stream is then divided into 2-bit units and each is mapped into 4-PSK symbols on the complex plane. The symbol interval is T. The symbol stream is then fed into *wave generator* to generate complex waveforms that will be passing through the channels. The basic waveform that will carry the data signal is chosen to be the square-root-raised-cosine waveform with varied roll-off values α and truncation lengths. Figure 4-2 shows the example of the raised-cosine waveform and its frequency response for α values of 0.1, 0.5 and 1. The α value in the model is chosen to be 0.1, which corresponds very closely to an ideal low-pass filter.

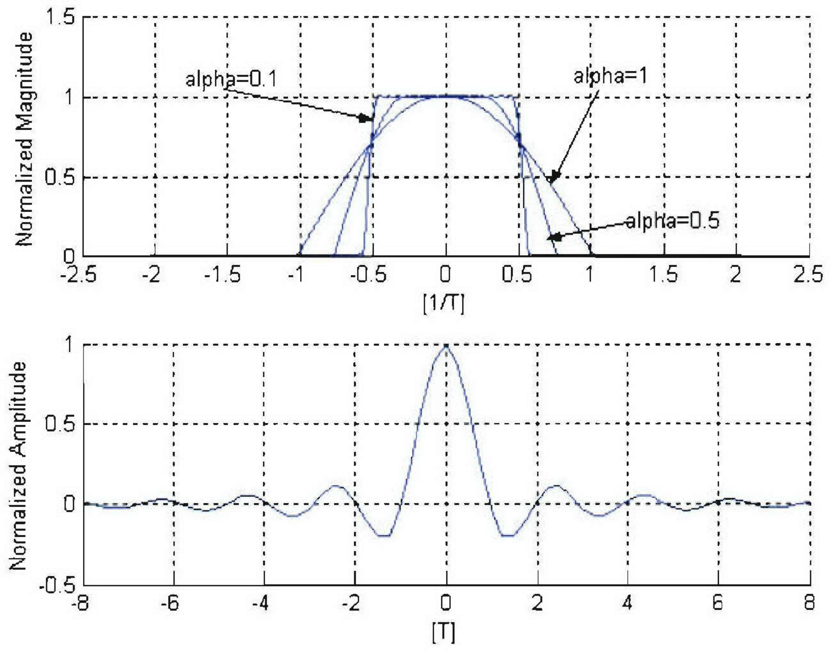

Figure 4-2 Raised cosine waveforms

A number of symbols generated comprise as one data packet. The data packet is show in Figure 4-3. The duration of entire data block is NT, where N is the number of symbols in one data packet. One data packet at time is then passed through time-varying channels.

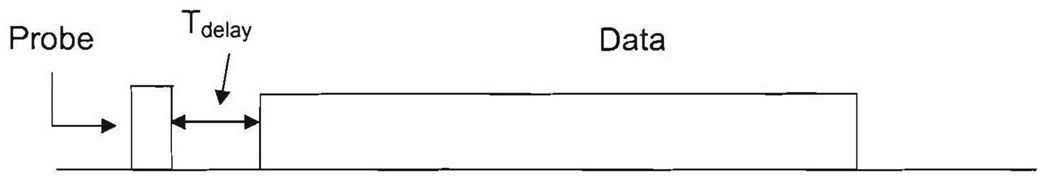

Figure 4-3 Data packet schematic

There are two filter blocks. One filter block will process before, and one after the complex waveforms passing through the channels. The first filter serves as the transmitting filter and the second filter is the receiving filter. Those two filters will be calculated depending on the direction (uplink or downlink) or the focusing scheme (either with no focusing, one-sided focusing or two-sided focusing). Inputs to the filter calculation block also include the channel information either deterministically calculated or stochastically estimated.

Channels are first constructed using the geometric model described earlier. Then the transfer function and impulse response of the channel are calculated inside *the channel impulse and frequency response* block. The resulting impulse response is used to simulate the transmissions through channels in the time-domain. The effect of varying channels is simulated using auto regressive model of order one (AR1) or auto regressive model of order 2 (AR2), which are implemented in *channel impulse and frequency response* block. Varied channel characteristics are generated recursively from the initial values which are calculated from the geometry.

Channel Estimation Simulator block simulates the estimation process using a non-model based, simple-delay estimator. The results of the estimation are then fed into *Filter Calculation* block to calculate filters. These calculated filter values are the inputted into the filter blocks to process the signal waveforms. Finally, the processed waveforms are sampled to generate received symbols, which are then processed to evaluate the signal-to-noise (SNR) ratios. The SNR for one channel realization is the time-average over the duration of the packet.

Table 4-1 summarizes model parameters for the simulation. Simulation runs are conducted using AR2 model to simulate channel variations. The damping factor of the channel is set to be $\xi = 0.5$. Three Doppler bandwidth conditions were simulated. They are 0.01Hz, 0.1Hz, and 1Hz. The estimator will estimate channels using the non-model based, simple time-delay scheme with three delay assumptions: ideal with no delay, 10ms delay and 1000ms delay.

Table 4-1Model parameters

Parameter Name	Parameter Value
Range of the model	*3000m*
Depth of the array elements	*75m*
Distance between arrays	$\lambda/2$
Number of channels	*4, 32*
Number of Paths/Channel	*3*
Nominal speed of sound	*1500m/s*
Sampling Frequency	*20000 Hz*
Carrier Frequency *	*15000 Hz*
Symbol Rate	*5000 Hz*
FFT Sample Points	*8192*
Length of Data Packet	*1000*
T (symbol interval)	*0.2ms*
Bit rate (4-PSK)	*10 kbps*
Doppler Bandwidth	*0.01 Hz,0.1Hz,1Hz*
Estimation Delay	*10ms, 1000ms*

*Note: Carrier frequency was used only to calculate the attenuation delay.

In each Doppler bandwidth condition, 4 channel and 32 channel performances were simulated and graphed for one-side focusing and time reversal. Simulation runs are conducted to compare the performance of one-side focusing to that of time-reversal in various conditions of Doppler bandwidth. The results are presented in the Sections 2, three and four. They are organized according to the channel Doppler bandwidth condition. We will start with the best condition first where Doppler spread is approximately 0.01 Hz.

The relationship between performance and array size were also investigated for the time-reversal and one-side focusing. The results are presented in Section 5.

Normalized Doppler spread is defined as the non-dimensional product of the channel Doppler spread and the estimation delay. It determines the difference between the estimated and the true channel characteristics. This fundamental relationship is demonstrated and discussed with simulation-run results in Section 6.

4.2 Doppler Bandwidth 0.01 Hz Channel Performance

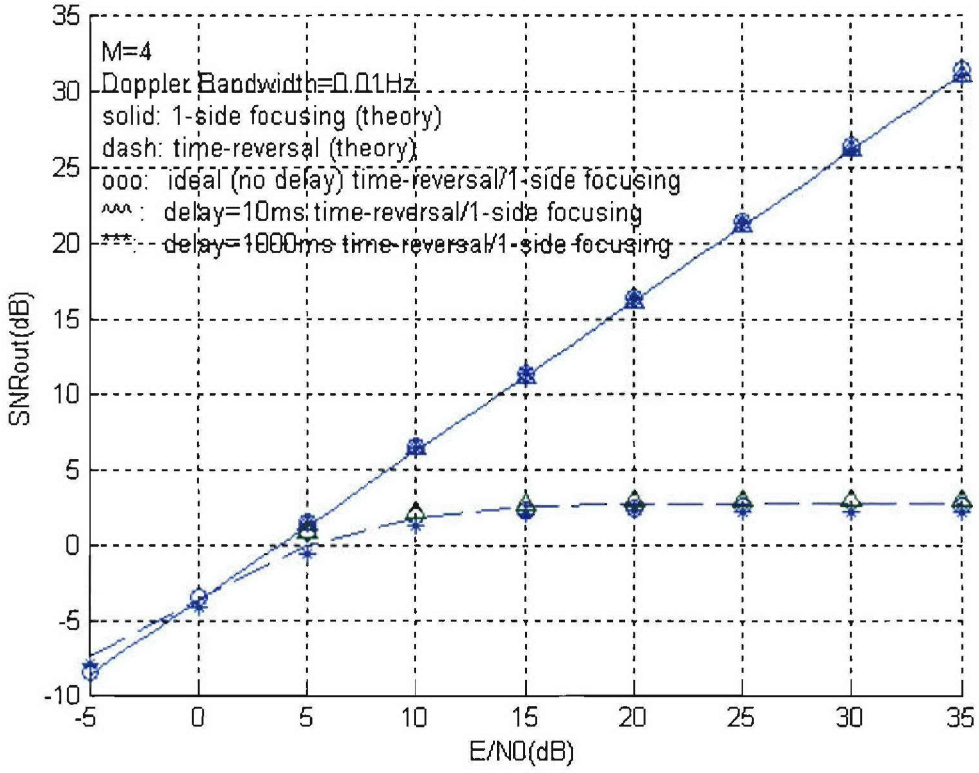

Figure 4-4 M=4 Doppler bandwidth 0.01 Hz performance

Doppler bandwidth 0.01 Hz corresponds to the channel coherence time of ~100s. This is the case where the channel is varying slowly (on the order of magnitude of minutes). Figure 4-4shows the performance for the 4-channel up/down link focusing as well as that of the time-reversal. Figure 4-5shows the performance of both schemes for the 32-channel case.

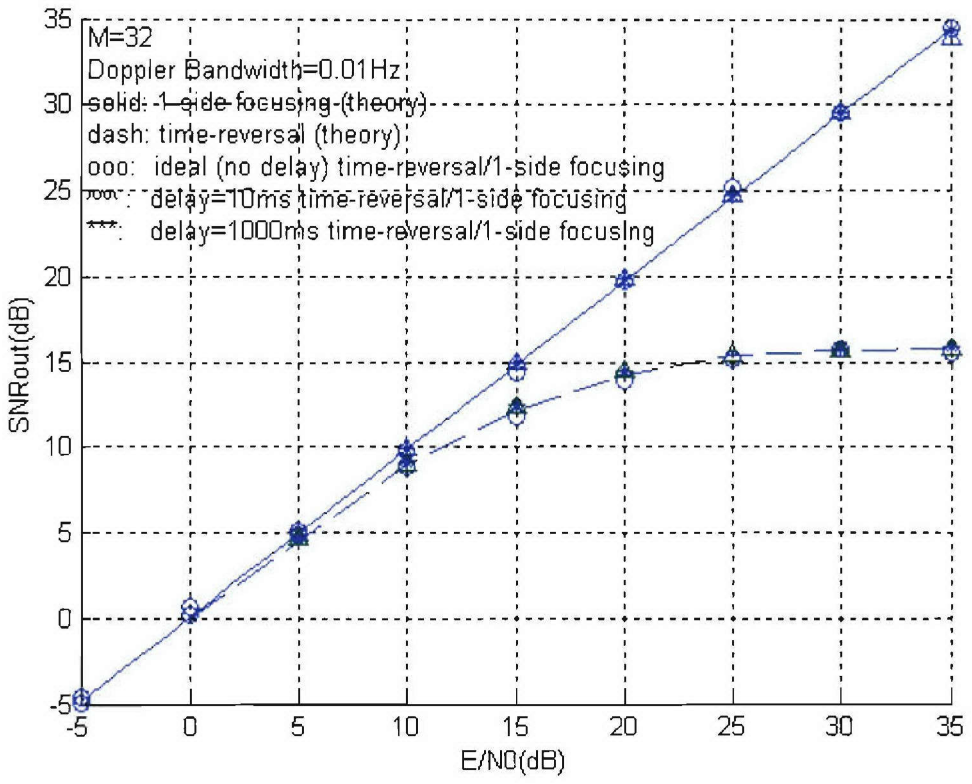

Figure 4-5 M=32 Doppler Bandwidth 0.01 Hz Performance

As may be expected, at the Doppler spread of 0.01 Hz, the performance does not deviate very much from the optimal results predicted by theory [3]. In this optimal condition, we can see that the performance of the spatiotemporal focusing is far superior to that of the time-reversal. Such superiority is more salient for the smaller array size (4-channel). With 4-channel array size at 20dB E/N0 noise level, focusing achieves 17dB SNRout while time-reversal only 3dB, which is below the level required for detection. One conclusion we can draw is that the time-reversal performance with a 4-channel array is not satisfactory whereas a system with focusing implemented performs quite well. Figure 4-6 and Figure 4-7 shows the scatter plots of the two schemes at input $E / N_0 = 20dB$. The time-reversal plot barely shows 4-PSK patterns while the focusing scheme clearly shows a clean pattern.

