1998

Digital Receivers

George Rofa
Edith Cowan University

Edith Cowan University

Copyright Warning

DIGITAL RECEIVERS

By

George Rofa

A Thesis Submitted in Fulfillment of the Requirements for the Award of

Bachelor of Engineering (Computer Systems)

At

Edith Cowan University
Faculty of Science, Technology and Engineering
School of Engineering

Supervisor

Dr. Daryoush Habibi

Submission Date

6 November 1998

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

DECLARATION

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously written by another person except where due reference is made in the text.

Signature 

Date 27/3/2000

Acknowledgements

First of all, I would like to thank my supervisor Dr. Daryoush Habibi for his guidance, help, support and encouragement that he has given to me throughout the last two years of this course.

My grateful thanks also go to all the lecturers and tutors that I had during my course, to Prof. Abdesselam Bouzerdoum for his support throughout this project, and finally special thanks to Miss Vis Ramakonar for her support and help in this project.

I should also mention my family members who have been very patient and understanding all throughout my course. Thank you very much.

Table of Contents

Chapter 1

PART 'A'

DIGITAL RECEIVERS

Introduction

In a data receiver, timing must be synchronized to the symbols of the incoming data signal. In analog modems, synchronization can be performed by two ways. It can be performed by a feedback loop that adjusts the phase of a local clock, or by a feedforward arrangement that regenerates a timing wave from the incoming signal. The local clock or the timing wave is used to sample or strobe the filtered output, of the modem, once per symbol interval, message data are then recovered from the strobes.

Using digital techniques to implement modems introduces sampling of the signal. In some occasions, the sampling can be synchronized to the symbol rate of the incoming signal (synchronously sampled modems). In this case, timing can be recovered in the same ways that are familiar from analog practice.

In other occasions, the sampling cannot be synchronised to the incoming signal. The sampling of the received signal is performed by a fixed sampling clock and thus sampling is not synchronized to the incoming symbols. The sampling clock must remain independent of the symbol timing. In that case, timing adjustment must be done by digital methods after sampling. This can be done by calculating the value of the signal at the desired time instants by interpolation. *Interpolation* is used to interpolate among the nonsynchronized samples so as to produce the correct and the same strobe values at the modem output as if the original sampling had been synchronized to the symbols. In other words the value of the received continuous-time signal is approximated at the desired time instant by interpolation. The error resulting from this approximation is considered as

additive noise which degrades the quality of the overall communication system and should be limited to sufficiently low level.

Interpolation is a timing adjustment operation on the signal itself and not on the local clock or the timing wave. Thus interpolation is different from timing adjustment in analog modems. Although the process of timing adjustment includes much more than interpolation alone and *rate conversion* is a more accurate label, interpolation is used to denote all of the processes that are involved in timing adjustment.

Chapter 2

The Model

Figure 2-1 shows a block diagram of the feedback timing recovery as in [1] Floyd M. Gardner (Mar. 1993 p. 502). The received signal $x(t)$ is a time-continuous PAM signal in which the symbol pulses are uniformly spaced at intervals T. The received signal $x(t)$ is assumed to be real for simplicity, but this assumption can be easily removed without any difficulty. It is also assumed to be bandlimited and thus can be sampled at a rate of $1/T_s$ without aliasing. If $x(t)$ is not adequately bandlimited then distortion is introduced by aliasing causing a performance penalty. Samples $x(mT_s) = x(m)$ are taken at uniform intervals T_s. Since the symbol timing is derived from source, which is independent of the sampling clock, then the ratio T/T_s is typically irrational. This is true in most practical situations. The signal samples are applied to the interpolator and interpolants, $y(kT_i) = y(k)$ are computed at intervals T_i.

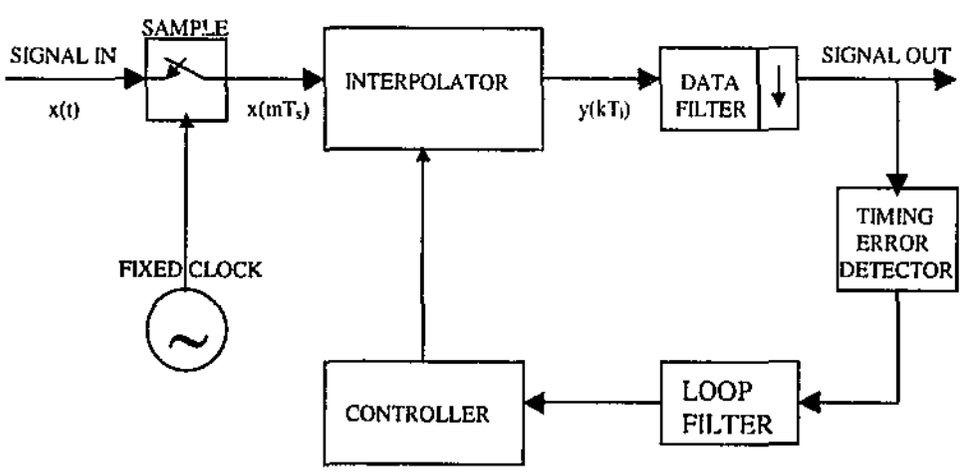

Figure 2-1 Elements of digital timing recovery.

The data filter employs the interpolants to compute the strobes that are used for data and timing recovery. Its placement is not essential. It could be after the interpolator or it could be outside the feedback loop prior to the interpolator. Placing it inside the loop, introduces delay, while with post placement, the data filter can decimate its output to the

required strobe rate saving on computing burden. Post placement is not really necessary since quite modest sampling rates provide excellent results even with very simple interpolators.

The feedback loop elements contribute to the synchronization process. The timing error detector measures the timing error, which is then filtered in the loop filter. The output of the loop filter drives the controller from which the interpolator obtains instructions for its computations.

Chapter 3

Interpolation

The task of the interpolator is to compute intermediate values between signal samples. Figure 3-1 is a fictitious hybrid analog/digital method of rate conversion as shown in [1] Gardner (1993 p. 503).

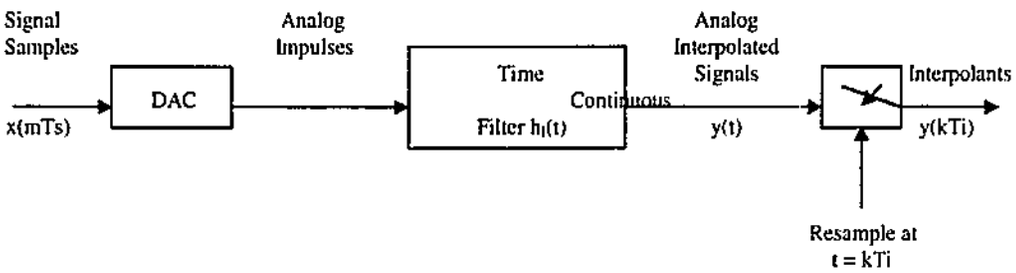

Figure 3-1 Rate conversion with time-continuous filter.

The samples are converted to a sequence of weighted analog impulses. These impulses are then applied to a time-continuous analog filter with impulse response $h_I(t)$. An ideal linear interpolator filter, according to [4] Meyr el at, (1998 p. 238) has an impulse response:

$$h_I(kT_s,\tau) = si\left[\frac{\pi}{T_s}(kT_s+\tau)\right] \qquad (3\text{-}1)$$

and frequency response

$$H_I(e^{j\omega T_s},\tau) = \frac{1}{T_s}\sum_{n=-\infty}^{\infty} H_I(\omega - \frac{2\pi}{T_s}n,\tau) \qquad (3\text{-}2)$$

where $H_I(\omega, \tau)$ is the Fourier transform of $si[\pi/T_s\ (t+\tau)]$:

$$H_I(\omega,\tau) = \begin{cases} T_s \exp(j\omega\tau) & \left|\frac{\omega}{2\pi}\right| < \frac{1}{2Ts} \\ 0 & elsewhere \end{cases} \qquad (3\text{-}3)$$

The time continuous output of the filter as in [1] is:

$$y(t) = \sum_m x(m) h_I(t - mT_s). \qquad (3\text{-}4)$$

In that case y(t) ≠ x(t) and there is no need to recover the original signal by this type of interpolation. Then y(t) is resampled at time instants $t = kT_i$. T_i is synchronised with the signal symbols and in general T_i/T_s is irrational since the sampling and symbol rates are incommensurate. The interpolants (new samples) are represented according to [1] Gardner (1993 p. 503) by:

$$y(kTi) = \sum_m x(mT_s) h_I(kT_i - mT_s). \qquad (3\text{-}5)$$

From equation (3-5), the interpolants can be computed by the knowledge of:

- the input sequence {x(m)}
- the impulse response $h_I(t)$ of the interpolating filter
- and the time instants mT_s and kT_i of the input and output samples.

The digitally computed interpolants have identically the same values as if the analog operations had been performed.

The indexing in equation (3-5) above can be rearranged to obtain a more useful format by defining new parameters as follows:

Filter index (i):

$$i = int[kTi/Ts] - m \qquad (3\text{-}6)$$

Basepoint index (m_k):

$$m_k = int[kT_i/T_s] \tag{3-7}$$

Fractional interval (μ_k):

$$\mu_k = kT_i/T_s - m_k \qquad (0 \le \mu_k < 1) \tag{3-8}$$

The relation between these parameters is shown in Figure 3-2.

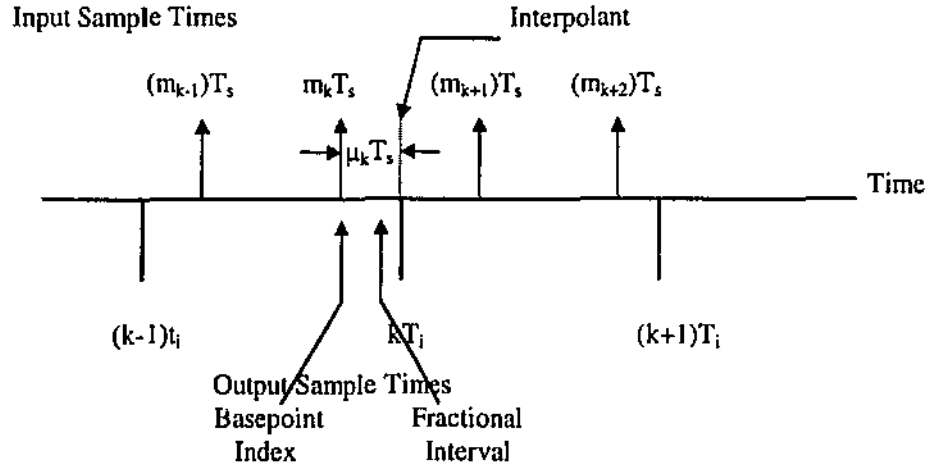

Figure 3-2 Sample time relations.

Equation (3-5) above can be rewritten as:

$$y(kT_i) = y[(m_k + \mu_k)\, T_s]$$

$$y(kT_i) = \sum_{i=11}^{12} x[(m_k - i)T_s]h_I[(i + \mu_k)T_s]. \tag{3-9}$$

Equation (3-9) is the foundation of digital interpolation in modems. The sequence of signal samples {x(m)}are taken at intervals T_s and $h_I(t)$ is the finite-duration impulse response of a fictitious, time-continuous, analog interpolating filter. Interpolants y(k) are

delivered at adjustable intervals T_i. The filter index $I = I_1$ to I_2, the basepoint index m_k identifies the $I = I_2 - I_1 + 1$ signal samples to be used for the kth interpolant and the fractional interval μ_k identifies the I filter coefficients to be employed for the *k*th interpolant.

When T_i is incommensurate with T_s, the fractional interval μ_k is irrational and thus will change for each interpolant, taking on an infinite number of values which never repeat exactly if μ_k is determined to infinite precision. On the other hand, if T_i is assumed to be very close to T_s (sampling in almost synchronized), then μ_k changes very slowly and if quantized it might remain constant over many interpolants. And when T_s were commensurate with T_i but not equal, then μ_k would cyclicly repeat a finite set of values.

3.1.3 Methods of Interpolation

An important task is to approximate a complicated function *f(x)* by another function *a(x)* which is simpler. One way of solving this problem is by interpolating *f* at discrete values. A function *f* in one or more variables can be given is by table as follows:

x_0	x_1	...	x_n
f_0	f_1	...	f_n

	y_0	—	y_m
x_0	$f_{0,0}$	—	$f_{0,m}$
|	|		|
x_n	$f_{n,0}$	—	$f_{n,m}$

In order to obtain the intermediate values of *f*, which are not tabulated, interpolation between neighbouring values of the table is required.

3.1.1 Interpolating Polynomials

Figure 3-3 illustrates the interpolation problem geometrically in one variable. If $N + 1$ distinct points are given with the coordinates (x_i, f_i), a curve $p(x)$ that passes through these points and can evaluate the curve for any argument x is to be found. The x_i are called abscissae, the f_i are called the interpolation ordinates and the points (x_i, f_i) are the interpolation points.

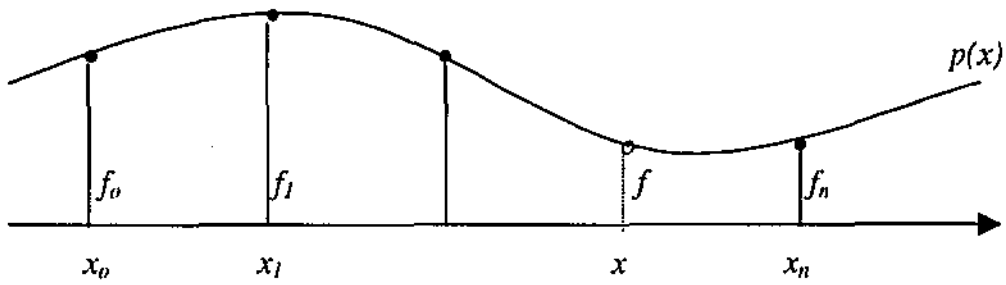

Figure 3-3 Interpolating polynomials.

This concept can be summarised in the following theorem as in [6] (Boehm & Prautzsch p. 89):

Theorem 1: *If $x_0,\ldots,x_n$ are $n + 1$ distinct arguments with corresponding ordinates $f_0,\ldots,f_n$, then there exists a unique polynomial* pol(x) *of degree at most n with the property that:*

$pol(x_i) = f_i, \qquad i = 0,\ldots,n.$

3.1.2 Lagrange Polynomials

Lagrange polynomial is the unique polynomial of degree n passing through $N + 1$ points. This polynomial interpolant can be thought of as an approximation of some other function passing through these $N + 1$ points. Therefore, the more data points are used, the better the approximation to the original function should be. Unfortunately, this is not the

case. Increasing the number of points simply increases the degree of polynomial since the degree of the Lagrange polynomial is one less than the number of data points. This results in increasing the amount of computations required. Also with higher degree polynomials, it can oscillate between the points. Therefore, polynomials of lesser degree passing through several consecutive points can be combined. The only problem with this is that the overall curve is not smooth at the joints. But smoothness conditions at the data points can be applied and thus obtaining a polynomial having these properties. This process of obtaining a smooth curve especially at the joints is called "Blending".

Lagrange classical formula is:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_n)}{(x_1-x_2)(x_1-x_3)...(x_1-x_n)} y_1 + \frac{(x-x_1)(x-x_3)...(x-x_n)}{(x_2-x_1)(x_2-x_3)...(x_2-x_n)} y_2 + \\ ... + \frac{(x-x_1)(x-x_2)...(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)...(x_n-x_{n-1})} y_n \qquad (3-10)$$

There are N terms, a polynomial of degree $N-1$ and constructed to be zero for all of the x_i, except one which is constructed to be y_i. The resulting algorithm gives no error estimate and is also awkward to program. A much better algorithm is derived using a blending function and the above equation can be rewritten as:

$$y(t) = \sum_{i=I_1}^{I_2} C_i x(I_1 + I_2 - i) \qquad (3-11)$$

where the blending function C_i is:

$$C_i = \prod_{j=I_1, j\neq i}^{I_2} \frac{t-t_j}{t_i-t_j} \qquad (3-12)$$

There must be an even number N of samples in the basepoint set. Interpolation is performed in the central interval of the basepoint set. Thus the interpolating polynomial must be of odd degree $N-1$. So for cubic interpolation, the number of points required is four.

3.1.3 Cubic Splines

A cubic spline has the property that the three coordinate functions, $x(u)$, $y(u)$ and $z(u)$, are each cubic polynomials in the variable u:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$
$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z. \qquad 0 \le u \le 1 \qquad (3\text{-}13)$$

The domain of u is finite and is assumed that $0 \le u \le 1$. If we have $N + 1$ control points, then we have N curve sections between those control points, defined by the above equations. For each of these three equations, the values of the four coefficients a, b, c and d should be determined.

A spline passes through two points and satisfies a differentiability condition at each of these endpoints. So there are four conditions that require a polynomial degree of at least 3. That is why cubic splines are so popular. N cubic splines are blended together if $N+1$ data points are given. Equal derivatives at the end points of each spline ensure smoothness at the joints.

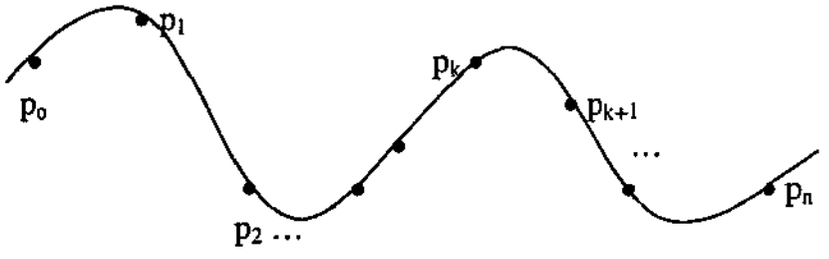

Figure 3-4 A piecewise continuous cubic-spline interpolation of n + 1 control points.

Compared to higher-order polynomials, cubic splines require less calculations and memory and are more stable. Compared to lower-order polynomials, they are more flexible for modeling arbitrary curve shapes.

There are different types of cubic splines, some of them are summarised in the following sections.

3.1.3.1 Natural Cubic Splines

A natural cubic spline is formulated by requiring that two adjacent curve sections have the same first and second parametric derivatives at their common boundary.
For $n + 1$ control points to fit as in Figure 3-4, there are n curve sections with a total of $4n$ polynomial coefficients to be calculated.

The problem with the natural cubic spline is that if the position of any control point is altered, the entire curve is affected. So the major disadvantage is that it allows for no "local control", so that we cannot restructure part of the curve without specifying an entirely new set of control points.

3.1.3.2 Hermite Interpolation

A Hermite spline as an interpolating piecewise cubic polynomial with a specified tangent at each control point. Unlike the natural cubic splines, Hermite splines can be adjusted locally because each curve section is only dependent on its endpoint constraints. The boundary conditions that define the Hermite curve section, between control points p_k and p_{k+1} represented by a parametric cubic point function $P(u)$ are:

$$\begin{aligned} P(0) &= p_k \\ P(1) &= p_{k+1} \\ P'(0) &= Dp_k \\ P'(1) &= Dp_{k+1} \end{aligned} \qquad (3\text{-}14)$$

where Dp_k and Dp_{k+1} specify the values for the parametric derivatives at control points p_k and p_{k+1}, respectively.

The Hermite curve section can be expressed according to equation 3-13, in matrix form as:

$$\mathbf{P}(u) = [u^3\ u^2\ u\ 1].\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{3-15}$$

and the derivative of the point function can be expressed as:

$$\mathbf{P'}(u) = [3u^2\ 2u\ 1\ 0] .\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{3-16}$$

Substituting endpoint values 0 and 1 for parameter u into equations 3-15 and 3-16, the Hermit boundary conditions 3-14 can be expressed in matrix form as:

$$\begin{bmatrix} \mathbf{P}_k \\ \mathbf{P}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} . \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{3-17}$$

Equation 3-17 can be solved for the polynomial coefficients as:

$$\begin{aligned} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} . \begin{bmatrix} \mathbf{P}_k \\ \mathbf{P}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} \mathbf{P}_k \\ \mathbf{P}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \\ &= \mathbf{M}_H . \begin{bmatrix} \mathbf{P}_k \\ \mathbf{P}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \end{aligned} \tag{3-18}$$

where $\mathbf{M}_H$, the Hermite matrix, is the inverse of the boundary matrix. So equation 3-15 can be rewritten in terms of the boundary conditions as:

$$\mathbf{P}(u) = [u3\ u2\ u\ 1]\ .\ \mathbf{M}_H\ .\ \begin{bmatrix} \mathbf{P}_k \\ \mathbf{P}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \qquad (3\text{-}19)$$

By expanding equation 3-19 amd collecting coefficients for the boundary constraints, expressions for the Hermite blending functions can be determined. The following polynomial is then obtained as in [5] Hearne & Baker (1997):

$$\mathbf{P}(u) = \mathbf{p}_k(2u^3 - 3u^2 + 1) + \mathbf{p}_{k+1}(-2u^3 + 3u^2) + \mathbf{Dp}_k(u^3 - 2u^2 + u) + \mathbf{Dp}_{k+1}(u^3 - u^2)$$

$$= \mathbf{p}_k\, H_0(u) + \mathbf{p}_{k+1}\, H_1(u) + \mathbf{Dp}_k\, H_2(u) + \mathbf{Dp}_{k+1}\, H_3(u). \qquad (3\text{-}20)$$

where $H_k(u)$ for $k = 0, 1, 2, 3$ are the blending functions that blend the boundary constraint values to obtain each coordinate position along the curve.

The disadvantages of Hermite cubic splines are:

- It is often created in an interactive environment and the user has no idea as to the value at the joints of the three derivatives x'(t), y'(t) and z'(t), unlike *Bezier curves* which allow the user to describe these smoothness conditions easily.
- Hermite polynomials suffer from one major defect where the values of the coefficients depend on having derivative information at the endpoints. In most applications, data are given as values at control points rather than as derivatives.

Cardinal splines and Kochanek-Bartels splines are variations on the Hermite splines that do not require input values for curve derivatives at the control points.

3.1.4 BEZIER POLYNOMIALS:

Bezier curves were developed by a French engineer called Pierre Bezier while working for the Renault automobile company in France. They have the properties that make them highly useful and convenient for curve and surface design. They are also very easy to implement. The same as with cubic splines, Bezier curves are blended at the joints.

A Bezier curve can be fitted to any number of control points. The degree of the Bezier polynomial is determined by the number of control points, as well as their relative position. As with cubic splines, the Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending functions.