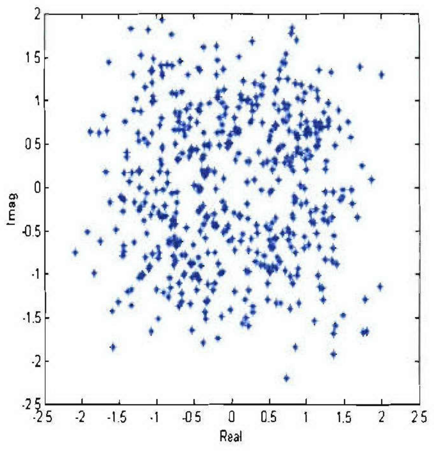

Figure 4-6 M=4 Time Reversal

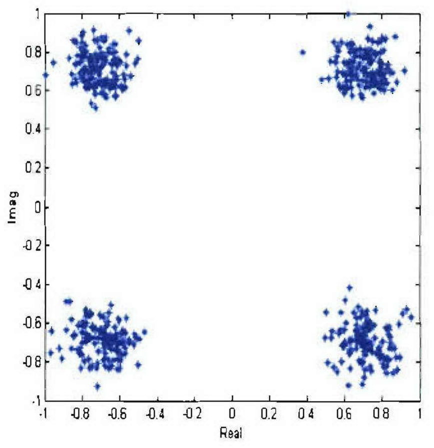

Figure 4-7 M=4 Focusing

In addition, time-reversal is shown to have performance saturation while spatiotemporal focusing does not. As observed in [3], the performance of time-reversal saturates whereas the performance of the optimal focusing will approach

∞ as E/N_0 approaches ∞. Such saturation in performance is shown for both 32-channel and 4-channel array sizes. Although the time-reversal performance increases significantly when the array size increases from 4-channel to 32-channel, it saturates at 15 dB. There is no such saturation phenomenon in focusing for both the 4-channel and 32-channel arrays.

Third, the performance of the 32-channel focusing does not increase very much (3-4dB) from the 4-channel case. Evidently, the spatiotemporal focusing performance is not sensitive to the array sizes because it has eliminated ISI even for the small size arrays. This can be advantageous when applications require limited array size. Time-reversal performance, on the other hand, depends solely on the array size as large performance increase is gained by going from 4-channel to 32-channel. We can conclude that the performance of focusing is generally superior to that of the time-reversal and even more so when the array size is small.

4.3 Doppler Bandwidth 0.1Hz Channel Performance

Doppler bandwidth of 0.1 Hz corresponds to the channel coherence time of 10s. In other words, channels with 0.1-Hz Doppler spread will vary on the order of magnitude of tens of seconds. With the propagation delay of 1 second, perhaps we will see some performance degradation due to such delays. Figure 4-8 shows the 4-channel array performance and Figure 4-9 shows the 32-channel array performance.

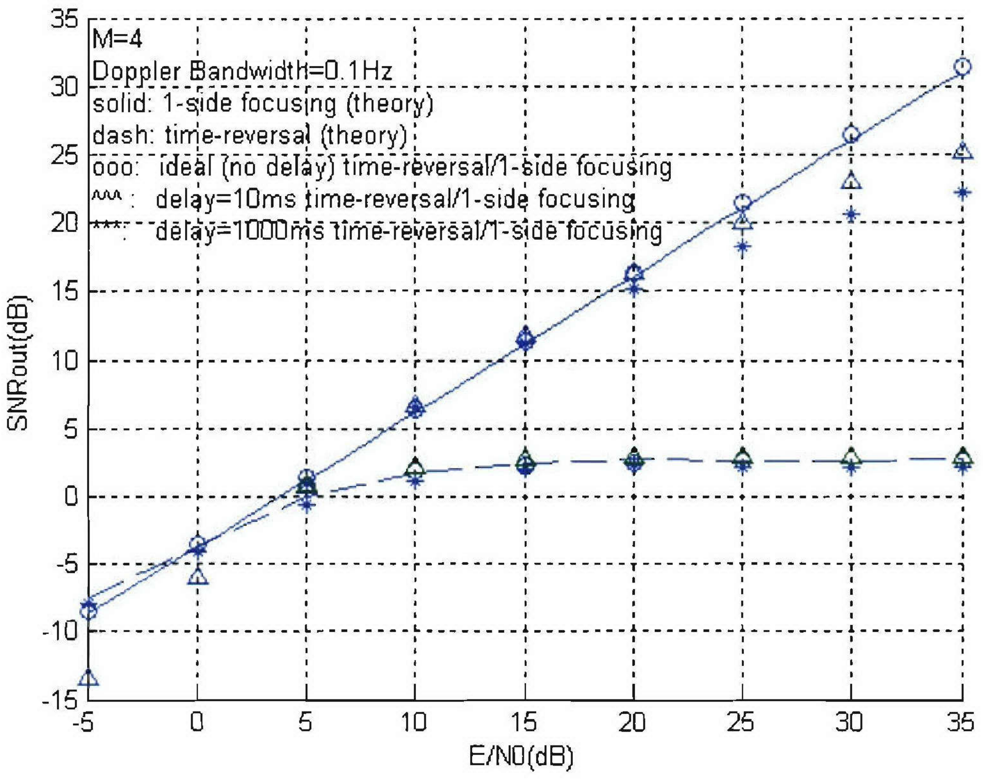

Figure 4-8 M=4 Doppler Bandwidth 0.1Hz Performance

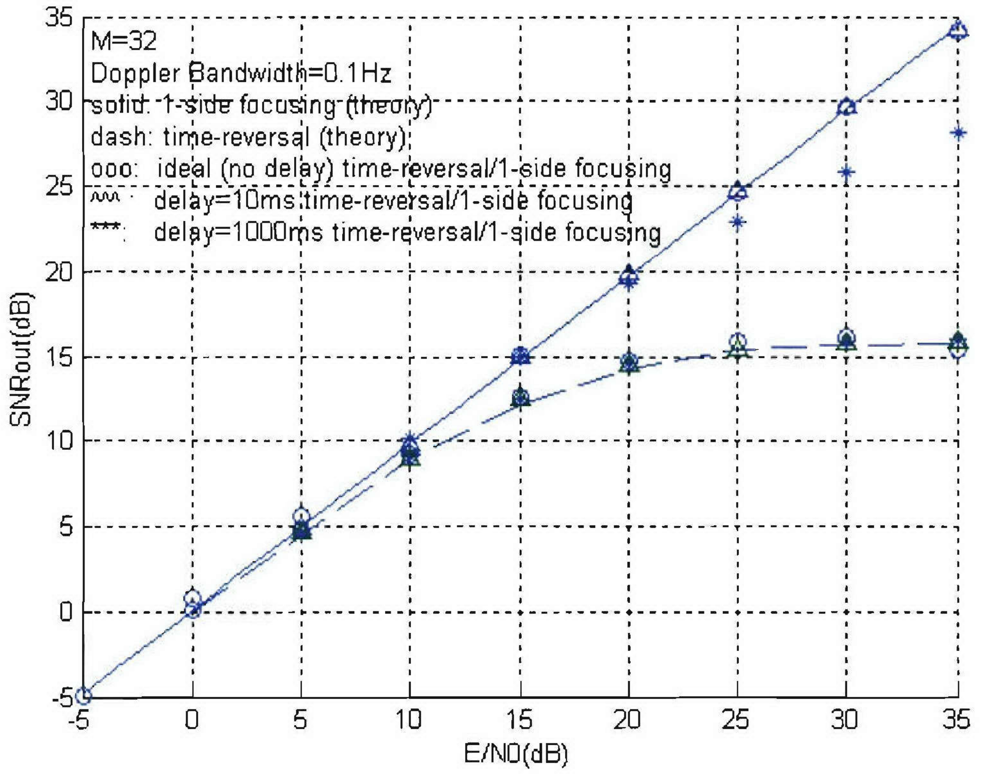

Figure 4-9 M=32 Doppler Bandwidth 0.1Hz Performance

First, we notice again there is no discernable performance saturation in the focusing performance for both the 4-channel and 32-channel arrays. Focusing performance in the presence of estimation error improves as E/N0 increases, albeit at a gradually reduced rate. This lack of performance saturation is in contrast with that of the time-reversal, which suffers saturation for both small and large size arrays.

From the result we also see that channel with Doppler spread of 0.1 Hz will affect the performance of spatiotemporal focusing for both the 4-channel and 32-channel arrays. The degradation increases with the delay. For 4-channel array, delay time of 10ms and 1s causes performance degradation. For 32-channel array, performance degradation can only be seen for the delay of 1s. There is no perceptible degradation for the10ms delay. Hence, the sensitivity to channel estimation error decreases with the array size. In addition, time-reversal appears less sensitive to the estimation error.

Even with the performance degradation, the focusing still outperforms the time-reversal by more than 10dB (4-channel array) for moderate input E/N0 levels. At a higher E/N0 levels, the margin is even larger. For the 4-channel array size, performance gained (>13dB) by focusing seems to fully justify the added complexity. The 32-channel array focusing performance, while degraded, is better than that of the time-reversal albeit at a lesser margin (5 for moderate E/N0 level and 10dB for higher E/N0 level). At a moderate E/N0 level, the performance advantage of the spatiotemporal focusing over the time-reversal with large array size may not entirely warrant the increased complexity that comes with spatiotemporal focusing.

4.4 Doppler Bandwidth 1Hz Channel Performance

Doppler bandwidth of 1Hz corresponds to channel coherence time of 1 second. This is the worst scenario in terms of channel Doppler spread condition. In this condition, the channel varies rapidly (order of magnitude of seconds). We should expect that with any delay on the order of one second (channel coherence time) will lead to severely degraded performance. Figure 4-10shows performance for the 4-channel array and Figure 4-11shows the performance for the 32-channel array.

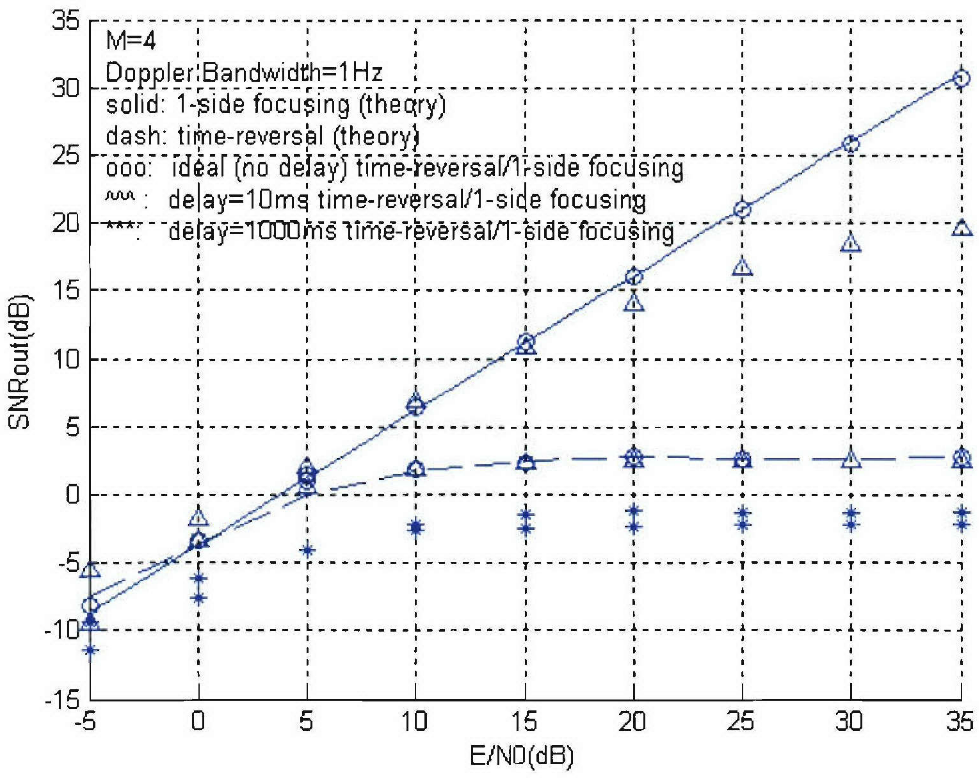

Figure 4-10 M=4 Doppler Bandwidth 1Hz Performance

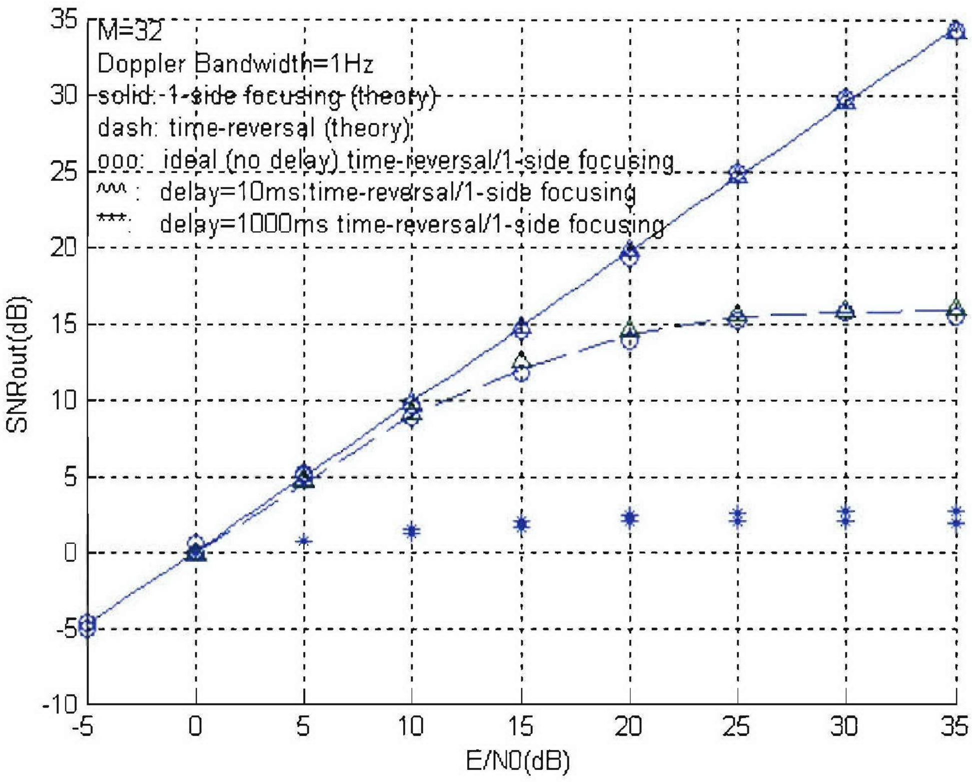

Figure 4-11 M=32 Doppler Bandwidth 1Hz Performance

We observe that that both the focusing and time-reversal suffers significant performance degradation for the 1s estimation delay case. This performance degradation is caused by the estimation delay that is on the order of channel coherence time. The error caused by this large delay is irreducible regardless of number of array elements used or input E/N_0. The dominant error here is caused by the estimation delay. At a high E/N_0 level and a large array size, the estimation error is entirely due to this irreducible error and the performance tends to saturate for both the time-reversal and focusing even as the input power increases to $+\infty$. Such saturation occurs for both small and large arrays. For 10ms estimation delay where the dominant error is not induced by such delay, focusing has no discernable performance saturation while time-reversal suffers saturation caused by dominant ISI-induced errors for both small and large array sizes.

Additionally, we will see that the masking effect also determines general shapes of performance curves. We have seen degraded performance curves gradually falling off theoretical predictions at high E/N0 level. Notice that at a low E/N0 level, the reduction isn't as large. This phenomenon is due to the same principle of dominant error masking. The total output noise level, which including both the noises that caused by estimation error and the inherent channel noise, will only reflect the dominant noise level. At very high E/N_0 level, the dominant noise source is the estimation error; at low E/N_0 level, the dominant noise source is the inherent channel noise. Therefore, we will not see much reduction in output SNR for lower E/N_0 level even with estimation delays. Due to the same reason, the time reversal scheme does not suffer significant reduction from its theoretical prediction in most cases because it has significant error caused by ISI (dominant noise) and ISI-induced error will mask the error caused by estimation error.

Third, again as in previous Doppler condition, one-side focusing performs better than the time-reversal even with the presence of modest estimation delay (10ms). In small array size (4 channel), this performance difference is quite large (>12dB), as can be seen from scatter plots shown in Figure 4-12 and Figure 4-13. They show that, at an array size of 4 and input $E/N_0 = 20dB$, the performance of one-side focusing is much superior to that of the time-reversal. For a larger array size, the difference is decreased somewhat (5dB) but still significant.

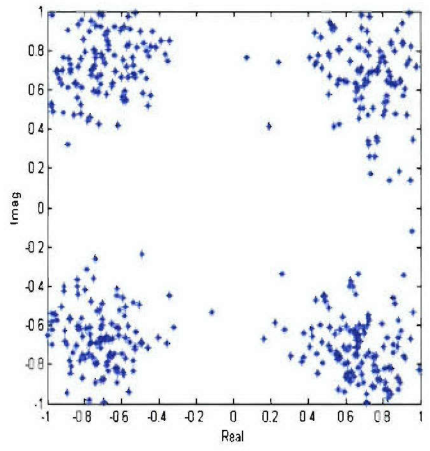

Figure 4-12 1-side focusing

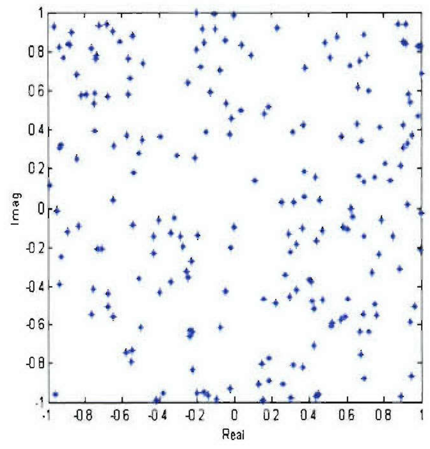

Figure 4-13 Time Reversal

4.5 Performance Sensitivity to Array Size

Simulation runs were conducted to investigate the effect of channel array size on performance. This set of simulation runs was executed using AR1 model to simulate the variation of the channel. Notice that the resulting output SNR is lower (3-4dB) than what would have been if AR2 model were used. Since the purpose is to investigate the general trend between array size and performance, the absolute values of output SNR will make little difference. Estimator used model-based predictive estimation scheme. The estimation delay is 10ms. The channel input E/N0 is 20dB.

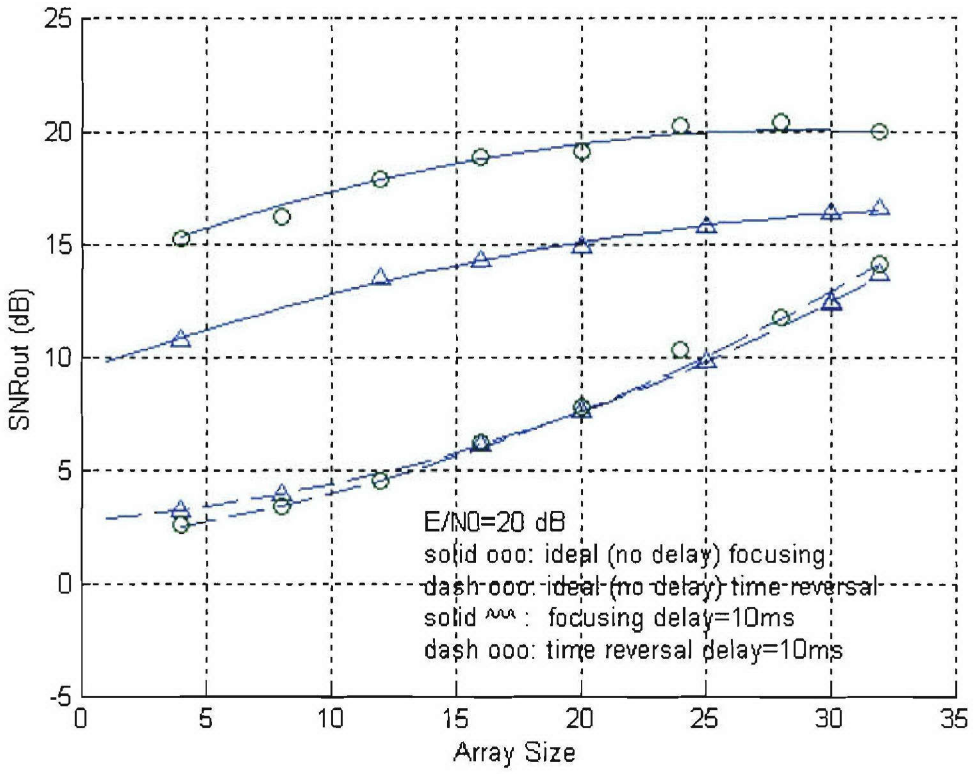

Figure 4-14 Performance vs. channel size at Doppler frequency 1Hz

Figure 4-14shows the effect of array size on output SNR. The input E/N_0 is at 20 dB. We can see that if there is no delay, the performance curve matches theoretical prediction [3]. And the performance degradation at 20dB is the same for all array sizes. In addition, it can be seen that the focusing is less sensitive to array size than time reversal. We gain 5dB when array size is increased from 4 to

35; on the other hand, we gain 12dB over the same range. In this regard, the focusing is superior to time reversal in that it does not depend on array size for optimal performance. Even with moderate array size (5), a very good output SNR (>12 dB) can be obtained. Time reversal, however, requires array size of almost 30 to obtain similar performance.

Furthermore, we observe that the more severe the limitation of array size is, the better suited for the spatiotemporal focusing technique over time reversal. In other words, the benefit of spatiotemporal focusing is more salient for smaller array size. We can see that at array size of 4, the performance advantages of focusing over time-reversal is 13dB, while at array size of 32 the performance of focusing over time-reversal is 5dB.

Finally, the performance of focusing will saturate at certain array size. From discussion by Stojanovic [3], we know that time reversal performance will also saturate at certain array size (40). Therefore, increasing array size over certain limit will not enhance the performance. The time reversal relies only on array size to improve performance. In this regard, spatiotemporal focusing is better than time-reversal because it does not rely solely on the array size to improve performance.

4.6 Normalized Doppler Spread

Fundamentally, performance degradation is caused by the difference between the estimated channel characteristics and that of the true channel. The difference can be caused by either fast-varying channel or channel estimation delay. A normalized Doppler spread is defined as $f_{d\_normal} = f_d T_{del}$, where f_d is the Doppler spread of the channel and T_{del} is the estimation delay. The performance level at different normalized Doppler spread is shown in Figure 4-15.

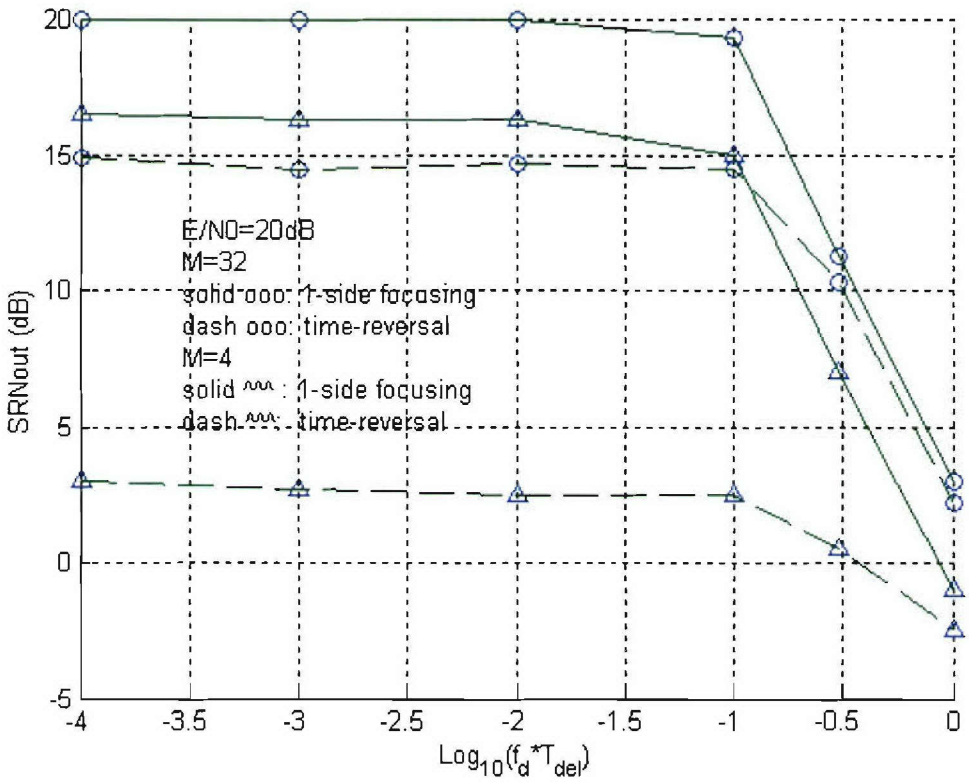

Figure 4-15 Performance vs. Normalized Doppler Spread

We can clearly see that when normalized Doppler spread is lower than 0.1 (-1 on logarithm scale), the performance levels are approximately equal. This is true for both time-reversal and spatiotemporal focusing. Both the 4-channel and 32 channel array display this property as well. Normalized Doppler spread of 0.1 corresponds to either 1Hz Doppler spread with 10ms delay or 0.1 Hz spread with 1 second delay. This non-dimensional normalized Doppler spread can be viewed as a measure of difference between the estimated and true channel coefficients.