The position vector ***P(u)***, that is produced by blending $n + 1$ control points is:

$$\mathbf{P}(u) = \sum_{k=0}^{n} p_k \, BEZ_{k,n}(u), \qquad 0 \le u \le 1 \qquad (3\text{-}21)$$

where the blending functions $BEZ_{k,n}(u)$ are the Brenstein polynomials:

$$BEZ_{k,n}(u) = C(n,k)u^k (1-u)^{n-k} \qquad (3\text{-}22)$$

and the $C(n,k)$ are the binomial coefficients:

$$C(n,k) = \frac{n!}{k!(n-k)!} \qquad (3\text{-}23)$$

Bezier blending functions can be defined equivalently with the recursive calculations as:

$$BEZ_{k,n}(u) = (1-u)\, BEZ_{k,n-1}(u) + uBEZ_{k-1,n-1}(u), \qquad n > k \ge 1 \qquad (3\text{-}24)$$

with: $BEZ_{k,k} = u^k$ *and* $BEZ_{0,k} = (1-u)^k$.

Many graphics packages provide only cubic spline functions, which gives reasonable design flexibility while avoiding the increased calculations needed with higher-order polynomials. Cubic Bezier curves, are generated with four control points. The curve passes only through the first and fourth data points, while the two intermediate points are used to define the slope of the curve at the endpoints. Substituting n = 3 in equation (3-22), we obtain the four blending functions for the cubic Bezier curves as:

$$\begin{aligned} BEZ_{0,3}(u) &= (1-u)^3 \\ BEZ_{1,3}(u) &= 3u(1-u)^2 \\ BEZ_{2,3}(u) &= 3u^2(1-u) \\ BEZ_{3,3}(u) &= u^3 \end{aligned} \qquad (3\text{-}25)$$

The polynomial expressions for the blending functions can be expanded and the cubic Bezier point function can be written according to [5] Hearn et. al (1997) as:

$$\mathbf{P}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} . M_{Bez} . \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \qquad (3\text{-}26)$$

where the Bezier matrix $\mathbf{M_{Bez}}$, is:

$$\mathbf{M}_{Bez} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad (3\text{-}27)$$

The advantage of Bezier splines over Cubic splines is that the direction of the curve (the derivatives) at the joints can be defined and changed simply by specifying the second (initial slope) and third (final slope) data points. They are widely used because they are easy to implement and stable. One disadvantage of Bezier curves is that they do not allow for local control, changing a control point not only affects the shape of the curve near that

point, but it affects the entire span of the curve. B-spline curves have this local shape property.

3.1.5 B-spline Curves

B-spline curves are the most widely used class of approximating splines. Given $n + 1$ control points of an input set $\mathbf{p}_k$, the calculation of coordinate positions along a B-spline curve in a blending-function formulation can be expressed according to [5] Hearn et. al (1997) as:

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k \, B_{k,d}(u), \qquad u_{\min} \le u \le u_{\max}, \qquad 2 \le d \le n+1 \qquad (3\text{-}28)$$

where $B_{k,d}$ are polynomial blending functions of degree $d - 1$, where d is any integer value between 2 and $n + 1$. If d is set to 1, then the curve is just a point plot of control points. Blending functions for B-spline curves are defined by Cox-deBoor recursion formulas as in [5] Hearn et. al (1997) as:

$$B_{k,1}(u) = \begin{cases} 1, & \text{if } u_k \le u < u_{k+1} \\ 0, & \text{otherwise} \end{cases} \qquad (3\text{-}29a)$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d+1} - u_k} B_{k,d-1} + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u) \qquad (3\text{-}29b)$$

where each blending function is defined over d subintervals of the total range of u, and it is assumed that any terms evaluated as 0/0 are to be assigned the value of 0.

Properties of B-spline curves are:

1) The polynomial curve has a degree d - 1 and C^{d-2} continuity over the range of u.
2) If we have $n + 1$ control points, the curve is then described with $n + 1$ blending functions.

3) Each blending function $B_{k,d}$ is defined over d subintervals of the total range of u, starting at knot value u_k.

4) The range of parameter u is divided into $n + d$ subintervals by the $n + d + 1$ values specified in the knot vector.

5) With knot values labeled $\{u_0, u_1, \ldots, u_{n+d}\}$, the resulting B-spline curve is defined only in the interval from knot value u_{d-1} up to knot value u_{n+1}.

6) Each section of the spline curve is influenced by d control points.

7) Any one control point can affect the shape of at most d curve sections.

8) Changes to a conrtol point only affects the curve in that locality.

9) Any number of points can be added without increasing the degree of polynomial.

10) As with Bezier curves, adding multiple points at or near a single position draws the curve towards that position.

11) Closed curves can be created, by making the first and last points the same, though continuity will not be maintained automatically.

B-splines are generally described according to the selected knot-vector class: uniform, open uniform and nonuniform.

3.1.5.1 Uniform, Periodic B-splines:

A uniform B-spline curve results when the spacing between knot values is constant. Uniform B-splines have periodic blending functions. All blending functions are the same and each successive blending function is a shifted version of the previous function as in [5] Hearn et. al (1997):

$$B_{k,d}(u) = B_{k+1,d}(u + \Delta u) = B_{k+2,d}(u + 2\Delta u) \qquad (3\text{-}30)$$

where Δu is the interval between adjacent knot values.

Given $d = n = 3$, the knot vector must contain $n + d + 1 = 7$ knot values:

$$\{0, 1, 2, 3, 4, 5, 6\}$$

and the range of parameter u is from 0 to 6, with $n + d = 6$ subintervals. Often knot values are normalized to the range between 0 and 1 as:

$$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}.$$

There are $n + 1 = 4$ blending functions, each spanning $d = 3$ subintervals in the range of u.

3.1.5.2 Open Uniform B-splines:

Open uniform B-splines are cross between uniform and nonuniform B-splines, in which the knot spacing is uniform except at the ends where the knot values are repeated d times, examples are:

$\{0, 0, 1, 2, 3, 3\}$, for $d = 2$ and $n = 3$

$\{0, 0, 0, 0, 1, 2, 2, 2, 2\}$, for $d = 4$ and $n = 4$.

These knot vectors can also be noramalized to the unit interval from 0 to 1.
An open uniform knot vector with integer values can be calculated for any values of d and n, according to [5] Hearn et. al (1997) as:

$$u_j = \begin{cases} 0, & \text{for } 0 \le j < d \\ j - d + 1, & \text{for } d \le j \le n \\ n - d + 2, & \text{for } j > n \end{cases} \qquad (3\text{-}31)$$

where j ranges from 0 to $n + d$.

Open uniform B-splines are very similar to Bezier splines. When $d = n + 1$ (degree of polynamial is n), open B-spline reduce to Bezier splines, with all knot values are either 0 or 1. For example, with a cubic open B-spline, where $d = 4$ and four control points, the knot vector is:

$$\{0, 0, 0, 0, 1, 1, 1, 1\}$$

Figure 3-5 is a C program that prompts the user for the number of control points and the degree of polynomial. It then prompts for the x-coordinates of the control points and calculates the corresponding sine values. It then generates an open uniform knot vector with integer values using (3-31) and calculates the coordinate positions along a B-spline curve in a blending-function formulation.

The open B-spline curve passes through the first and last control points. The slope of the parametric curves at the first control point is parallel to the line connecting the first two control points, and the parametric slope at the last control point is parallel to the line connecting the last two control points. These geometric constraints for matching curve sections are the same as for Bezier curves.

```
/*
  A general expression for the calculation of coordinate positions
  along a B-spline curve (open uniform) in a blending function formulation.
  Here k is the knot number, vector is the knot vector, degree of d-1,n+1
  control points and u is the value at which the function is evaluated.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>

  float P;
  float b, blend(), div();

  #define SIZE 12

  typedef float vector_type[SIZE];

  vector_type knot_x, knot_y;
```

```
float blend (int k, int d, float u, float *knot_x);

main()
{
  int i, k, d, degree, n, points_num;
  float u;
  printf("\n\n Enter number of control points: ");
  scanf("%d", &(points_num));
  n = points_num - 1;
  printf(" Enter degree of polynomial: ");
  scanf("%d", &degree);
  d = degree + 1;
  printf("\n");

  for (k=0, i=0; k<=n+d; k++, i++)
  {
        if (k < d-1)
          knot_x[k] = 0;
        else if (k <= n+d-1)
        {
          printf(" Enter x coordinate of control point number %d : ", (k+2-d));
          scanf("%f", &(knot_x[k]));
        }
        else
          knot_x[k] = knot_x[k-1];
  }

  printf("\n");

  for (k=0, i=0; k<=n+d, i<=n; k++)
  {
        if (knot_x[k] != 0)
        {
          knot_y[i] = sin (knot_x[k]);
          i++;
        }
  }
  printf(" The knots are: \n");
  for (k=0 ; k <= n+d; k++)
        printf (" %f\t", knot_x[k]);
  printf("\n\n The y coordinates are : \n");

  for (i=0; i<=n; i++)
        printf(" %f\t", knot_y[i]);

while (u!= 0)
{
  printf("\n\n Enter a value for u between %f and %f :\t ",
            knot_x[d-1], knot_x[n+d-1]);
  scanf("%f", &u);

  P = 0;
  for (k=0; k<=n; k++)
    P = P + knot_y[k] * blend (k, d, u, knot_x);
```

```
  }
}

/*
  This function is to calculate the blending functions for B-spline
  curves, defined by the Cox-deBoor recursion formulas.
*/
  float blend (int k, int d, float u, float *knot_x)
  {
    float div (float a, float b);
    if (d==1)
          {
            if (knot_x[k] <= u && u < knot_x[k+1])
              return ((float) 1);
            else
              return ((float) 0);
          }
    else
          return ((float) div ((u-knot_x[k])*blend(k, d-1, u, knot_x),
            (knot_x[k+d-1]-knot_x[k]))
           + div (( knot_x[k+d]-u)*blend(k+1, d-1, u, knot_x),
            (knot_x[k+d]-knot_x[k+1])));
  }

/*
  This function is to assign a value of 0 to any blending function
  in case we get 0/0.
*/
  float div ( float a, float b)
  {
    if (a == 0 && b == 0) return (0);
    if (a != 0 && b == 0) return (DBL_MAX);
    return (a/b);
  }
```

Figure 3-5 A general expression for the calculation of coordinate positions along a B-spline curve.

A sample output of the program in Figure 3-5 is shown in Figure 3-6.

```
Enter number of control points: 3
Enter degree of polynomial: 3

Enter x coordinate of control point number 1 : .35
Enter x coordinate of control point number 2 : .6
Enter x coordinate of control point number 3 : .8

The knots are:
0.000000    0.000000    0.000000    0.350000    0.600000
0.800000    0.800000

The y coordinates are :
0.342898    0.564642    0.717356
```

```
Enter a value for u between 0.350000 and 0.800000 :   .5
Enter a value for u between 0.350000 and 0.800000 :   0
Type EXIT to return to Turbo C. . .
```

Figure 3-6 Sample output of the program in Figure 3-5.

3.1.5.3 Nonuniform B-splines

Any values and intervals for the knot vector can be specified in nonuniform B-splines, with multiple internal knot values and unequal spacing between the knot values, such as:

$$\{0, 1, 2, 3, 3, 4\}$$
$$\{0, 2, 2, 3, 3, 6\}$$
$$\{0, 0, 0, 1, 1, 3, 3, 3\}$$
$$\{0, 0.2, 0.6, 0.9, 1.0\}$$

Different shapes for the blending functions in different intervals, which can be used to adjust spline shapes, can be obtained with unequally spaced intervals in the knot vector, and stable variations can be produced with increasing knot multiplicity. To reduce computations, the knot intervals are often restricted to be either 0 or 1.

In general B-spline curves have two advantages over Bezier splines:

1) The degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations).
2) They allow local control over the shape of a spline curve.

Some of the advantages also are that any number of control points can be added or modified to manupilate curve shapes, and the number values in the knot vector can be increased to aid in the curve design. However, to do this, we need to add control points, since the size of the knot vector depends on parameter n. The tradeoff is that B-splines are more complex than Bezier splines.

3.1.6 Newton Form

This has been introduced by Newton. It depends on the interpolation abscissae, namely the $n + 1$ monic polynomial $n_i(x)$ of degree i which vanish at the first i interpolation abscissae :

$$n_i(x) = (x - x_0)...(x - x_{i-1}) \qquad (3\text{-}32)$$

Then the interpolation polynomial can be represented as in [6] Boehm & Prautzsch (1991) as:

$$P(x) = f_0 n_0(x) + f_{0,1} n_1(x) + ... + f_{0,...,n} n_n(x) \qquad (3\text{-}33).$$

The $f_{0,...,i}$ are called divided differences and satisfy the recurrence relation:

$$f_{i,...,k} = \frac{f_{i,...,k-1} - f_{i+1,...,k}}{x_i - x_k} \qquad (3\text{-}34)$$

Figure 3-7 shows Newton's Scheme in which the calculation of $f_{0,...,i}$ using equation (3-34) is effected best. The underlined values in the figure are the ones involved in the computation of $f_{1,2,3}$. The advantage of Newton representation is that the degree of the interpolation polynomial can be easily increased or decreased.

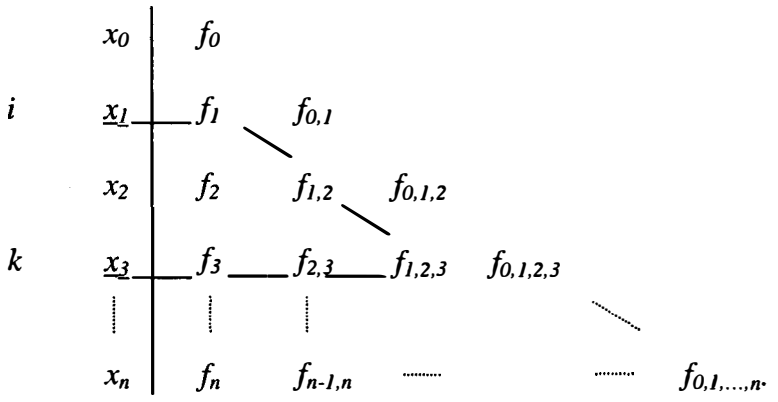

Figure 3-7 Newton's Scheme.

Example 1:

As an example, suppose we have the points:

x_i	0	1	2
f_i	8	5	4

At x = 3: $n_0 = 1$, $n_1(3) = (3-0) = 3$, $n_2(3) = (3-0)(3-1) = 6$.

So the divided difference table is:

0	8		
1	5	-3	
2	4	-1	1.

And $f(3) = (8.1) - (3.3) + (1.6) = 5$.

A program written in C to evaluate the interpolation polynomial using Newton's method is shown in Figure 3-8.

```
/*
  A general expression for the calculation of coordinate positions
  along curve using Newton's Method.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>

  #define SIZE 8
  typedef float vector_type[SIZE];
  vector_type knot_x, knot_y;

/*
  This function is to assign a value of 0 to any blending function
  in case we get 0/0.
*/

  float div ( float a, float b)
  {
    if (a == 0 && b == 0) return (0);
    if (a != 0 && b == 0) return (DBL_MAX);
```

```
    return (a/b);
  }

/*
  This function is to calculate the divided differences.
*/

  float differ(int i, int k, float x, float *knot_x, float *knot_y)
  {
    float div (float a, float b);

    if (i==k)
          return((float) knot_y[i]);
    if (i==k+1)
          return ((float) div((knot_y[i]-knot_y[k]), (knot_x[i]-knot_x[k])));
    if (i<k+1)
          return ((float) div((differ(i, k-1, x, knot_x, knot_y)-
            differ(i+1, k, x, knot_x, knot_y)),(knot_x[i]-knot_x[k])));
  }

/*
  This function is to introduce the n+1 polynomials of degree i.
*/

  float pol(int k, float x, float *knot_x)
  {
    int i;
    float n;

    if (k==0)
          return ((float) 1);
    if (k==1)
          return (x-knot_x[k-1]);
    if (k>1)
          n = 1.0;
          for (i=0; i<=k-1; i++)
            n = n * (x-knot_x[i]);
          return n;
  }

  main()
  {
    int points_num, n, k, i=0;
    float x, P;
    printf("\n\n Enter number of control points: ");
    scanf("%d", &(points_num));
    n = points_num - 1;
    for (k=0; k<=n; k++)
    {
          printf("\n Enter x coordinate of point number %d : ", (k+1));
          scanf("%f", &(knot_x[k]));
          printf(" Enter y-coordinate of point number %d : ", (k+1));
          scanf ("%f", &(knot_y[k]));
    }
    printf("\n The knots are: \n");
```

```
    for (k=0 ; k<=n; k++)
        printf ("\n%f\t%f", knot_x[k], knot_y[k]);
    printf("\n Enter a point at which the function is to be evaluated: ");
    scanf("%f", &x);

P = 0;
    for (i=0; i<=n; i++)
        P = P + differ(0, i, x, knot_x, knot_y) * pol(i, x, knot_x);
    printf("\n The function at %f = %f", x, P);
}
```

Figure 3-8 A C program to calculate the coordinate positions along a curve by Newton's Method.

The program in Figure 3-8 used to solve the problem in *Example 1* above and the outcome is shown in Figure 3-9.

```
Enter number of control points: 3

Enter x coordinate of point number 1 : 0
Enter y-coordinate of point number 1 : 8

Enter x coordinate of point number 2 : 1
Enter y-coordinate of point number 2 : 5

Enter x coordinate of point number 3 : 2
Enter y-coordinate of point number 3 : 4

 The knots are:

0.000000      8.000000
1.000000      5.000000
2.000000      4.000000
Enter a point at which the function is to be evaluated: 3

The function at 3.000000 = 5.000000
```

Figure 3-9 Solution of example (1) using the C program of Figure 3-8.

The program in Figure 3-8 can be modified a bit so as to calculate the sine function of the control points. Then to approximate the function of a point keyed in by the user by interpolation. It keeps prompting for points to be evaluated until the user key in zero. Then the program exits. This program is shown in Figure 3-10.

```
/*
  A general expression for the calculation of coordinate positions
  along curve using Newton's Method. It evaluates the sine value of
  the control points, then prompts the user for the point at which
  the function is to be evaluated, and exits when 0 is entered for
  that point.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>


  #define SIZE 8
  typedef float vector_type[SIZE];
  vector_type knot_x, knot_y;


/*
  This function is to assign a value of 0 to any blending function
  in case we get 0/0.
*/

  float div ( float a, float b)
  {
    if (a == 0 && b == 0) return (0);
    if (a != 0 && b == 0) return (DBL_MAX);
    return (a/b);
  }

/*
  This function is to calculate the divided differences.
*/

  float differ(int i, int k, float x, float *knot_x, float *knot_y)
  {
    float div (float a, float b);

    if (i==k)
          return((float) knot_y[i]);
    if (i==k+1)
          return ((float) div((knot_y[i]-knot_y[k]), (knot_x[i]-knot_x[k])));
    if (i<k+1)
          return ((float) div((differ(i, k-1, x, knot_x, knot_y)-
            differ(i+1, k, x, knot_x, knot_y)),(knot_x[i]-knot_x[k])));
  }

/*
  This function is to introduce the n+1 polynomials of degree i.
*/

  float pol(int k, float x, float *knot_x)
  {
    int i;
```

```
  float n;

  if (k==0)
        return ((float) 1);
  if (k==1)
        return (x-knot_x[k-1]);
  if (k>1)
        n = 1.0;
        for (i=0; i<=k-1; i++)
          n = n * (x-knot_x[i]);
        return n;
}

main()
{
  int points_num, n, k, i=0;
  float x = 1.0, P;
  printf("\n\n Enter number of control points: ");
  scanf("%d", &(points_num));
  n = points_num - 1;
  for (k=0; k<=n; k++)
  {
        printf(" Enter x coordinate of point number %d : ", (k+1));
        scanf("%f", &(knot_x[k]));
        knot_y[k] = sin (knot_x[k]);
  }
  printf("\n The knots are: ");
  for (k=0 ; k<=n; k++)
        printf ("\n%f\t%f", knot_x[k], knot_y[k]);
  while (x != 0)
  {
  printf("\n Enter a point at which the function is to be evaluated: ");
  scanf("%f", &x);

  P = 0;
  for (i=0; i<=n; i++)
        P = P + differ(0, i, x, knot_x, knot_y) * pol(i, x, knot_x);
  printf(" The function at %f = %f", x, P);
  }
}
```

Figure 3-10 Evaluating sine function using Newton's Method.

The previous program is executed to approximate the sine function for a number of points and the results are shown in Figure 3-11.

```
Enter number of control points: 4
Enter x coordinate of point number 1 : 0.2
Enter x coordinate of point number 2 : 0.6
Enter x coordinate of point number 3 : 1.0
Enter x coordinate of point number 4 : 1.4

The knots are:
0.200000    0.198669
0.600000    0.564642
1.000000    0.841471
1.400000    0.985450
Enter a point at which the function is to be evaluated: 0.2
The function at 0.200000 = 0.198669
Enter a point at which the function is to be evaluated: 0.3
The function at 0.300000 = 0.296130
Enter a point at which the function is to be evaluated: 0.4
The function at 0.400000 = 0.390067
Enter a point at which the function is to be evaluated: 0.5
The function at 0.500000 = 0.479799
Enter a point at which the function is to be evaluated: 0.6
The function at 0.600000 = 0.564642
Enter a point at which the function is to be evaluated: 0.7
The function at 0.700000 = 0.643914
Enter a point at which the function is to be evaluated: 0.8
The function at 0.800000 = 0.716931
Enter a point at which the function is to be evaluated: 0.9
The function at 0.900000 = 0.783011
Enter a point at which the function is to be evaluated: 1.0
The function at 1.000000 = 0.841471
Enter a point at which the function is to be evaluated: 1.1
The function at 1.100000 = 0.891628
Enter a point at which the function is to be evaluated: 1.2
The function at 1.200000 = 0.932798
Enter a point at which the function is to be evaluated: 1.3
The function at 1.300000 = 0.964300
Enter a point at which the function is to be evaluated: 1.4
The function at 1.400000 = 0.985450
Enter a point at which the function is to be evaluated: 0
```

Figure 3-11 The results of executing the program in Figure 3-10.