It is the determining factor for the performance degradations.

5.0 Conclusions and Future Research

5.1 Future Work

The estimator implemented in both the AR1 and AR2 models were model based. Although adequate for analysis purposes, such estimators rely on the correctness of the model that describes the channel. In practice, the correct model is not known. Thus, a more practical way to approach the estimation problem is to devise an adaptive estimator algorithm that does not depend on the channel model. Instead, the estimator computes the channel complex gains based on the received waveforms and known training signals.

Experiments should also be conducted to validate spatiotemporal focusing techniques in the shallow water environment. The experiment design should also allow for testing of time reversal for comparison purpose. Some of the programs used in the simulation can be used for data processing in such an experiment. Examples of such programs are filter calculations, data and wave generation, data sampling and SNR comparison computations. A practical estimator algorithm is needed for the experimental data processing since we can not assume a channel model in a realistic experiment.

Finally, analytical expressions for output SNR in the presence of estimation errors should be derived. Analytical expressions for output SNR without estimator errors exists [3]. Following the same approach, analytical results may be derived. The expression may have a recursive or closed form. Only with analytical results can we properly conclude the initial chapter in spatiotemporal focusing investigation.

5.2 Conclusion

The goal of the spatiotemporal processing is to eliminate ISI while maximizing output SNR. One-side and two-side focusing techniques fulfill this goal by providing a flexible means to eliminate ISI while maximize output SNR. Simulation had been conducted to evaluate the performance of spatiotemporal focusing as well as the performance of time-reversal. Spatiotemporal focusing has proven to be a superior alternative to time-reversal. It achieves higher output SNR with modest array size; its performance does not saturate while that of the time-reversal does; it gives the system designer the flexibility to tailor complexity according to the application requirements. The performance improvement margin of the spatiotemporal focusing over the time-reversal is especially salient with the small array size. At the expense of moderate increasing of complexity, spatiotemporal focusing eliminates ISI thus achieves optimal performance level with array size large or small.

Fundamentally, performance degradation is caused by difference between estimated and true channel characteristics. The degree of difference can be represented by a non-dimensional normalized Doppler spread. When this normalized Doppler spread is below 0.1, the performance degradation from the ideal is negligible. From 0.1 to 1, the dominant error is increasingly caused by the channel variations and associated large estimation delays. Spatiotemporal focusing and time-reversal will both suffer additional performance degradations.

To fully realize the promise of spatiotemporal focusing approach, additional processing tools must be utilized when there is significant degree of difference between the estimated and true channel characteristics. When normalized Doppler spread is relatively large (>0.1), ISI increasingly ceases to be the dominant error contributor while the delay-induced error increasingly dominates. Its dominance makes the benefit of ISI-elimination unrealized (masking effect). Additional processing tools are needed to reduce estimation delays, especially for fast varying channels.

Predictive algorithm is a effective tool to reduce the estimation delays. With the predictive processing, estimation delay will depend on the quality of predictions and not on the estimation delay time. By reducing the effective estimation delays, the benefit of spatiotemporal focusing can be fully recovered.

References

[1] Edelmann, G.F.; Hodgkiss, W.S. et al. "Underwater Acoustic Communication Using Time Reversal". *OCEANS, 2001. MTS/IEEE Conference and Exhibition* , Volume: 4 , 5-8 Nov. 2001, Pages:2231 - 2235 vol.4.

[2] Rouseff, D.; Jackson, D.R. et al. "Underwater Acoustic Communication by Passive-Phase Conjugation: Theory and Experimental Results". *Oceanic Engineering, IEEE Journal of,* Volume: 26, Issue: 4, Oct. 2001 Pages: 821 – 831.

[3] Stojanovic M. "Retrofocusing Techniques for High Rate Acoustic Communications". *J. of. Acoustic Soc. Am.*, Vol. 117,No.3 Pt.1, March 2005.

[4] Stojanovic M, Catipovic J and Proakis J. G. "Adaptive Multi-channel Combining and Equalization for Underwater Acoustic Communications". *Journal of. Acoustic Soc.* Am. 94(3). Pt. 1. Sept. 1993.

[5] Kilfoyle, D.; Preisig J. and Baggeroer, A. "Spatial Modulation over Partially Coherent Multi-input/multi-output Channels". *IEEE Trans. Sig. Proc.* , Vol.51, No. 3, pp 794-804, Mar. 2003.

[6] Kuperman W. A., Hodgkiss W. S and Song H. C. "Phase conjugation in the ocean: Experimental demonstration of an acoustic time-reversal mirror". *Journal of Acoustic Soc.Am.* 103(1), pp.25-40, Jan 1998.

[7] Jackson D. , Ritcey J. A. and Fox W. L. J. et, all. "Experimental Testing of Passive Phase-Conjugation for Underwater Acoustic Communication". *Proceedings of 34th Asilomar conference,* pp. 680-683, 2000.

[8] Gomes J. and Barroso, V. "Time-Reversed Communication over Doppler-Spread underwater channels". *Proceedings ICASSP'02 Conference,* pp2849-2852 (III), 2002.