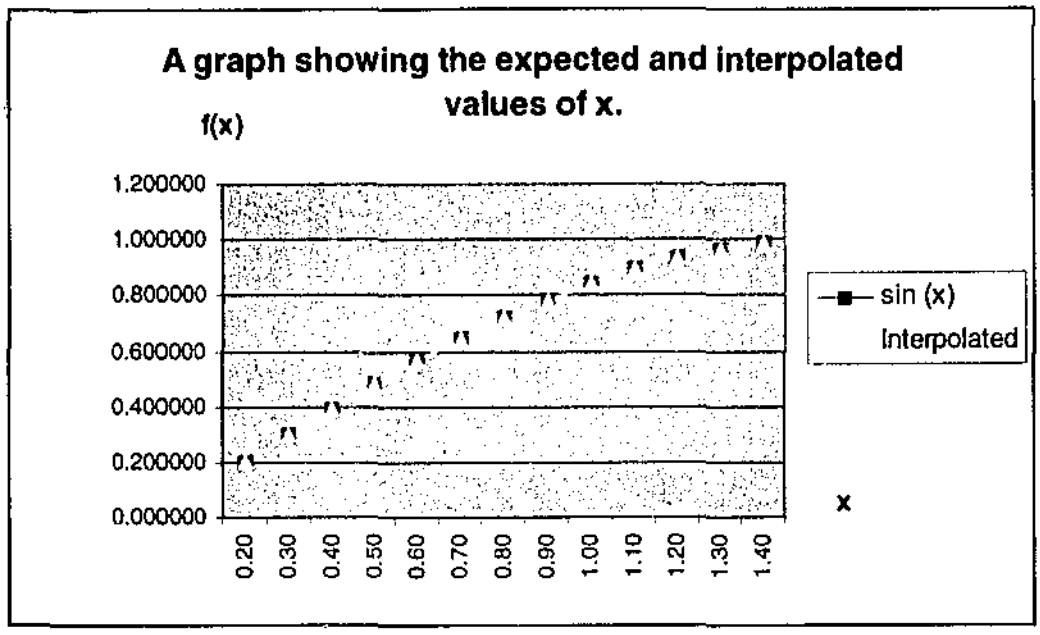

Figure 3-12 Expected and Interpolated values versus X.

The previous results are graphed in Figure 3-12, in which the accuracy of the approximation can be noticed.

3.1.7 The Chosen Interpolating Method

From the preceding sections we can conclude that most of the methods introduced can give good approximations. Each one of them has advantages as well as disadvantages over the others. In general, cubic splines compared to higher-order polynomials, require less calculations and memory and are more stable, while compared to lower-order polynomials, they are more flexible for modeling arbitrary curve shapes.

The advantage of Bezier splines over Cubic splines is that the direction of the curve at the joints can be defined and changed simply by specifying the second and third data points. They are widely used because they are easy to implement and stable. One disadvantage of Bezier curves is that they do not allow for local control, changing a control point not only

affects the shape of the curve near that point, but it affects the entire span of the curve. B-spline curves have this local shape property.

The main advantages of B-spline curves over Bezier splines:
The degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations). They allow local control over the shape of a spline curve. Any number of control points can be added or modified to manupilate curve shapes, and the number values in the knot vector can be increased to aid in the curve design. However, to do this, we need to add control points, since the size of the knot vector depends on parameter n. The tradeoff is that B-splines are more complex than Bezier splines.

Finally, Lagrange and Newton polynomials proved to be accurate and the easiest to implement. For the purpose of simulation, Lagrange interpolation was used as shown in Appendix A.

Chapter 4

The Controller

The controller provides the interpolator with information needed to perform the computations according to equation (3-9). The controller in Figure 2-1 is shown with expanded details in Figure 4-1. It is responsible for determining and providing the basepoint index m_k, and the fractional interval μ_k, based on the output of a timing estimator or a timing error estimator, to the interpolator. Once these have been identified, the selected signal and impulse response samples are loaded into the interpolation filter structure for computations.

A number of controlled oscillator (NCO) can provide the necessary control assuming that the signal samples are uniformly clocked through a shift register at a rate $1/T_s$ and the NCO is clocked at a rate synchronized to $1/T_s$. The NCO clock period is T_s and its average period is T_i. The basepoint index is identified by flagging the correct set of signal samples rather than explicitly computing m_k.

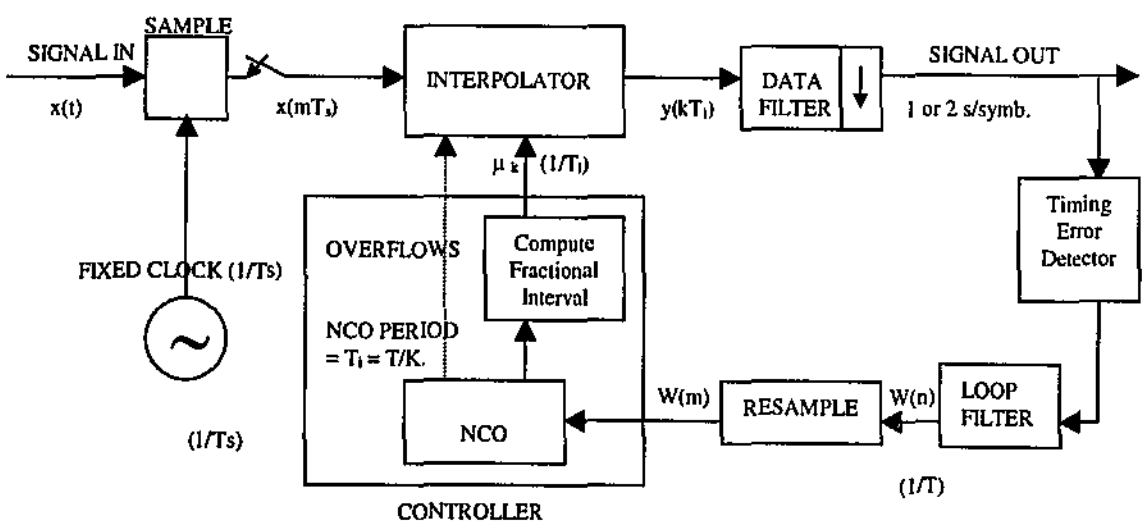

Figure 4-1 The controller with expanded detail.

4.1 Fractional Interval μ_k

Upon recycling, the contents of NCO's register are used to calculate the fractional interval μ_k. At the mth clock tick, the NCO register contents computed are designated as $\eta(m)$ and the control word as $W(m)$. the NCO difference equation as in [1] is:

$$\eta(m) = [\eta(m-1) - W(m-1)] \bmod 1. \qquad (4-1)$$

Under loop equilibrium conditions, $W(m)$ is nearly constant. The contents of the NCO register is decremented by $W(m)$ every T_s seconds, and on average the register underflows every $1/W(m)$ clock ticks. So the NCO period is $Ti = Ts/W(m)$ and in the absence of any disturbance the control word assumes its correct value:

$$W(m) \cong \frac{T_s}{T_i}. \qquad (4-2)$$

$W(m)$ is the synchronizer's estimate of the average frequency of interpolation $1/T_i$, expressed relative to the sampling frequency $1/T_s$. It is an estimate because it is produced from filtering of multiple, noisy measurements of timing error.

The fractional interval as given in [1] is:

$$\mu_k \frac{\eta(m_k)}{1 - \eta(m_k + 1) + \eta(m_k)} = \frac{\eta(m_k)}{W(m_k)}. \qquad (4-3)$$

Equation (4-3) is an estimate of the exact fractional interval because its constituents $W(m_k)$ and $\eta(m_k)$ are both estimates of the true frequency and phase.

4.2 Alternative Control Method

[4] Moeneclaey, Meyr & Fechtel (1998) suggested an alternative control method without using an NCO. Two successive interpolants are performed for time instants:

$$kT_i = (m_k + \mu_k)T_s \qquad (4-4)$$

$$(k+1)T_i = (m_{k+1} + \mu_{k+1})T_s \qquad (4-5)$$

Subtracting and rearranging the two equations in (4-4) and (4-5) yields the recursion for the estimates as in [1]:

$$m_{k+1} = m_k + T_i/T_s + \mu_k - \mu_{k+1}.$$

$$m_{k+1} = m_k + w(m_k) + \mu_k - \mu_{k+1}. \qquad (4\text{-}6)$$

where as in [4] T_i/T_s is approximated by $w(m_k)$. The increment in sample count from one interpolation to the next is:

$$m_{k+1} - m_k = \text{int}[w(m_k) + \mu_k] \qquad (4\text{-}7)$$

and

$$\mu_{k+1} = [\mu_k + w(m_k)]_{\text{mod}1} \qquad (4\text{-}8)$$

A program written in C that can do the computation of the controller and provide the increments of $m_{k+1} - m_k$ and μ_k for different ratios of T_i/T_s, using this method is shown in Figure 4-2.

```
#include <stdio.h>
#include <math.h>


main()
{
  float frac_delay;
  int base_point, next_base, base_diff;
  float tme, Ts, Ti;
  float control_word;
  int count;

  printf(" Enter Ti, Ts, mu and m: \n");
  scanf("%f %f %f %d", &Ti, &Ts, &frac_delay, &base_point);

  control_word = Ti / Ts;
  printf(" W is %f\n", control_word);

  for (count = 0; count < 8; count++)
  {
    next_base = (int) (base_point + frac_delay + control_word);
    base_diff = next_base - base_point;
    base_point = next_base;
    frac_delay = frac_delay + control_word - (int) (frac_delay + control_word);

    printf(" Base diff  = %d", base_diff);
    printf(" Frac delay = %f\n", frac_delay);
  }

}
```

Figure 4-2 A C program to do the calculations of the controller.

The previous program was executed twice and the results were as follows:

a) For $T_i / T_s = 1.575$, $\mu_0 = 0$, the output is shown in Figure 4- 3.

```
Enter Ti, Ts, mu and m:
1.575 1 0 0
 W is 1.575000
 Base diff  = 1 Frac delay = 0.575000
 Base diff  = 2 Frac delay = 0.150000
 Base diff  = 1 Frac delay = 0.725000
 Base diff  = 2 Frac delay = 0.300000
 Base diff  = 1 Frac delay = 0.875000
 Base diff  = 2 Frac delay = 0.450000
 Base diff  = 2 Frac delay = 0.025000
 Base diff  = 1 Frac delay = 0.600000
```

Figure 4- 3 The result for Ti/Ts = 1.575.

b) For $T_i / T_s = 1 - 10^{-5}$, $\mu_0 = 4.1 * 10^{-5}$, the results are shown in Figure 4- 4.

```
Enter Ti, Ts, mu and m:
0.99999 1 0.000041 0
 W is 0.999990
 Base diff  = 1 Frac delay = 0.000031
 Base diff  = 1 Frac delay = 0.000021
 Base diff  = 1 Frac delay = 0.000011
 Base diff  = 1 Frac delay = 0.000001
 Base diff  = 0 Frac delay = 0.999991
 Base diff  = 1 Frac delay = 0.999981
 Base diff  = 1 Frac delay = 0.999971
 Base diff  = 1 Frac delay = 0.999961
```

Figure 4- 4 The result for Ti/Ts = 0.99999

When the designated value of T_i/T_s is an exact integer, the actual value in practice will be slightly different due to clock tolerances. In this case, the fractional delay changes very slowly as a function of time. On the other hand, if the design of the value is not an integer, then the fractional delay varies more rapidly with time. In a hardware implementation, it can be dept constant over a large number of symbols which simplifies the hardware. This control method is most useful in systems where the data are consumed at the same location as the data receiver, without reclocking.

4.3 System Description

The block diagram of the digital receiver system considered for simulation results is shown in Figure 4-5.

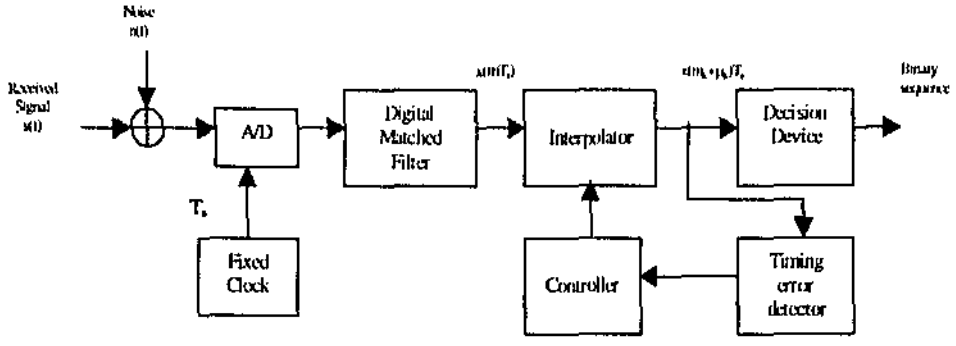

Figure 4-5 Digital Interpolating Receiver.

The transmitted signal *s(t)* passes through an additive white Gaussian noise (AWGN) channel before it is sampled by the fixed clock at uniform intervals of T_s seconds. The number of samples per symbol is assumed to be an integer. The samples pass through a digital matched filter whose continuous time impulse response is described as:

$$h(t) = \begin{cases} ks(T-t); & 0 \le t \le T \\ 0 \quad ; & elsewhere \end{cases} \tag{4-9}$$

Signal-to-noise ratio of the received signal is maximised by the digital matched filter at the filter output. The interpolator then uses the samples from the matched filter to calculate the interpolated point. The *m*th sample from the filter output is denoted by $x(mT_s)$.

Cubic interpolation, based on the Lagrange formula is used by the interpolator. The interpolator provides an approximation of the output sample of the matched filter, at the optimal decision instant. The controller calculates the fractional delay and passes it to the interpolator. The fractional delay is varied to simulate the situation where the controller calculates an incorrect value of μ_k. The output of the interpolator is fed to a detection device and is used to calculate the bit error rate (BER).

The transmitted signal $s(t)$ considered in this paper is a bipolar baseband pulse amplitude modulated (PAM) signal with raised cosine pulse shape of the form [4]:

$$s(t) = \cos^2 \frac{\pi t}{2T}, \quad |t| < T \tag{4-10}$$

where T is the symbol period.

4.4 Simulation Results

A program written in C++ mainly by Ms. Vis Ramakonar is included in appendix A. The number of samples per symbol is assumed to be an integer and sampling begins at $t = 0$. Simulations were carried out for a sampling rate of 5 samples per symbol for 1 million symbols. The signal to noise ratio was varied between 2dB and 14dB, and the fractional delay, μ_k, was varied from 0 to 0.95. The correct value of μ_k in this case is 0, which is the case for ideal interpolation, when one of the existing samples is the interpolated point.

A timing error reduces the useful signal component at the input of the decision device, and also introduces intersymbol interference (ISI). The sample at the input of the decision device consists of a useful signal term, an ISI term and an additive noise term, $v_k(\mu_k)$, which is according to Bucket & Moeneclaey (1994), sited in[8]:

$$v_k(\mu_k) = \sqrt{E_b}(a_k s(\mu_k) + ISI(\mu_k)) + \sqrt{N_o / 2\sigma}(\mu_k)\omega_k \tag{4-11}$$

where $v_k(\mu_k)$ is the sample at the input of the decision device, E_b is the energy per bit, a_k is the data symbol, $s(\mu_k)$ is the useful signal amplitude, $ISI(\mu_k)$ is the intersymbol interference term, N_o is the Gaussian noise with ω_k being a zero–mean univariate Gaussian random variable, and $\sigma^2(\mu_k)$ is the noise variance.

The useful signal amplitude , $s(\mu_k)$ as in [8]:

$$s(\mu_k) = \sum_{i=-N_1}^{N_2} h_I(i;\mu_k)\, g(-\mu_k T_s - iT_s) \tag{4-12}$$

where g(t) is the output of the matched filter.

The ISI term as in [8] is:

$$ISI(\mu_k) = \sum_{p\neq 0} a_{k-p} \sum_{i=-N_1}^{N_2} h_I(i;\mu_k)\, g(-\mu_k T_s + pT_s - iT_s) \tag{4-13}$$

and the noise variance is given by:

$$\sigma^2(\mu_k) = \sum_{i,j=-N_1}^{N_2} h_I(i;\mu_k) h_I(j;\mu_k)\, g(-iT_s + jT_s) \tag{4-14}$$

The BER corresponding to given values of E_b/N_o and K is given by:

$$P((E_b/N_o)\ ;K) = E[P(f^2(\mu)E_b/N_o;K)] \tag{4-15}$$

where

$$f(\mu) = (s(\mu) + ISI(\mu))/\sigma(\mu) \tag{4-16}$$

and K is the carrier-to-multipath ratio. When K = ∞ the AWGN channel is obtained. The quantity E_b/N_o is the signal-to-noise ratio (SNR). For ideal interpolation, $s(\mu_k) = \sigma^2(\mu_k) =$ 1 and $ISI(\mu_k) = 0$.

For binary PAM, the theoretical expression for the BER in the case of ideal interpolation according to Haykin (1994) sited in [8] is:

$$BER = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{4-17}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} \exp\left(-\frac{u^2}{2}\right) du \tag{4-18}$$

Figure 4-6 shows a plot of the BER versus SNR for different values of μ for 5 samples per symbol as in [8]. Obviously variations in μ can affect the performance of the system significantly. For an SNR of 5dB, when $\mu = 0.25$, BER is increased by a factor of 10 dB, and when $\mu = 0.75$, BER is increased by a factor of about 100 dB.

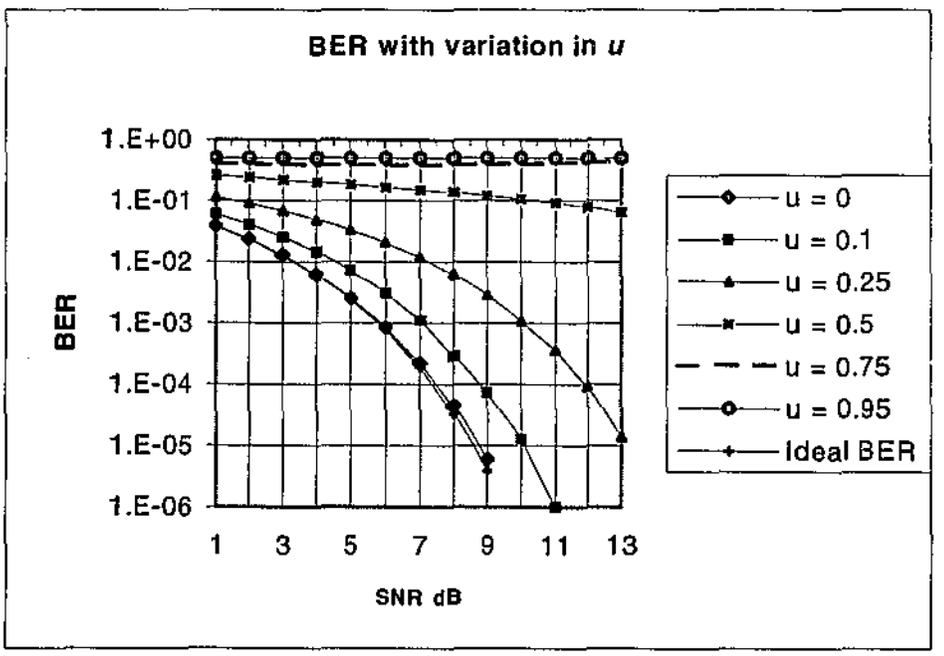

Figure 4-6 BER with Variation in μ_k for 5 samples per symbol.

The theoretical BER for ideal interpolation is also plotted in the figure. It can be observed that the simulated results have a close match to the theoretical BER values for ideal interpolation.

The same simulations were performed again with 10 samples per symbol. The results are plotted in Figure 4-7 as in [9]. They indicate that increasing the number of samples per symbol improves the BER more visibly when the estimation of μ is less accurate, and for higher signal to noise ratios. But even so, the improvement is not very large.

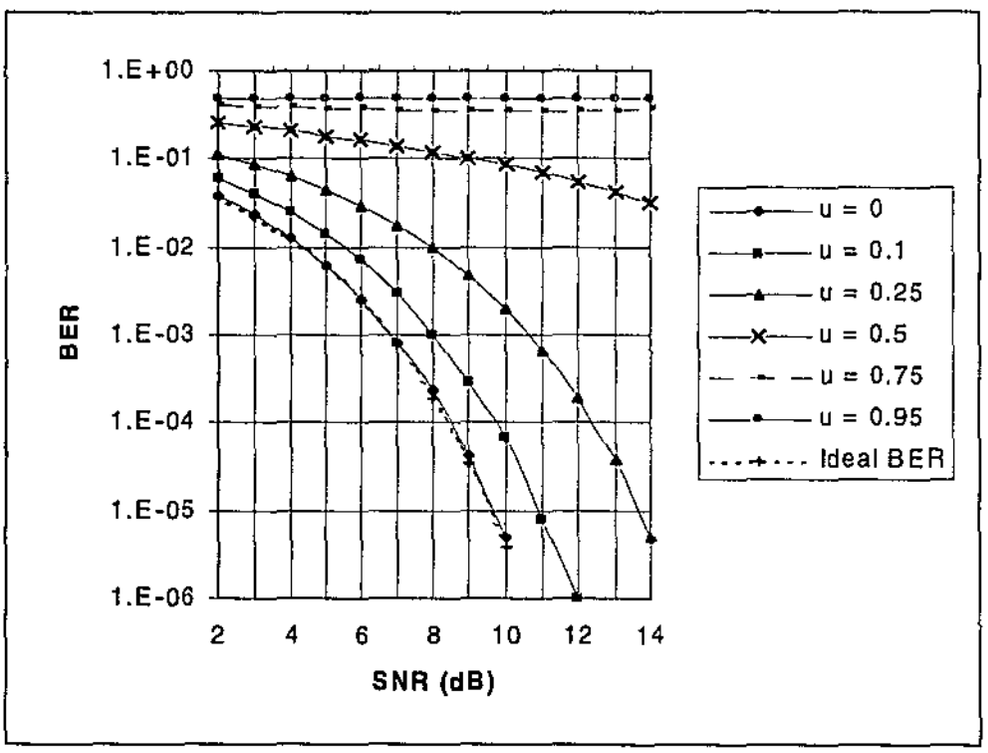

Figure 4-7 BER with Variation in μ_k for 10 samples per symbol.

Chapter 5

Conclusion

In a digital receiver, if sampling is not synchronized with the data symbols, timing must be adjusted by interpolating new samples among the original ones. Interpolation in that case actually combines interpolation and subsequent decimation by resampling. Interpolants can be computed digitally from the input samples and the knowledge of the sampled impulse response of the fictitious analog filter. Equation (3-9) underlies interpolation operations in digital modems.

A number controlled oscillator NCO provides the interpolator with the values of basepoint index m_k and the fractional interval μ_k.

In this research, it has been shown that the performance of a fixed clock digital receiver is highly dependent on accurate calculation of the fractional interval, μ, which estimates the position of the ideal sampling instance with respect to the actual samples taken by the fixed clock sampler. For a typical set of system parameters it was found that the errors of up to about 10% in the value of μ does not degrade the performance of the system significantly. Errors greater than 10% in the estimation of μ can increase the BER of the system to unacceptable levels, in the order of several hundred to several thousand times greater than the BER corresponding to accurate calculation of μ.

It has also been shown that increasing the sampling rate does not improve the BER dramatically. Therefore it is not a good design practice to opt for a higher sampling rate to compensate for performance degradation resulting from inaccurate estimation of μ.