[9] Proakis G. John. *Digital Communications.* 4th Edition; McGraw-Hill Book Co. 2001. p801-803.

```
function simRunAR2(channel_number,side,dir,est_delay,res,N,snr_v,nbit)

range=3000;
depth=75;
E=1; %bit energy
Ns=4; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
M=4; %4-ary signal
n_mean=0;
trunc=16;
fd=1/Tsym;
array_d=(1500/fc)/2;
max_delay=0.01; %longest path delay time
alpha=0.1;

gtSingle=sqrtrcos(alpha,Ns,trunc);


%*******************************************************************************
%input block
% side=1;               %0:time-reversal 1:onesided focusing 2:twosided focusing
% dir=0;        %direction of the transimission 0 is uplink 1 is downlink
% channel_number=4;
%nbit=1000; %1000 symbols
Nd=nbit/2;
%snr_v=[-5 0 10 15 25 35];

%estimate machine
%est_delay=0;
ndiff=est_delay/res;
total=ndiff+N;

%*******************************************************************************

time=max_delay+Nd*Ns/fs;
min_df=1/time;
minFlength=fs/min_df
freq_length=2^nextpow2(minFlength);

padn=(freq_length-2*trunc*Ns)/2;
gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

%data generation
datastream=randint(2,Nd);
for m=1:nbit/2
   a=int2str(datastream(1,m));
   b=int2str(datastream(2,m));
   binstr=[a b];
   data(m)=bin2dec(binstr);
end;
dn=exp(j*2*pi.*((2*data+1)./2)./M);


[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
%[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,dummy,fc);

[Cmf3d,GMA2d,Cmf3dhat,GMA2dhat]=DTchannel_responseAR2(channel_number,CH_char,freq
_length,fs,fc,est_delay,res,N);
% Cmf3d=Cmf3d(:,:,ndiff+1:total);
% GMA2d=GMA2d(ndiff+1:total,:);
% Cmf3dhat=Cmf3dhat(:,:,1:total-ndiff);
% GMA2dhat=GMA2dhat(1:total-ndiff,:);

for n=1:length(snr_v)
   datapoint=snr_v(n)
   loop=0;
   for k=1:total-ndiff
      loop=loop+1
      CH_f=Cmf3d(:,:,k);
      GMA=GMA2d(k,:);
      CH_fhat=Cmf3dhat(:,:,k);
      GMAhat=GMA2dhat(k,:);
      N0=E/(10^(snr_v(n)/10));
      %using estimated value for filter generation also input Swf
      [G0_f,G_f,K_f]=filter_calculation(E,N0,GMAhat,CH_fhat,xf,gf,freq_length,dir,side);
      %time domain filter
      g0_t=ifft(fftshift(G0_f));
      for i=1:channel_number
         gf_t(i,:)=ifft(fftshift(G_f(i,:)));
      end;
      x=dn;
      if(dir==1)  %downlink
         U=waveGen(channel_number,dn,gf_t,Ns);
         R=channel(U,CH_f,E,snr_v(n),n_mean);
         Y=processor(R,g0_t,freq_length);
      elseif(dir==0) %uplink
         U=waveGen(channel_number,dn,g0_t,Ns);
         R=channel(U,CH_f,E,snr_v(n),n_mean);
         Y=processor(R,gf_t,freq_length);
      else
         disp('error in NewTest, up or down?');
      end;
      %sample
      delay=freq_length/2+1;
      y=sig_map(Y,delay,Ns,Nd,0);
      if(side==0)
%          %normalize energy increase due to channel addition
%          c=max(abs(ifft(GMA,freq_length)));
%          y=y./c;
         [snr_out(k,n),diff]=SNROUT(x,y);
      else
         [snr_out(k,n),diff]=SNROUT(x,y);
      end;
   end;
end;
GMAini=GMA2d(1,:);

if(total==1)
   temp=snr_out;
else
   temp=mean(snr_out);
end;

if(side==1)
   [SNRpredict]=SNRcompare1_noPlot(freq_length,xf,GMAini,snr_v,temp);
else
   [SNRpredict]=SNRcompare2_noPlot(freq_length,xf,GMAini,snr_v,temp);
end;


if(side==1)
   [SNRpredict]=SNRcompare1_noPlot(freq_length,xf,GMAini,snr_v,temp);
else
   [SNRpredict]=SNRcompare2_noPlot(freq_length,xf,GMAini,snr_v,temp);
end;

switch side
case 0
   if(dir==0)
      save output00.mat  snr_v snr_out SNRpredict -ascii -double -tabs
   elseif(dir==1)
      save output01.mat  snr_v snr_out SNRpredict -ascii -double -tabs
   else
      disp('error in simRun');
      save outputerr0.mat snr_v snr_out SNRpredict -ascii -double -tabs
   end;
case 1
      if(dir==0)
      save output10.mat  snr_v snr_out SNRpredict -ascii -double -tabs
      elseif(dir==1)
         save output11.mat  snr_v snr_out SNRpredict -ascii -double -tabs
      else
         disp('error in simRun');
         save outputerr1.mat snr_v snr_out SNRpredict -ascii -double -tabs
      end;
case 2
      if(dir==0)
      save output20.mat  snr_v snr_out SNRpredict -ascii -double -tabs
      elseif(dir==1)
         save output21.mat  snr_v snr_out SNRpredict -ascii -double -tabs
      else
         disp('error in simRun');
         save outputerr2.mat snr_v snr_out SNRpredict -ascii -double -tabs
      end;
otherwise
   save outputanyway.mat snr_v snr_out SNRpredict -ascii -double -tabs
end;
```

```
function [C,gama,Chat,gamahat]=DTchannel_responseAR2(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N

f_dop=1/2;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%nomalizing cp
s=sum((abs(CP)).^2);
factor=(1/n)/s;
CP=sqrt(factor).*CP;

xi=0.5 %damping factor
fn=f_dop/(sqrt((1-2*xi^2)+sqrt(1+(1-2*xi^2)^2)))
wn=2*pi*fn;

%a0=exp(-2*pi*f_dop*delT);

a0=2*exp(-xi*wn*delT)*cos(sqrt(1-xi^2)*wn*delT);
a1=-exp(-2*xi*wn*delT);

power=mean(CP.^2);
%sigma_p=power-(mean(CP))^2;
sigma_p=power;
rho=a0/(1-a1);
sigma_zita=(1-a0^2-a1^2-2*a0*a1*rho)*sigma_p;

r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
zita=r.*exp(j*theta);

ka0=a0^ndiff;
ka1=-abs(a1^(ndiff-1));


for m=1:n
```

```
  for p=1:path
     Phi(m,p)=exp(-j*(m-1)*P(p));
     Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
  end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;
CP3(:,:,2)=Coeff;

for i=1:delTlength
  recursion=i

%    A0=a0.^[i-1+ndiff-1:-1:0];
%    X=zita(1:ndiff+i-1);
%    CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%    CP3ealier(:,:,i+1)=a0*C
%    CP3hat(:,:,i+1)
  CP3(:,:,i+2)=a0*CP3(:,:,i+1)+a1*CP3(:,:,i)+Phi*diag(zita(i));
  CP3hat(:,:,i)=CP3(:,:,i);
end;

for i=1:N
  count=i
  for m=1:n
     C(m,:,i)=CP3(m,1,i+1+ndiff)*ones(size(f));
     Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
     for p=2:path
        C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
        Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

     end;
  end;
  gama(i,:)=zeros(1,length(f));
  gamahat(i,:)=zeros(1,length(f));
  for m=1:n
     gama(i,:)=gama(i,:)+(abs(C(m,:,i)).^2);
     gamahat(i,:)=gamahat(i,:)+(abs(Chat(m,:,i)).^2);
  end;

end;
```

```
function [Ch_char,CC]=channel_construc(n,R,D,d,fc,fs)
%cconstruct channel gemetrically and output CH_char, CC


%contruct channels
%n: channel number
%R: range between arrays
%D: depth of array
%d: array interval (vertical)


%CC: output is three dimensional array. row is path, column is [Attenuation Total_delay]
%third dimension channel
```

```
%CH_char: row dimension Path. Column dimension
[Attenuation,Phi_p,path_delay,delay_causedBy_Phi]
%total_delay=path_delay+delay_due_to_Phi_p

fc=fc/1000;
k=1.5;
c=1500;
r_fac=1/sqrt(2); %reflection factor
lamda=c/(fc*1000);
T=1/fs;

a_fc=0.11*fc^2/(1+fc^2)+44*fc^2/(4100+fc^2)+0.000275*fc^2+0.003;
a_fc=10^(a_fc/10);

%path one direct path
L1=R;
T1=1;
offset=L1/c;
tau1=L1/c-offset;
theta1=0;
CP1=T1/sqrt((L1/1000)^k*a_fc^(L1/1000));
phi1=2*pi*(d/lamda)*sin(theta1);
t1=d*sin(theta1)/c;

%path two surface reflection
L2=sqrt(R^2+(2*D)^2);
T2=r_fac;
tau2=L2/c-offset;
theta2=atan(2*D/R);
CP2=T2/sqrt((L2/1000)^k*a_fc^(L2/1000));
phi2=2*pi*(d/lamda)*sin(theta2);
t2=d*sin(theta2)/c;

%path three suface bottom surface refelection
L3=4*sqrt((R/4)^2+D^2);
T3=(r_fac)^2;
tau3=L3/c-offset;
theta3=atan(D/(R/4));
CP3=T3/sqrt((L3/1000)^k*a_fc^(L3/1000));
phi3=2*pi*(d/lamda)*sin(theta3);
t3=d*sin(theta3)/c;

CP=[CP1 CP2 CP3];
P=[phi1 phi2 phi3];
TAU_rel=[tau1 tau2 tau3];
tmp=[t1,t2,t3];

Ch_char=[CP;P;TAU_rel;tmp].';

for m=1:n
   CC(:,:,m)=[ CP1  tau1+(m-1)*t1;
            CP2  tau2+(m-1)*t2;
            CP3  tau3+(m-1)*t3;];
end;
```

```
function
[C,gama,Chat,gamahat]=DTchannel_response_alt_alt_alt(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N

f_dop=1/2;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%normalize cp
s=sum((abs(CP)).^2);
factor=(1/n)/s;
CP=sqrt(factor).*CP;

power=mean(CP.^2);

%sigma_p=power-(mean(CP))^2;
sigma_p=power;
a0=exp(-2*pi*f_dop*delT);
sigma_zita=(1-a0^2)*sigma_p;

r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
zita=r.*exp(j*theta);

kfac=a0^ndiff;
kfac=1;

for m=1:n
   for p=1:path
      Phi(m,p)=exp(-j*(m-1)*P(p));
      Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
   end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;
```

```
for i=1:delTlength
   recursion=i

%    A0=a0.^[i-1+ndiff-1:-1:0];
%    X=zita(1:ndiff+i-1);
%    CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%    CP3ealier(:,:,i+1)=a0*C
%    CP3hat(:,:,i+1)
   CP3(:,:,i+1)=a0*CP3(:,:,i)+Phi*diag(zita(:,i));
   CP3hat(:,:,i)=kfac*CP3(:,:,i);
end;

for i=1:N
   count=i
   for m=1:n
      C(m,:,i)=CP3(m,1,i+ndiff)*ones(size(f));
      Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
      for p=2:path
         C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
         Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

      end;
   end;
   gama(i,:)=zeros(1,length(f));
   gamahat(i,:)=zeros(1,length(f));
   for m=1:n
      gama(i,:)=gama(i,:)+abs(C(m,:,i).^2);
      gamahat(i,:)=gamahat(i,:)+abs(Chat(m,:,i).^2);
   end;
end;
```

```
function Y_t=channel(uu,C,E,snr_in_db,noise_mean)
%pass signal through channel specified by C
%input:
%uu: input signal. can be single row (uplink) or multi-row downlink
%C:  channel spec. row channel column: frequency sample
%E:  Energy per bit
%snr_in_db: Channel noise in dB
%noise_mean: channel noise mean

%output:
%Y_t: signal received on the other end with noise
SNR=10^(snr_in_db/10);
var=1/SNR;


[chn,f_l]=size(C);
[sn,sc]=size(uu);
%fac=noise_norm(uu,var,snr_in_db,chn,f_l); %normalize noise factor due to addtion effect
for m=1:chn
   if(chn>sn) %uplink case
      Cm_f=C(m,:);
      ct=ifftshift(ifft(fftshift(Cm_f)));
      ut=uu;
      y=conv(ct,ut);
      y=y(f_l/2+1:length(y));
      %generate and add noise
      r=randn(1,length(y)).*sqrt(var)+noise_mean;
      theta=rand(1,length(y))*2*pi;
      noise=r.*exp(j*theta);
 %       noise=noise*sqrt(fac);
      y=y+noise;
      Y_t(m,:)=y;
      disp('uplink channel');

   elseif(chn==sn) %downlink case
      Cm_f=C(m,:);
      ct=ifftshift(ifft(fftshift(Cm_f)));
      ut=uu(m,:);
      y=conv(ct,ut);
      y=y(f_l/2+1:length(y));
      Y_t(m,:)=y;
      disp('downlink channel');
   else
      disp('error in channel.m');
   end;
end;
if(chn==sn)  %downlink case noise
   Y_t=sum(Y_t);
   r=randn(1,length(Y_t)).*sqrt(var)+noise_mean;
   theta=rand(1,length(Y_t))*2*pi;
   noise=r.*exp(j*theta);
  %  noise=noise*sqrt(fac);
   Y_t=Y_t+noise;
end;
```

```
function [C,gama,Chat,gamahat]=DTchannel_responseAR20(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N