Conclusions found out through out this research are useful in setting the performance criteria for the design of the timing error detector and the controller in a fixed clock digital receiver.

Appendix A

Figure A-1 and Figure A-2 show the header file and the program file written in C++ that are used for simulations.

```
#define    RAN_THRESHOLD 0.5
#define    BIT_POS 1
#define    BIT_NEG -1

#define                PI 3.141592654

// Do not change these values unless you are a better programmer
// than Donald E. Knuth!!!

const long   MBIG  = 1000000000;
const long   MSEED = 161803398;
const int    MZ    = 0;
const double FAC   = (1.0/MBIG);

class RAN_GEN
{
 int  inext,inextp;
 long ma[56];
 long mj,mk;
 int  i,ii,k;

 public:
float output(); // will return a value between 0 and 1 //
 void  initialise(int idum);
 int   bit();
 //void  test(int number_tests);
};

double process(double numsymb,RAN_GEN ran_num_gen, double SNR, double samp_symb, double
trans_symb, int bits);
int    genrand(void);
double quantbits(double num, int bits);
double raisedcos(double t, int T);
double scale (int samp_symb, double base[2][4], int numbase, int SNR);
void   noisegen(double sigma, double nawgn[2], RAN_GEN noise_ran);
void interpolate(double match1[2][4], double match2[2][4], double inter_time, double y[2]);

//Interpolation
void interpolate(double match1[2][4], double match2[2][4], double inter_time, double y[2])
{
    int   i,j;
    int   I1, I2;
    double Ci1, Ci2;
    double ti1, ti2;
    double tj1, tj2;

    I1 = 1;
```

```
    I2 = 4;

    y[0] = 0.0;
    y[1] = 0.0;
    inter_time = inter_time ;

    for (i=I1-1; i<I2; i++)
    {
            ti1 = match1[0][i];
            Ci1 = 1;
            ti2 = match2[0][i];
            Ci2 = 1;

            for (j=I1-1; j<I2; j++)
            {
                    tj1 = match1[0][j];
                    tj2 = match2[0][j];
                    if (j != i)
                    {
                            //numtime is the time at which we
                            //calculate the interpolated point
                            Ci1 *= (((inter_time) - tj1)/(ti1 - tj1)) ;
                            Ci2 *= (((inter_time) - tj2)/(ti2 - tj2)) ;
                    }
            }
            y[0] += (Ci1 * match1[1][i]);
            y[1] += (Ci2 * match2[1][i]);
    }
}

    //Quantises a value to a certain number of bits
double quantbits(double num, int bits)
{
    double rnum;
    double bit_pow;
    double step_size;
    double step_num;

    if (num >= 1.0)
            rnum = 1;
    else if (num <= -1.0)
            rnum = -1;
    else
    {
            step_size = 2.0/pow(2,bits);
            step_num = ceil(num/step_size);
            rnum = step_num*step_size;
    }

    //bit_pow = pow(2,(bits-1.0));
  //rnum = (floor(bit_pow*num)/bit_pow) + (1.0/pow(2,bits));
    return (rnum);
}

//Calculates the raised cosine of a time point
```

```
double raisedcos(double t, int T)
{
      double rcos;
      double x;
      double pi;

   pi = 3.141592654;
      x = (pi*(t))/(2.0*T);
      rcos = cos(x)*cos(x);
      return (rcos);
}

// This routine was modified from the one presented in
//Numerical Algorithms in C, p213., The modification makes it conform to OOP.
//Author W.H. Press (et.al), Modified by Joseph Austin-Crowe 9/3/97
//
//
float RAN_GEN::output()
{

 if (++inext  == 56) inext  = 1;
 if (++inextp == 56) inextp = 1;
 mj = ma[inext] - ma[inextp];
 if (mj < MZ)
 {
   mj += MBIG;
 }
 ma[inext] = mj;

 return(mj*FAC);
};

void RAN_GEN::initialise(int idum)
{

 mj = MSEED-(idum < 0 ? -idum : idum);
 mj %= MBIG;
 ma[55] = mj;
 mk = 1;
 for (i=1 ; i<=54 ; i++)
 {
   ii = (21 * i) % 55;
   ma[ii] = mk;
   mk = mj - mk;
   if (mk < MZ)
   {
     mk += MBIG;
   }
   mj=ma[ii];
 }
 for (k = 1 ; k <= 4 ; k++)
 {
   for (i = 1 ; i <= 55 ; i++)
   {
     ma[i] -= ma[ 1 + (i + 30) % 55];
```

```
    if (ma[i] < MZ)
    {
      ma[i] += MBIG;
    }
  }
  inext  = 0;
  inextp = 31;
  idum  = 1;
 }
};

//This function generates the random bits (either +1 or -1)

int RAN_GEN::bit()
{
 if (output()> RAN_THRESHOLD)
       return(BIT_POS);
 else
       return(BIT_NEG);
};

//Scales the AWGN noise
double scale (int samp_symb, double base[2][4], int numbase, int SNR)
{
     double power;
     int   scale_loop;
     double sigma;
     double ssig;

     power = 0;

     for (scale_loop = 0; scale_loop < numbase; scale_loop++)
     {
               //power += 0.;
               //power += (base[1][scale_loop])*(base[1][scale_loop]);
     }
     power = power / numbase;
     ssig = (samp_symb/(2.0 * SNR)) * power;
     sigma = sqrt(ssig);
     return (sigma);
}

//AWGN noise generator
//transforms uniform to gaussian

void noisegen(double sigma, double nawgn[2], RAN_GEN noise_ran)
{
   double lambda1,lambda2,V1,V2,S;

      lambda1 = noise_ran.output();
      lambda2 = noise_ran.output();
      V1 = 2 * lambda1 - 1.0;
      V2 = 2 * lambda2 - 1.0;
      S = (V1*V1) + (V2*V2);
      while (S >= 1.0)
```

```
    {
            S=0;
     lambda1 = noise_ran.output();
    lambda2 = noise_ran.output();
    V1 = 2.0 * lambda1 - 1.0;
    V2 = 2.0 * lambda2 - 1.0;
    S = (V1*V1) + (V2*V2);
    }
    //printf("sigma is %f\n",sigma);
    nawgn[0] = (V1*sqrt((-2.0*log(S))/S))*sigma;
    nawgn[1] = (V2*sqrt((-2.0*log(S))/S))*sigma;
}
```

Figure A-1 The Header File main_includes.h

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "main2_includes.h"


main(int argc, char *argv[])
{
        double numsymb;
        double Bit_Err;
        double sim_SNR2;
        int    sim_SNR;
        double sim_samp_symb; // fixed sampling clock rate
        double sim_trans_symb;  //transmitter symbol rate
        int    numbits;

        RAN_GEN ran_num_gen;
        numsymb = atof(argv[1]);
        sim_SNR = atof(argv[4]);
        sim_trans_symb = atof(argv[3]);
        sim_samp_symb = atof(argv[2]);

        ran_num_gen.initialise(-2); // The one and only random number generator!!
   numbits=3;
   for (sim_SNR = 2; sim_SNR < 15; sim_SNR++)
        {
           printf("for a SNR of %d ", sim_SNR);
           sim_SNR2 = pow(10.0 , (sim_SNR/10.0)); // convert SNR from dB
                Bit_Err = process(numsymb, ran_num_gen, sim_SNR2, sim_samp_symb,
sim_trans_symb,numbits);
                printf("the BER is %f %%\n",Bit_Err);//*100.0));
        }
  return(0);
 }


// Interpolation
```

```
double process(double numsymb,  RAN_GEN ran_num_gen, double SNR, double samp_symb,
double trans_symb, int bits)
{
        double inter_loop;
        double numtime;
        int             base_loop;
        int             present_bit;
        double leftmost_sample, leftmost_sample1;
        double tme, tme2;
        double bawgn[2];
    double base[2][4], match_base1[2][4], match_base2[2][4],coeff1[5], coeff2[5];
        double sigma,Eb;
        int   numbase;
        int   period;
        int   error, signal;
        double Z[2], BER;
        double total_noise, noise_bandwidth;
        double total_signal;
        double  h1t, h2t;
        double interval;
        double integral;
        int   match_loop1, coeff_loop, match_loop2, match_loop3, match_loop4;
        int   noise_index;
        double const1, const2, const3, const4;
        double const5, const6, const7, const8, const9;
        double signal1, signal2;
        double frac_delay;
        double mk;
        double Ti, Ts;

        numbase = 4;
        period = 1;
        noise_bandwidth = 0;
        const1 = -3.0/(8.0*PI);
        const2 = 1.0/(2.0*PI);
        const3 = 1.0/(16.0*PI);
        const4 = 2.0*PI;
        const5 = 3.0/8.0;
        const8 = 1.0/8.0;

        error = 0;
        total_signal =0;
        total_noise  = 0;
        integral = 0;
        frac_delay = 0;
        Ts = period/samp_symb;
        Ti = trans_symb;

   // set up the 'queue' of three bits..
   //present_bit = ran_num_gen.bit();
   //future_bit  = ran_num_gen.bit();

        for (inter_loop = 0; inter_loop<numsymb; inter_loop++)
        {
```

```
present_bit = ran_num_gen.bit();
numtime  = inter_loop;

        leftmost_sample1 = (floor(numtime * samp_symb) - 1)*period;
        leftmost_sample = (- 1*period);
        for (base_loop = 0; base_loop < numbase; base_loop++)
        {
        integral = 0;
              tme = (leftmost_sample + base_loop) / samp_symb;
              tme2 = (leftmost_sample1 + base_loop) / samp_symb;
              const6 = PI-(PI*tme2);
              const7 = 2.0*PI*tme2;
              const9 = sin(const6);

      h1t = (const1*const6)-(const2*const9)-(const3*sin(const4-const7));
              h2t = ((-const6-const9)/(const4));

              noise_bandwidth = h1t/(h2t*h2t);
              Eb = (const5*tme2)+ (1/(2*PI))*sin(PI*tme2)+(1/(16*PI))*sin(2*PI*tme2);
     sigma = sqrt((samp_symb*numsymb*1.29)/(2.0*numsymb*samp_symb*SNR));

              base[0][base_loop] = tme2;
              match_base1[0][base_loop] = base[0][base_loop];
              match_base2[0][base_loop] = base[0][base_loop];

              //for the matched filter integration step size
              interval = tme/10.0;
    }

  //Matched Filter coefficients
  for (coeff_loop = 0; coeff_loop < samp_symb; coeff_loop++)
  {
        coeff1[coeff_loop] = raisedcos((2- coeff_loop)/samp_symb,period);
        coeff2[coeff_loop] = raisedcos((2- coeff_loop)/samp_symb,period)*-1;
  }

  //Get first sample (t = -0.2)
  match_base1[1][0] = 0;
  match_base2[1][0] = 0;
  signal1 = 0;

  noisegen(sigma, bawgn , ran_num_gen);
  for (match_loop1 = 0; match_loop1 < 2; match_loop1++)
        {
        noise_index = match_loop1;
        //signal1 = raisedcos((-1-match_loop1)/samp_symb,period) * present_bit;
        signal1  =  quantbits(((raisedcos((-1-match_loop1)/samp_symb,period)  *  present_bit)
+bawgn[noise_index]),bits);
        //signal1     =     raisedcos((-1-match_loop1)/samp_symb,period)     *     present_bit
+bawgn[noise_index];
        match_base1[1][0] += signal1 * coeff1[match_loop1];
        match_base2[1][0] += signal1 * coeff2[match_loop1];
        }