f_dop=1/20;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%nomalizing cp
s=sum((abs(CP)).^2);
```

```
factor=(1/n)/s;
CP=sqrt(factor).*CP;

xi=0.5 %damping factor
fn=f_dop/(sqrt((1-2*xi^2)+sqrt(1+(1-2*xi^2)^2)))
wn=2*pi*fn;

%a0=exp(-2*pi*f_dop*delT);

a0=2*exp(-xi*wn*delT)*cos(sqrt(1-xi^2)*wn*delT);
a1=-exp(-2*xi*wn*delT);

power=mean(CP.^2);
%sigma_p=power-(mean(CP))^2;
sigma_p=power;
rho=a0/(1-a1);
sigma_zita=(1-a0^2-a1^2-2*a0*a1*rho)*sigma_p;

r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
zita=r.*exp(j*theta);

ka0=a0^ndiff;
ka1=-abs(a1^(ndiff-1));

for m=1:n
   for p=1:path
      Phi(m,p)=exp(-j*(m-1)*P(p));
      Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
   end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;
CP3(:,:,2)=Coeff;

for i=1:delTlength
   recursion=i

%     A0=a0.^[i-1+ndiff-1:-1:0];
%     X=zita(1:ndiff+i-1);
%     CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%     CP3ealier(:,:,i+1)=a0*C
%     CP3hat(:,:,i+1)
   CP3(:,:,i+2)=a0*CP3(:,:,i+1)+a1*CP3(:,:,i)+Phi*diag(zita(i));
   CP3hat(:,:,i)=CP3(:,:,i);
end;


for i=1:N
   count=i
   for m=1:n
      C(m,:,i)=CP3(m,1,i+1+ndiff)*ones(size(f));
      Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
      for p=2:path
```

```
            C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
            Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

        end;
    end;
    gama(i,:)=zeros(1,length(f));
    gamahat(i,:)=zeros(1,length(f));
    for m=1:n
        gama(i,:)=gama(i,:)+(abs(C(m,:,i)).^2);
        gamahat(i,:)=gamahat(i,:)+(abs(Chat(m,:,i)).^2);
    end;

end;
```

```
function
[C,gama,Chat,gamahat]=DTchannel_response_alt_alt_alt10(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N


f_dop=1/20;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%normalize cp
s=sum((abs(CP)).^2);
factor=(1/n)/s;
CP=sqrt(factor).*CP;

power=mean(CP.^2);

%sigma_p=power-(mean(CP))^2;
sigma_p=power;
a0=exp(-2*pi*f_dop*delT);
sigma_zita=(1-a0^2)*sigma_p;

r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
```

```
zita=r.*exp(j*theta);

%kfac=a0^ndiff;
kfac=1;


for m=1:n
   for p=1:path
      Phi(m,p)=exp(-j*(m-1)*P(p));
      Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
   end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;

for i=1:delTlength
   recursion=i

%     A0=a0.^[i-1+ndiff-1:-1:0];
%     X=zita(1:ndiff+i-1);
%     CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%     CP3ealier(:,:,i+1)=a0*C
%     CP3hat(:,:,i+1)
   CP3(:,:,i+1)=a0*CP3(:,:,i)+Phi*diag(zita(:,i));
   CP3hat(:,:,i)=kfac*CP3(:,:,i);
end;


for i=1:N
   count=i
   for m=1:n
      C(m,:,i)=CP3(m,1,i+ndiff)*ones(size(f));
      Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
      for p=2:path
         C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
         Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

      end;
   end;
   gama(i,:)=zeros(1,length(f));
   gamahat(i,:)=zeros(1,length(f));
   for m=1:n
      gama(i,:)=gama(i,:)+abs(C(m,:,i).^2);
      gamahat(i,:)=gamahat(i,:)+abs(Chat(m,:,i).^2);
   end;

end;
```

```
function [C,gama,Chat,gamahat]=DTchannel_responseAR200(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N
```

```
f_dop=1/200;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%nomalizing cp
s=sum((abs(CP)).^2);
factor=(1/n)/s;
CP=sqrt(factor).*CP;



xi=0.5 %damping factor
fn=f_dop/(sqrt((1-2*xi^2)+sqrt(1+(1-2*xi^2)^2)))
wn=2*pi*fn;

%a0=exp(-2*pi*f_dop*delT);

a0=2*exp(-xi*wn*delT)*cos(sqrt(1-xi^2)*wn*delT);
a1=-exp(-2*xi*wn*delT);

power=mean(CP.^2);
%sigma_p=power-(mean(CP))^2;
sigma_p=power;
rho=a0/(1-a1);
sigma_zita=(1-a0^2-a1^2-2*a0*a1*rho)*sigma_p;



r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
zita=r.*exp(j*theta);

ka0=a0^ndiff;
ka1=-abs(a1^(ndiff-1));


for m=1:n
   for p=1:path
      Phi(m,p)=exp(-j*(m-1)*P(p));
      Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
   end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;
CP3(:,:,2)=Coeff;

for i=1:delTlength
   recursion=i

%     A0=a0.^[i-1+ndiff-1:-1:0];
%     X=zita(1:ndiff+i-1);
%     CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%     CP3ealier(:,:,i+1)=a0*C
%     CP3hat(:,:,i+1)
   CP3(:,:,i+2)=a0*CP3(:,:,i+1)+a1*CP3(:,:,i)+Phi*diag(zita(i));
   CP3hat(:,:,i)=CP3(:,:,i);
end;


for i=1:N
   count=i
   for m=1:n
      C(m,:,i)=CP3(m,1,i+1+ndiff)*ones(size(f));
      Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
      for p=2:path
         C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
         Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

      end;
   end;
   gama(i,:)=zeros(1,length(f));
   gamahat(i,:)=zeros(1,length(f));
   for m=1:n
      gama(i,:)=gama(i,:)+(abs(C(m,:,i)).^2);
      gamahat(i,:)=gamahat(i,:)+(abs(Chat(m,:,i)).^2);
   end;

end;
```

```
function [G0_f,G_f,K_f]=filter_calculation(E,N0,gama,ch_f,xf,gf,freq_l,char,flag)
%calculate filters for 2sided uplink and downlink
%char: uplink or downlink
%flag=1 for 1 sided
%flag=2 for 2 sided

[r,c]=size(ch_f);
shape=ones(r,1);
GAMA=shape*gama;
GF=shape*gf;

switch flag
case 2  %two sided

   Swf=N0;
   var_d=1;
   df=2*pi/freq_l;
```

```
  v=abs(xf)./sqrt(gama);
  Integral=(1/(2*pi))*sum(df*sqrt(Swf).*v);
  beta=(E/var_d)/Integral;
  KK_f=Swf^(1/4)*sqrt(beta);

  if(char==0)
    K_f=KK_f;
    disp('two-sided uplink filter');
  elseif(char==1)
    K_f=1/KK_f;
    disp('two-sided downlink filter');
  else
    disp('error! select either downlink or uplink');
  end;

  G_f=(1/K_f)*GF.*conj(ch_f)./GAMA.^(3/4);
  G0_f=K_f*gf./gama.^(1/4);
case 1   %one side
    x0=1;
    var_d=1;
    Swf=N0;
  if(char==0)  %uplink
    disp('one sided uplink filter');
    K=sqrt((E/var_d)/x0);

    G_f=1/K*GF.*conj(ch_f)./GAMA;
    %G0_f=fftshift(fft(gtSingle,freq_l));
    G0_f=gf;
    K_f=K;
  elseif(char==1)  %downlink
    disp('one-sided downlink filter');
    df=2*pi/freq_l;
    v=abs(xf)./gama;

    Integral=(1/(2*pi))*sum(df*Swf.*v);
    beta=(E/var_d)/Integral;
    K_f=Swf^(1/2)*sqrt(beta);

    G_f=K_f*GF.*conj(ch_f)./GAMA;
    %G0_f=(1/K_f).*fftshift(fft(gtSingle,freq_l));
    G0_f=(1/K_f).*gf;
  else
    disp('error in filter calculation, down or up?');
  end;
case 0  %phase conjugation
    x0=1;
    var_d=1;
    Swf=N0;
  if(char==0)  %uplink passive conjagation
    disp('passive conjagation uplink filter');
    K=sqrt((E/var_d)/x0);
    K_f=K;
    G0_f=K*gf;
    G0F=shape*G0_f;
    G_f=conj(G0F).*conj(ch_f);
    elseif(char==1)  %downlink
        disp('active conjagation downlink filter');
        df=2*pi/freq_l;
        v=abs(xf);
        Integral=(1/(2*pi))*sum(df.*v);
        beta=(E/var_d)/Integral;
        K_f=sqrt(beta);
        G_f=K_f*(GF).*conj(ch_f);
        G0_f=gf;

    else
        disp('error in filter calculation, down or up?');
    end;

otherwise
    disp('error in filter calculation function');
end;
```

```
function
[C,gama,Chat,gamahat]=DTchannel_response_alt_alt_alt100(n,CH_char,f_l,fs,fc,delay,res,N)
%delay is the estimate delay
%res is the delta T that used
%N is the number of the sample averages that need to be computed
%total number of the pages is ndiff+N


f_dop=1/200;
delT=res
ndiff=delay/res;
delTlength=ndiff+N;

df=fs/(f_l);
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);

% N=f_l/2;
% f=linspace(-N,N,f_l)/N*fs;
% f=linspace(0,fs,f_l);
path=3;
CP=CH_char(:,1);
P=CH_char(:,2);
TAU_rel=CH_char(:,3);
tmp=CH_char(:,4);

%normalize cp
s=sum((abs(CP)).^2);
factor=(1/n)/s;
CP=sqrt(factor).*CP;

power=mean(CP.^2);

%sigma_p=power-(mean(CP))^2;
sigma_p=power;
a0=exp(-2*pi*f_dop*delT);
sigma_zita=(1-a0^2)*sigma_p;
```

```
r=sqrt(sigma_zita)*randn(1,delTlength);

theta=rand(1,delTlength)*2*pi;
%zita=[1;1;1]*(r.*exp(j*theta));
zita=r.*exp(j*theta);

%kfac=a0^ndiff;
kfac=1;


for m=1:n
   for p=1:path
      Phi(m,p)=exp(-j*(m-1)*P(p));
      Coeff(m,p)=CP(p)*Phi(m,p)*exp(-j*2*pi*fc*TAU_rel(p));
   end;
end;
%CP3 is 3-d matrix. add delta t dimension
CP3(:,:,1)=Coeff;

for i=1:delTlength
   recursion=i

%    A0=a0.^[i-1+ndiff-1:-1:0];
%    X=zita(1:ndiff+i-1);
%    CP3(:,:,i+1)=a0^(ndiff+i-1)+A0.*X*(repmat(Phi,1,ndiff+i-1));
%    CP3ealier(:,:,i+1)=a0*C
%    CP3hat(:,:,i+1)
   CP3(:,:,i+1)=a0*CP3(:,:,i)+Phi*diag(zita(:,i));
   CP3hat(:,:,i)=kfac*CP3(:,:,i);
end;

for i=1:N
   count=i
   for m=1:n
      C(m,:,i)=CP3(m,1,i+ndiff)*ones(size(f));
      Chat(m,:,i)=CP3hat(m,1,i)*ones(size(f));
      for p=2:path
         C(m,:,i)=C(m,:,i)+CP3(m,p,i+ndiff)*exp(-j*2*pi*f*TAU_rel(p));
         Chat(m,:,i)=Chat(m,:,i)+CP3hat(m,p,i)*exp(-j*2*pi*f*TAU_rel(p));

      end;
   end;
   gama(i,:)=zeros(1,length(f));
   gamahat(i,:)=zeros(1,length(f));
   for m=1:n
      gama(i,:)=gama(i,:)+abs(C(m,:,i).^2);
      gamahat(i,:)=gamahat(i,:)+abs(Chat(m,:,i).^2);
   end;

end;


function out=detect(in)

%detect input sequence
phi=angle(in);
```

```
for n=1:length(in)
   if(phi(n)>0 & phi(n)< pi/2)
      out(1,n)=exp(j*pi/4);
   elseif(phi(n)>pi/2 & phi(n)<pi)
      out(1,n)=exp(j*3*pi/4);
   elseif(phi(n)>-pi & phi(n)<-pi/2)
      out(1,n)=exp(-j*3*pi/4);
   elseif(phi(n)>-pi/2& phi(n)<0)
      out(1,n)=exp(-j*pi/4);
   else
      out(1,n)=0;
   end;
end;
```

```
function [y_t]=processor(R,g,freq_length)


[r,c]=size(R);
[gr,gc]=size(g);
if(gr==1) %downlink case
   disp('downlink in processor');
   if(r~=1)
      disp('error in processor')
   end;
   y_t=zeros(1,c+freq_length/2-1);
   %g_t=ifft(fftshift(G_f(k,:)));
   temp=conv(g,R);
   temp=temp(freq_length/2+1:length(temp));
   y_t=temp;
end;
if(gr>1)  %uplink case
   disp('uplink in processor');
   y_t=zeros(1,c+freq_length/2-1);
   for k=1:r
      %g_t=ifft(fftshift(G_f(k,:)));
      temp=conv(g(k,:),R(k,:));
      temp=temp(freq_length/2+1:length(temp));
      y_t=y_t+temp;
   end;
end;

function y=sig_gen(dn,g,Ns);
%dn is the input stream
n=length(dn);
lg=length(g);
N=(n-1)*Ns+lg; %extra are the tails
trunc=((lg-1)/Ns)/2;
y=zeros(1,N);

for k=1:n
      temp=[zeros(1,(k-1)*Ns) dn(k).*g  zeros(1,N-(k-1)*Ns-lg)];
      y=y+temp;
```

```
end;

function [snr,diff]=SNROUT(x,y);

E=1;
% ref=detect(y);
% diff=ref-x;

diff=0;
z=y-x;
z_mag=abs(z);
d=mean(z_mag.^2);

snr=10*log10(E/d);

% sqrtrcos.m

% square root raised cosine;

function q=sqrtrcos(alpha,Ns,trunc);

% alpha = roll-off factor
% Ns = samples per symbol;
% trunc =  left/right truncation length in sb. int.

tn=(-trunc*Ns:trunc*Ns)/Ns;
%p=sin(pi*tn)./(pi*tn).*cos(alpha*pi*tn)./(1-4*alpha^2*tn.^2);
pitn=pi*tn;
q=cos((1+alpha)*pitn)+sin((1-alpha)*pitn)./(4*alpha*tn);
q=4*alpha/(pi*sqrt(Ns))*q./(1-(4*alpha*tn).^2);
%f0=find(p==Inf|p==-Inf);p(f0)=zeros(size(f0));
q(Ns*trunc+1)=1/sqrt(Ns)*(4*alpha/pi+1-alpha);
ind=find(tn==1/(4*alpha)|tn==-1/(4*alpha));
if ~isempty(ind);
  a=4*alpha/pi;
  a1=(1+alpha)/a;
  a2=(1-alpha)/a;
  q(ind)=a/sqrt(Ns)/2*(a1*sin(a1)-a2*cos(a2)+sin(a2));
end;

%figure(1),plot(tn,q)
```

```
function [xin,yout,GMA,CH_f,xf]=uplink_1(Nd,channel_number,noise_snr,freq_length,ddd,alpha)

range=3000;
depth=75;
snr_in_db=noise_snr;
E=1; %bit energy
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
M=4; %4-ary signal
n_mean=0;
```

```
trunc=16;
N0=0.5/(10^(snr_in_db/10));
fd=1/Tsym;

%random data stream
for i=1:Nd
   temp=rand;
   if(temp<0.25)
      m(i)=1/2;
   elseif(temp<0.5)
      m(i)=3/2;
   elseif(temp<0.75)
      m(i)=5/2;
   else
      m(i)=7/2;
   end
end;

dn=exp(j*2*pi.*m./M);
xin=dn;
array_d=(1500/fc)/2;

% u=rcosflt(dn,1/Tsym,1/T,'fir/sqrt',alpha,4,0.0001); %input wave form in sample rate
% u=u.';
% ddn=zeros(1,Ns*Nd-Ns+1);
% ddn(1:8:length(ddn))=dn(:);

[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
% maxdel=max(CC(3,2,:)-CC(1,2,:));

% freq_length=((Nd-1)*Ns+(2*trunc*Ns+1)+ceil(maxdel*fs))*2; %freq domain analysis vector
length
%filter vector to gurantee zero outside of W row vector
W=(Ns*(1/Tsym)/(s_rate))*freq_length;
F=lowpass(W,freq_length);
dummy=ones(1,channel_number);
FF=F.'*dummy;
FF=FF.';

[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,F,fc);


padn=(freq_length-2*trunc*Ns)/2;

gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

gtSingle=sqrtrcos(alpha,Ns,trunc);

% u=rcosflt(dn,1/Tsym,1/T,'fir/sqrt',alpha,4,0.0001); %input wave form in sample rate
% u=u.';
% length(u)
% u=sig_gen(dn,gtSingle,Ns);
```

```
u=waveGen(channel_number,dn,gtSingle,Ns);


[G_f]=up_receiver(E,N0,GMA,CH_f,xf,gf,freq_length,F);

% df=2*pi/freq_length;
% energy=sum(df*conj(G0_f).*G0_f)*(1/(2*pi))


%Rtotal_t=channel_t_alt(u,CH_f,E,snr_in_db,n_mean,1/Tsym,1/T,alpha,fc,CH_char,freq_length);
Rtotal_t=channel(u,CH_f,E,snr_in_db,n_mean);

% Rtotal_t=[Rtotal_t zeros(channel_number,length(Rtotal_t))];

[m,c]=size(Rtotal_t);
y_t=zeros(1,c+freq_length/2-1);
for k=1:channel_number
   %Rm_t=rcosflt(Rtotal_t(k,:),1/Tsym,1/T,'fir/sqrt/Fs',alpha,4,0.0001)
   Rm_t=Rtotal_t(k,:);
   Rm_t=Rm_t.';
   g_t=ifft(fftshift(G_f(k,:)));
   temp=conv(g_t,Rm_t.');
   temp=temp(freq_length/2+1:length(temp));
   y_t=y_t+temp;
end;

delay=trunc*Ns+1;
yout=sig_map(y_t,delay,Ns,Nd,0);

% test=[];
%
% for k=1:length(yout)
%     if(abs(yout(k))<0.5*(mean(abs(yout))))
%         test(k)=1;
%     else
%         test(k)=0;
%     end;
% end;
%
% figure
% stem(test);

function [xin,yout,GMA,CH_f,xf]=uplink_2(Nd,channel_number,noise_snr,freq_length,ddd,alpha)
%main engine running simulation
%input:
%Nd: number of random symbols
%channel_number
%noise_snr: one number
%freq_length
%ddd:detection delay offset. normally 0
%alpha: raised cosine

E=1;
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
```

```
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
snr_in_db=noise_snr;
n_mean=0;
N0=E*0.5/(10^(snr_in_db/10));
M=4; %4-ary signal
array_d=(1500/fc)/2;

%random data stream
for i=1:Nd
   temp=rand;
   if(temp<0.25)
      m(i)=1/2;
   elseif(temp<0.5)
      m(i)=3/2;
   elseif(temp<0.75)
      m(i)=5/2;
   else
      m(i)=7/2;
   end
end
dn=exp(j*2*pi.*m./M);
xin=dn;

%this block is for legacy only ignore
W=(Ns*(1/Tsym)/(s_rate))*freq_length;
F=lowpass(W,freq_length);
dummy=ones(1,channel_number);
FF=F.'*dummy;
FF=FF.';
%ignore above block. legacy

% generate channel gemetrically produce CH_char which is the model of the channel
[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
%calculate channel response based on CH_char
[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,F,fc);

trunc=16;
padn=(freq_length-2*trunc*Ns)/2;
gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

gtSingle=sqrtrcos(alpha,Ns,trunc);


% u=rcosflt(dn,1/Tsym,1/T,'fir/sqrt',alpha,4,0.0001); %input wave form in sample rate
% u=u.';
% length(u)
% ddn=zeros(1,Ns*Nd-Ns+1);
% ddn(1:8:length(ddn))=dn(:);

%[G0_f,G_f,K_f]=up_receiver2(E,N0,GMA,CH_f,xf,gf,freq_length,F);
```

```
[G0_f,G_f,K_f]=filterCALC2(E,N0,GMA,CH_f,xf,gf,freq_length,0);

%compute per bit energy for varification only ignore
df=2*pi/freq_length;
energy=sum(df*conj(G0_f).*G0_f)*(1/(2*pi))
%above block calcualte per bit energy

%this block generate signal to be input into channel
g0_t=ifft(fftshift(G0_f));
g0_t=g0_t;
%u=sig_gen(dn,g0_t,Ns);
u=waveGen(channel_number,dn,g0_t,Ns);
%signal through water
%Rtotal_t=channel_t_alt(u,CH_f,E,snr_in_db,n_mean,1/Tsym,1/T,alpha,fc,CH_char,freq_length);
Rtotal_t=channel(u,CH_f,E,snr_in_db,n_mean);

%this block process the received signal
[m,c]=size(Rtotal_t);
y_t=zeros(1,c+freq_length/2-1);
for k=1:channel_number
   %Rm_t=rcosflt(Rtotal_t(k,:),1/Tsym,1/T,'fir/sqrt/Fs',alpha,4,0.0001)
   Rm_t=Rtotal_t(k,:);
   Rm_t=Rm_t.';
   g_t=ifft(fftshift(G_f(k,:)));
   temp=conv(g_t,Rm_t.');
   temp=temp(freq_length/2+1:length(temp));
   y_t=y_t+temp;
end;

%this block sample the processed signal at the right interval and delay
delay=freq_length/2+1;
yout=sig_map(y_t,delay,Ns,Nd,0);

%ignore below block. for test only
% test=[];
% for k=1:length(yout)
%     if(abs(yout(k))<0.5*(mean(abs(yout))))
%        test(k)=1;
%     else
%        test(k)=0;
%     end;
% end;
%
% figure
% stem(test);

function G_f=up_receiver(E,N0,gama,ch_f,xf,gf,freq_l,F)
%implement the equation for filter on the receiver side
%filter is a vector filtering out freq outside W
[r c]=size(ch_f);
x0=1;
var_d=1;
K=sqrt((E/var_d)/x0);

shape=ones(r,1);
GAMA=shape*gama;
```

```
GF=shape*gf;
G_f=1/K*GF.*conj(ch_f)./GAMA;

%implement multipath
%three paths

clear all;
range=3000;
depth=75;
E=1; %bit energy
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
M=4; %4-ary signal
n_mean=0;
trunc=16;
fd=1/Tsym;
array_d=(1500/fc)/2;
max_delay=0.01; %longest path delay time
alpha=0.1;
snr_out=[];

gtSingle=sqrtrcos(alpha,Ns,trunc);

side=0;              %0:time-reversal 1:onesided focusing 2:twosided focusing
dir=1;        %direction of the transimission 0 is uplink 1 is downlink
channel_number=32;
nbit=1000;
Nd=nbit/2;
snr_v=[35];

est_delay=0.00;
ndiff=est_delay/(1/5000);
total=1;


time=max_delay+Nd*Ns/fs;
min_df=1/time;
minFlength=fs/min_df
freq_length=2^nextpow2(minFlength);

padn=(freq_length-2*trunc*Ns)/2;
gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);


%data generation
datastream=randint(2,Nd);
for m=1:nbit/2
   a=int2str(datastream(1,m));
   b=int2str(datastream(2,m));
   binstr=[a b];
   data(m)=bin2dec(binstr);
end;
dn=exp(j*2*pi.*((2*data+1)./2)./M);

dummy=0;
[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,dummy,fc);


%[Cmf3d,GMA2d,Cmf3dhat,GMA2dhat]=DTchannel_response_alt(channel_number,CH_char,fre
q_length,fs,F,fc,ndiff,total);

loop=0;
for n=1:length(snr_v)
   for k=1:total-ndiff
      loop=loop+1

%[x,y,xf]=sim_engine(channel_number,data,k,ndiff,Cmf3d,GMA2d,Cmf3dhat,GMA2dhat,CH_cha
r,snr_v(n),freq_length,alpha);
      N0=E*0.5/(10^(snr_v(n)/10));
      [G0_f,G_f,K_f]=filter_calculation(E,N0,GMA,CH_f,xf,gf,freq_length,dir,side);
      g0_t=ifft(fftshift(G0_f));
      for i=1:channel_number
         gf_t(i,:)=ifft(fftshift(G_f(i,:)));
      end;
      x=dn;
      if(dir==1)  %downlink
         U=waveGen(channel_number,dn,gf_t,Ns);
         R=channel(U,CH_f,E,snr_v(n),n_mean);
         Y=processor(R,g0_t,freq_length);
      elseif(dir==0) %uplink
         U=waveGen(channel_number,dn,g0_t,Ns);
         R=channel(U,CH_f,E,snr_v(n),n_mean);
         Y=processor(R,gf_t,freq_length);
         df=2*pi/freq_length;
         energy=sum(df*conj(G0_f).*G0_f)*(1/(2*pi))
      else
         disp('error in NewTest, up or down?');
      end;
      %sample
      delay=freq_length/2+1;
      y=sig_map(Y,delay,Ns,Nd,0);
      if(side==0)
         [snr_out(k,n),diff]=SNROUT_alt(x,y);
      else
         [snr_out(k,n),diff]=SNROUT(x,y);
      end;

   end;
end;
GMAini=GMA;

%plot(y,'*');
% phi=(angle(diff));
% mag=abs(diff);
% figure
% stem(phi);
% figure
% stem(mag);
%

figure
plot(y,'*');

if(total==1)
  temp=snr_out;
else
  temp=mean(snr_out);
end;

if(side==1)
  snrPredict=SNRcompare1_noPlot(freq_length,xf,GMAini,snr_v,temp);
else
  snrPredict=SNRcompare2_noPlot(freq_length,xf,GMAini,snr_v,temp);
end;

snr_out
%save perfect00.mat  snr_v snr_out -ascii -double -tabs
function simRun2000bitsFull(channel_number,side,dir,est_delay)

range=3000;
depth=75;
E=1; %bit energy
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
M=4; %4-ary signal
n_mean=0;
trunc=16;
fd=1/Tsym;
array_d=(1500/fc)/2;
max_delay=0.01; %longest path delay time
alpha=0.1;
snr_out=[];

gtSingle=sqrtrcos(alpha,Ns,trunc);


%*********************************************************************************
%input block
% side=1;             %0:time-reversal 1:onesided focusing 2:twosided focusing
% dir=0;       %direction of the transimission 0 is uplink 1 is downlink
% channel_number=4;
nbit=2000; %1000 symbols
Nd=nbit/2;
snr_v=[30 35];
```

```
%estimate machine
%est_delay=0;
ndiff=est_delay/(1/5000);
total=Nd+1;

%*******************************************************************************

time=max_delay+Nd*Ns/fs;
min_df=1/time;
minFlength=fs/min_df
freq_length=2^nextpow2(minFlength);

padn=(freq_length-2*trunc*Ns)/2;
gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

%data generation
datastream=randint(2,Nd);
for m=1:nbit/2
   a=int2str(datastream(1,m));
   b=int2str(datastream(2,m));
   binstr=[a b];
   data(m)=bin2dec(binstr);
end;
dn=exp(j*2*pi.*((2*data+1)./2)./M);

dummy=0;
[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
%[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,dummy,fc);

[Cmf3d,GMA2d,Cmf3dhat,GMA2dhat]=DTchannel_response_alt(channel_number,CH_char,freq_
length,fs,dummy,fc,ndiff,total);


for n=1:length(snr_v)
   datapoint=snr_v(n)
   loop=0;
   for k=1:total-ndiff
      loop=loop+1
      CH_f=Cmf3d(:,:,k+ndiff);
      GMA=GMA2d(k+ndiff,:);
      CH_fhat=Cmf3dhat(:,:,k);
      GMAhat=GMA2dhat(k,:);
      N0=E*0.5/(10^(snr_v(n)/10));
      %using estimated value for filter generation also input Swf
      [G0_f,G_f,K_f]=filter_calculation(E,N0,GMAhat,CH_fhat,xf,gf,freq_length,dir,side);
      %time domain filter
      g0_t=ifft(fftshift(G0_f));
      for i=1:channel_number
         gf_t(i,:)=ifft(fftshift(G_f(i,:)));
      end;
      x=dn;
      if(dir==1)  %downlink
```

```
          U=waveGen(channel_number,dn,gf_t,Ns);
          R=channel(U,CH_f,E,snr_v(n),n_mean);
          Y=processor(R,g0_t,freq_length);
       elseif(dir==0) %uplink
          U=waveGen(channel_number,dn,g0_t,Ns);
          R=channel(U,CH_f,E,snr_v(n),n_mean);
          Y=processor(R,gf_t,freq_length);
       else
          disp('error in NewTest, up or down?');
       end;
       %sample
       delay=freq_length/2+1;
       y=sig_map(Y,delay,Ns,Nd,0);
       if(side==0)
          y=Ns*y;
          [snr_out(k,n),diff]=SNROUT_alt(x,y);
       else
          [snr_out(k,n),diff]=SNROUT(x,y);
       end;

    end;
 end;
 GMAini=GMA;

 if(total==1)
    temp=snr_out;
 else
    temp=mean(snr_out);
 end;

 if(side==1)
    [SNRpredict]=SNRcompare1_noPlot(freq_length,xf,GMAini,snr_v,temp);
 else
    [SNRpredict]=SNRcompare2_noPlot(freq_length,xf,GMAini,snr_v,temp);
 end;

 switch side
 case 0
    if(dir==0)
       save output00.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    elseif(dir==1)
       save output01.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    else
       disp('error in simRun2000bits');
    end;
 case 1
       if(dir==0)
       save output10.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    elseif(dir==1)
       save output11.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    else
       disp('error in simRun2000bits');
    end;
 case 2
       if(dir==0)
       save output20.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    elseif(dir==1)
        save output21.mat  snr_v snr_out SNRpredict -ascii -double -tabs
    else
        disp('error in simRun2000bits');
    end;
end;

function Y_t=channel_t_alt_deltaT(uu,C,E,snr_in_db,noise_mean,fd,fs,alpha,fc,ch_char,f_l)

[chn,c]=size(C);
% W=(fd/fs)*f_l;
% F=lowpass(W,f_l);
% dummy=ones(1,chn);
% FF=F.'*dummy;
% FF=FF.';

% maxdel=max(CC(3,2,:)-CC(1,2,:));
% maxn=(length(u)+ceil(maxdel*fs));
% y=zeros(1,maxn);
% tmp=ch_char(:,4)
% for n=1:chn
%     for k=1:3
%         %y=fftfilt(ch_t(chn,:),u);
%         tau=CC(k,2,n)-(n-1)*tmp(k);
%         a=CC(k,1,n)*exp(-j*2*pi*fc*(n-1)*tmp(k));
%         dummy=nth_path_t_alt(tau,a,u,fd,fs,alpha,fc);
%         y=y+[dummy zeros(1,maxn-length(dummy))];
%     end;
df=fs/f_l;
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);
path=3;
CP=ch_char(:,1);
P=ch_char(:,2);
TAU_rel=ch_char(:,3)
tmp=ch_char(:,4);

% maxdel=max(CC(3,2,:)-CC(1,2,:));
% maxn=(length(u)+ceil(maxdel*fs));
% del_n=ceil(TAU_rel*fs);

shape=[0:1:chn-1]';
Phi=exp(-j*shape*(P.'));
Coeff=Phi*diag(CP);
for m=1:chn
    for p=1:path
        Phi(m,p)=exp(-j*(m-1)*P(p));
        Coeff(m,p)=CP(p)*Phi(m,p);
    end;
end;
for m=1:chn
    C(m,:)=Coeff(m,1)*ones(size(f));
    for p=2:path
        C(m,:)=C(m,:)+Coeff(m,p)*exp(-j*2*pi*f*TAU_rel(p));
    end;
    Cm_f=C(m,:);
    ct=ifftshift(ifft(fftshift(Cm_f)));
    y=conv(ct,uu);
    y=y(f_l/2+1:length(y));

%    figure
%    plot(abs(Gm_f));
%    figure
%    plot(unwrap(angle(Gm_f)/pi));
%generate and add noise
    SNR=10^(snr_in_db/10);
    var=1/2/SNR;
    r=randn(1,length(y)).*sqrt(var)+noise_mean;
    theta=rand(1,length(y))*2*pi;
    noise=r.*exp(j*theta);
    y=y+noise;
    Y_t(m,:)=y;
end;
```

```
function SNRcompare1(freq_length,xf,GMA,snr_v,snr_out)

E=1;
N0=0.5*E./10.^(snr_v/10);
x0=1;

df=2*pi/freq_length;

integral=(1/2/pi)*sum(df.*abs(xf)./GMA);
SNRpredict=10*log10(((2*E./N0)*x0)/integral);


figure
title('one sided');
hold on
plot(snr_v,snr_out,'*');
xlabel('E/N0 (dB)');
ylabel('SNRout')
plot(snr_v,SNRpredict);
grid on
hold off
%implement multipath
%three paths

clear all
Nd=500;
channel_number=4; %ch number
fs=40000;
snr_in_db=35;
snr_out=[];
alpha=0.5;

E=1;
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
```

```
fc= 15000;% carrier frequency
fs=s_rate;

n_mean=0;
N0=E/(10^(snr_in_db/10));
M=4; %4-ary signal
array_d=(1500/fc)/2;

%random data stream
for i=1:Nd
   temp=rand;
   if(temp<0.25)
      m(i)=1/2;
   elseif(temp<0.5)
      m(i)=3/2;
   elseif(temp<0.75)
      m(i)=5/2;
   else
      m(i)=7/2;
   end
end
dn=exp(j*2*pi.*m./M);
xin=dn;
trunc=16;

max_delay=0.01;
time=max_delay+Nd*Ns/fs;
min_df=1/time;
minFlength=fs/min_df

freq_length=2^nextpow2(minFlength)

%
[CH_char,CC]=channel_construc(channel_number,3000,75,array_d,fc,fs);
% maxdel=max(CC(3,2,:)-CC(1,2,:));
%
% df=1/maxdel;
% minf=fs/df;

W=(Ns*(1/Tsym)/(s_rate))*freq_length;
F=lowpass(W,freq_length);
dummy=ones(1,channel_number);
FF=F.'*dummy;
FF=FF.';

[CH_f,GMA]=channel_response_alt(channel_number,CH_char,freq_length,fs,F,fc);

%[CH_f,GMA]=multipath(channel_number,3,3000,75,75,freq_length,fc,fs);

%filter vector to gurantee zero outside of W row vector

padn=(freq_length-2*trunc*Ns)/2;

gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
%gt=sqrtrcos(alpha,Ns,trunc);
gf=fftshift(fft(gt,freq_length));
```

```
%gf=abs(gf);
%xt=rcos(alpha,Ns,trunc);
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

gtSingle=sqrtrcos(alpha,Ns,trunc);


% u=rcosflt(dn,1/Tsym,1/T,'fir/sqrt',alpha,4,0.0001); %input wave form in sample rate
% u=u.';
% length(u)


% ddn=zeros(1,Ns*Nd-Ns+1);
% ddn(1:8:length(ddn))=dn(:);

[G0_f,G_f,K_f]=up_receiver2(E,N0,GMA,CH_f,xf,gf,freq_length,F);
df=2*pi/freq_length;
energy=sum(df*conj(G0_f).*G0_f)*(1/(2*pi))

g0_t=ifft(fftshift(G0_f));
g0_t=g0_t;

u=sig_gen(dn,g0_t,Ns);


Rtotal_t=channel_t_alt(u,CH_f,E,snr_in_db,n_mean,1/Tsym,1/T,alpha,fc,CH_char,freq_length);


[m,c]=size(Rtotal_t);
y_t=zeros(1,c+freq_length/2-1);
for k=1:channel_number
   %Rm_t=rcosflt(Rtotal_t(k,:),1/Tsym,1/T,'fir/sqrt/Fs',alpha,4,0.0001)
   Rm_t=Rtotal_t(k,:);
   Rm_t=Rm_t.';
   g_t=ifft(fftshift(G_f(k,:)));
   temp=conv(g_t,Rm_t.');
   temp=temp(freq_length/2+1:length(temp));
   y_t=y_t+temp;
end;

delay=freq_length/2+1;

yout=sig_map(y_t,delay,Ns,Nd,0);

test=[];

for k=1:length(yout)
   if(abs(yout(k))<0.80*(mean(abs(yout))))
      test(k)=1;
   else
      test(k)=0;
   end;
end;
```

```
figure
stem(test);

TEST=G0_f.*sum(G_f.*CH_f);
ff=[0:freq_length-1]/freq_length*fs/1000;
figure
plot(ff,abs(TEST));
grid on
title('TEST');

TEST_t=(ifft(fftshift(TEST)));

figure
subplot(2,1,1);
plot(real(TEST_t));
grid on
ylabel('real part of TEST t');
grid on
subplot(2,1,2);
plot(imag(TEST_t));
ylabel('imag part of TEST_t');

figure
stem(real(TEST_t(401:8:600)));


figure
plot(real(u));
grid on
title('u');
figure
plot(real(Rtotal_t(1,:)));
grid on
title('Rt1')
figure
plot(real(y_t));
grid on
title('yt');

ch_t=ifft(fftshift(GMA));
tt=[0:1/fs:(length(ch_t)-1)*1/fs]*1000;
figure
plot(abs(ifftshift(ch_t)));
grid on
title('gama');
[snrOut,diff]=SNROUT(dn,yout);
snrOut
```

```
function
[xin,yout,xf]=sim_engine(channel_number,packet,count,ndiff,Cmf3d,GMA2d,Cmf3dhat,GMA2dha
t,CH_char,noise_snr,freq_length,alpha)
M=4;
range=3000;
depth=75;
```

```
snr_in_db=noise_snr;
E=1; %bit energy
Ns=8; %sample per symbol
Tsym=1/5000;  %symbol interval
s_rate=Ns*(1/Tsym); %sampling rate
T=1/s_rate; %smapling interval
fc= 15000;% carrier frequency
fs=s_rate;
n_mean=0;
trunc=16;
N0=0.5*1/(10^(snr_in_db/10));
fd=1/Tsym;
Nd=length(packet);
%delTlength=ndiff+3;
%random data stream
% for i=1:Nd
%     temp=rand;
%     if(temp<0.25)
%         m(i)=1/2;
%     elseif(temp<0.5)
%         m(i)=3/2;
%     elseif(temp<0.75)
%         m(i)=5/2;
%     else
%         m(i)=7/2;
%     end
% end;
%

dn=exp(j*2*pi.*((2*packet+1)./2)./M);
xin=dn;
array_d=(1500/fc)/2;

W=(Ns*(1/Tsym)/(s_rate))*freq_length;
F=lowpass(W,freq_length);
dummy=ones(1,channel_number);
FF=F.'*dummy;
FF=FF.';

CH_f=Cmf3d(:,:,count+ndiff);
GMA=GMA2d(count+ndiff,:);
CH_fhat=Cmf3dhat(:,:,count);
GMAhat=GMA2dhat(count,:);

padn=(freq_length-2*trunc*Ns)/2;

gt=[zeros(1,padn) sqrtrcos(alpha,Ns,trunc)  zeros(1,padn)];
gf=fftshift(fft(gt,freq_length));
xt=[zeros(1,padn) rcos(alpha,Ns,trunc) zeros(1,padn)];
xf=fftshift(fft(xt,freq_length));
xf=abs(xf);

gtSingle=sqrtrcos(alpha,Ns,trunc);

% u=rcosflt(dn,1/Tsym,1/T,'fir/sqrt',alpha,4,0.0001); %input wave form in sample rate
% u=u.';
% length(u)
u=sig_gen(dn,gtSingle,Ns);

% ddn=zeros(1,Ns*Nd-Ns+1);
% ddn(1:8:length(ddn))=dn(:);

[G_f]=up_receiver(E,N0,GMAhat,CH_fhat,xf,gf,freq_length,F);

% df=2*pi/freq_length;
% energy=sum(df*conj(G0_f).*G0_f)*(1/(2*pi))
%
%Rtotal_t=channel_t_alt(u,CH_f,E,snr_in_db,n_mean,1/Tsym,1/T,alpha,fc,CH_char,freq_length);

Rtotal_t=channel(u,CH_f,E,snr_in_db,n_mean);


[m,c]=size(Rtotal_t);
y_t=zeros(1,c+freq_length/2-1);
for k=1:channel_number
   %Rm_t=rcosflt(Rtotal_t(k,:),1/Tsym,1/T,'fir/sqrt/Fs',alpha,4,0.0001)
   Rm_t=Rtotal_t(k,:);
   Rm_t=Rm_t.';
   g_t=ifft(fftshift(G_f(k,:)));
   temp=conv(g_t,Rm_t.');
   temp=temp(freq_length/2+1:length(temp));
   y_t=y_t+temp;
end;

delay=trunc*Ns+1;

% yout=y_t(delay:Ns:Nd*Ns);

yout=sig_map(y_t,delay,Ns,Nd,0);
% test=[];
%
% for k=1:length(yout)
%    if(abs(yout(k))<0.5*(mean(abs(yout))))
%       test(k)=1;
%    else
%       test(k)=0;
%    end;
% end;
%
% figure
% stem(test);

Swf=N0/2;
beta=sqrt(Swf);
df=2*pi/freq_length;
energy=sum(df*conj(gf).*gf)*(1/(2*pi))

% raised cosine;

function p=rcos(alpha,Ns,trunc);

% alpha = roll-off factor
% Ns = samples per symbol;
% trunc =  left/right truncation length in sb. int.

tn=(-trunc*Ns:trunc*Ns)/Ns;
p=sin(pi*tn)./(pi*tn).*cos(alpha*pi*tn)./(1-4*alpha^2*tn.^2);
f0=find(p==Inf|p==-Inf);p(f0)=zeros(size(f0));
p(Ns*trunc+1)=1;


function Y_t=channel_t_alt(uu,C,E,snr_in_db,noise_mean,fd,fs,alpha,fc,ch_char,f_l)

[chn,c]=size(C);
% W=(fd/fs)*f_l;
% F=lowpass(W,f_l);
% dummy=ones(1,chn);
% FF=F.'*dummy;
% FF=FF.';

% maxdel=max(CC(3,2,:)-CC(1,2,:));
% maxn=(length(u)+ceil(maxdel*fs));
% y=zeros(1,maxn);
% tmp=ch_char(:,4)
% for n=1:chn
%     for k=1:3
%        %y=fftfilt(ch_t(chn,:),u);
%        tau=CC(k,2,n)-(n-1)*tmp(k);
%        a=CC(k,1,n)*exp(-j*2*pi*fc*(n-1)*tmp(k));
%        dummy=nth_path_t_alt(tau,a,u,fd,fs,alpha,fc);
%        y=y+[dummy zeros(1,maxn-length(dummy))];
%     end;
df=fs/f_l;
f=-fs/2:df:fs/2;
f=f(1:length(f)-1);
path=3;
CP=ch_char(:,1);
P=ch_char(:,2);
TAU_rel=ch_char(:,3)
tmp=ch_char(:,4);

% maxdel=max(CC(3,2,:)-CC(1,2,:));
% maxn=(length(u)+ceil(maxdel*fs));
% del_n=ceil(TAU_rel*fs);

for m=1:chn
%     for p=1:path
%        Phi(m,p)=exp(-j*(m-1)*P(p));
%        Coeff(m,p)=CP(p)*Phi(m,p).*exp(-j*2*pi*fc*TAU_rel(p));
%     end;
% end;
% for m=1:chn
%     C(m,:)=Coeff(m,1)*ones(size(f));
%     for p=2:path
%        C(m,:)=C(m,:)+Coeff(m,p)*exp(-j*2*pi*f*TAU_rel(p));
%     end;

   Cm_f=C(m,:);
   ct=ifftshift(ifft(fftshift(Cm_f)));
   y=conv(ct,uu);
   y=y(f_l/2+1:length(y));

%    figure
%    plot(abs(Gm_f));
%    figure
%    plot(unwrap(angle(Gm_f)/pi));
%generate and add noise
   SNR=10^(snr_in_db/10);
   var=1/2/SNR;
   r=randn(1,length(y)).*sqrt(var)+noise_mean;
   theta=rand(1,length(y))*2*pi;
   noise=r.*exp(j*theta);
   y=y+noise;
   Y_t(m,:)=y;
end;
```