  //Get second sample (t = 0)
```

```
  match_base1[1][1] = 0;
  match_base2[1][1] = 0;
  signal1 = 0;
  signal2 =0;
  noisegen(sigma, bawgn , ran_num_gen);

  for (match_loop2 = 0; match_loop2 < 3; match_loop2++)
  {
        if (match_loop2 <= 1)
        noise_index = match_loop2;
        else
        {
          noisegen(sigma, bawgn , ran_num_gen);
          noise_index = match_loop2 - 2;
        }
        //signal1 = raisedcos(- match_loop2/samp_symb,period)*present_bit;
        signal1 = quantbits(((raisedcos(- match_loop2/samp_symb,period)*present_bit)
+bawgn[noise_index]),bits);
        //signal1 = raisedcos(- match_loop2/samp_symb,period)*present_bit
+bawgn[noise_index];
        match_base1[1][1] += signal1 * coeff1[match_loop2];
        match_base2[1][1] += signal1 * coeff2[match_loop2];
  }

  //Get third sample (t= 0.2)
  match_base1[1][2] = 0;
  match_base2[1][2] = 0;
  signal1 = 0;
  signal2 =0;
  noisegen(sigma, bawgn , ran_num_gen);

  for (match_loop3 = 0; match_loop3 < 4; match_loop3++)
  {
        if (match_loop3 <= 1)
        noise_index = match_loop3;
        else
        {
          noisegen(sigma, bawgn , ran_num_gen);
          noise_index = match_loop3 - 2;
        }
        //signal1 = raisedcos(1- match_loop3/samp_symb,period)*present_bit*present_bit;
        signal1 = quantbits(((raisedcos(1-
match_loop3/samp_symb,period)*present_bit*present_bit) +bawgn[noise_index]),bits);
        //signal1 = raisedcos(1- match_loop3/samp_symb,period)*present_bit*present_bit
+bawgn[noise_index];
        match_base1[1][2] += signal1 * coeff1[match_loop3];
        match_base2[1][2] += signal1 * coeff2[match_loop3];
  }

  //Get fourth sample (t=0.4)
  match_base1[1][3] = 0;
  match_base2[1][3] = 0;
  signal1 = 0;
  signal2 =0;
  noisegen(sigma, bawgn , ran_num_gen);
```

```
  for (match_loop4 = 0; match_loop4 < 5; match_loop4++)
  {
        if (match_loop4 <= 1)
        noise_index = match_loop4;
        else if ((match_loop4 > 1) && (match_loop4 <= 3))
        {
          noisegen(sigma, bawgn , ran_num_gen);
          noise_index = match_loop4 - 2;
        }
        else
        {
          noisegen(sigma, bawgn , ran_num_gen);
          noise_index = match_loop4 - 4;
        }
        //signal1 = raisedcos(2- match_loop4/samp_symb,period)*present_bit;
        signal1 = quantbits(((raisedcos(2- match_loop4/samp_symb,period)*present_bit) +
bawgn[noise_index]),bits);
        match_base1[1][3] += signal1 * coeff1[match_loop4];
        match_base2[1][3] += signal1 * coeff2[match_loop4];
  }
  mk = int((inter_loop*Ti) / Ts);
  frac_delay = (((inter_loop)*Ti) / Ts)- mk;
  numtime = (mk + frac_delay)*Ts;

  interpolate(match_base1,match_base2,numtime,Z);


  // matched filter
  if (Z[0] >= Z[1])
        signal = 1;
  else
        signal = -1;

  if (signal != present_bit)
        error++;

  BER = error/numsymb;

  }
  return (BER);
}
```

Figure A-2 The C++ program used for Simulations.

References

[1] Floyd M. Gardner, "Interpolation in Digital Modems-Part 1: Fundamentals," *IEEE Trans. Commun.*, vol. 41, pp. 501-507, Mar. 1993.

[2] L. Erup, F. M. Gardner, & R. A. Harris, "Interpolation in Digital Modems-Part 2: Implementation and Performance," *IEEE Trans. Commun.*, vol. 41, pp. 998-1008, June 1993.

[3] D. Kim, M. J. Narasimha, & D. C. Cox, "Design of Optimal Interpolation Filter for Symbol Timing Recovery", *IEEE Trans. Commun.*, vol. 45, pp. 877-884, July 1997.

[4] H. Meyr, M. Moeneclaey, & S. A. Fechtel (1998). *Digital Communication Receivers.* John Wiley & Sons, Inc.

[5] D. Hearne, & M. P. Baker (1997). *Computer Graphics (C Version), Second Edition.* Prentice-Hall International, Inc.

[6] W. Boehm, & H. Prautzsch (1991). *Numerical Methods.* A K Peters, Ltd.

[7] M. Berger (1986). *Computer Graphics with Pascal.* The Benjamin/Cummings Publishing Company, Inc.

[8] V. Ramakonar, D. Habibi, & A. Bouzerdoum (1998). *Performance Criteria for the Interoplation Controller fo a Digital Modem.* Edith Cowan University.

[9] V. Ramakonar, D. Habibi, & A. Bouzerdoum (1998). *Effects of Timing Error on the Bit Error Rate of a Fixed Clock Digital Receiver.* Edith Cowan University.

PART B: AN OVERVIEW OF THE VARIOUS MODULATION CLASSIFICATION SCHEMES AVAILABLE

Abstract:

Modulation recognizers that automatically report modulation types of received signals in general become to play an important role nowadays. Some modulation recognizers utilize analog modulations only, some utilize digital modulations only and some utilize both analog and digital modulations. The following sections give an overview for some of the most recently published algorithms for the modulation recognition.

1. Introduction:

An automatic modulation classifier is a system that automatically identifies the modulation type of the received radio signal given that the signal exists and its parameters lie in a known range. Modulation classification is a branch of non-cooperative theory that exploits several classical communication disciplines such as detection and estimation among others. It has recently attracted increasingly growing interest from both the military and commercial sectors due to its capability of placing several receivers in one universal receiver. This has practical utility for example in a network environment where it is required for an incoming signal to be routed to an appropriate processor. Modulation classification algorithms play an important role in some communication applications such as signal confirmation, interference identification, monitoring, spectrum management, surveillance, electronic warfare, military threat analysis and electronic counter-counter measure.

Generally, modulation classification can be approached either from a decision-theoretic or a statistical pattern recognition framework. In a decision theoretic framework, probabilistic and hypothesis-testing arguments are employed to formulate the classification problem. The resultant classifier is optimum in the sense that it minimizes

the average cost function. The complexity of this approach depends on the number of unknown parameters, associated with the received waveform and the classifier performance is expressed in terms of the probability of making a correct decision. In a statistical pattern recognition approach, the classification system is composed of two subsystems: the feature extraction and the pattern recognition subsystems. The role of the feature extraction subsystem is to extract useful information from the raw data and can be viewed as a mapping of the set of incoming signals into a chosen feature space. The function of the pattern recognizer subsystem is to indicate the membership of the modulation type class. In this paper, an overview of the different classification schemes using any or both of those approaches are explored.

2. Automatic Modulation Classification using Zero Crossing

A modulation recognizer using zero-crossing sampler as a signal conditioner is proposed by Hsue & Soliman [1] & [2], to classify constant-envelope signals of modulation types such as CW, MPSK and MFSK. This algorithm uses a combination of the two approaches mentioned in section (1). The phase difference and other related parameters are used as features. The density functions of these parameters are derived and then hypothesis testing is applied to formulate the classification rule. The zero-crossing sampling has wide applications and is an attractive tool in modern modulation recognition. The zero-crossing sampling records the sampling time, as the input signal crosses the zero voltage level. Its advantages include providing useful information related to the phase transitions of the received waveform and operating in a wide frequency dynamic range.

The classification strategy consists of two steps. The algorithm first separates single-tone (CW and MPSK) from multitone (MFSK) signals. Then it makes a final decision based on the number of states detected.

2.1 Problem Fomulation

The received waveform is modeled as:

$$r(t) = s(t) + v(t) \quad 0 \le t \le T_0 \qquad (2-1)$$

where $s(t)$ is a constant-envelope modulated signal with unknown modulation type. T_0 is the length of the observation interval, $v(t)$ is an additive band-limited Gaussian noise with zero mean and autocorrelation function $\psi(\tau)$. The signal $r(t)$ is sampled using the zero crossing sampler and a zero crossing sequence $\{x(i), i = 1, 2, \ldots, N\}$ is formed. Two other sequences $y(i)$ and $z(i)$ are necessary to extract the phase and frequency information from $\{x(i)\}$. The zero-crossing interval sequence {y(i)} is a measure of the instantaneous frequency ($y(i) = 1/2f_i$) and is defined as:

$$y(i) = x(i+1) - x(i) \quad i = 1, 2, \ldots, N-1 \quad (2-2)$$

The zero-crossing interval sequence $\{z(i)\}$ is a measure of the variation in $y(i)$ and is defined as:

$$z(i) = y(i+1) - y(i) \quad i = 1, 2, \ldots, N-2 \quad (2-3).$$

A waveform consisting of sinusoid and noise is represented by:

$$r(t) = A\cos 2\pi f_c t + v(t) \quad 0 \le t \le T_0 \qquad (2-4)$$

where A and f_c are the amplitude and frequency of the sinusoidal wave, respectively. The ith zero-crossing point can be written as:

$$x(i) = \frac{i - 0.5}{2f_c} + \alpha(i) \quad i = 1, 2, \ldots, N \qquad (2-5)$$

where $\alpha(i)$ are *independent and identically destributed* (IID) random variables that represent the variation due to $v(t)$ and any measurement errors. At high carrier-to-noise

ratio (CNR), the density function of $\alpha(i)$ is Gaussian, Rice S. cited in [2] Hsue et al., with zero mean variance:

$$\sigma_{\alpha}^{2} = \frac{1}{2(2\pi f_c)^2 \gamma} \qquad (2-6)$$

where γ is the CNR and is defined as $\gamma = A^2/(2\psi(0))$.

Similarly, the $y(i)$ is given by:

$$y(i) = \frac{1}{2f_c} + \varepsilon(i) \qquad (2-7)$$

where $\varepsilon(i) = \alpha(i+1) - \alpha(i)$ and its value depends on the phase of the carrier.

The conditional PDFs of the interval variations ε (i), conditioned on the knowledge of f_c can be approximated by the Gaussian density function with zero mean and variance as in [1]:

$$\sigma_{\varepsilon}^{2} = \frac{1}{(2\pi f_c)^2 \gamma}\left[1 + \rho\left(\frac{1}{2f_c}\right)\right] \qquad (2-8)$$

where $\rho(\tau) = \psi(\tau)/\psi(0)$ is the normalized autocorrelation function of the noise. Similarly, it can be shown that $z(i)$ is conditionally Gaussian with zero mean and variance [1]:

$$\sigma_{z}^{2} = \frac{1}{(2\pi f_c)^2 \gamma}\left[3 + 4\rho\left(\frac{1}{2f_c}\right) + \rho\left(\frac{1}{f_c}\right)\right] \qquad (2-9)$$

The signal parameters, f_c, γ and the symbol rate in the above equations, are assumed to be known. The information-bearing parameter is assumed to remain constant for almost all the symbol duration, except for short transitions between steady states that take place occasionally. This event is referred to as the intersymbol transition (IST) and it affects the accuracy in estimating the carrier frequency and symbol rate.

2.2 Discrimination between Single-Tone and Multitone Signals

Figure (2-1) shows a flow chart of a proposed algorithm used by Hsue el at. as in [1], to separate single-tone (CW and MPSK) from multitone (MFSK) signals.

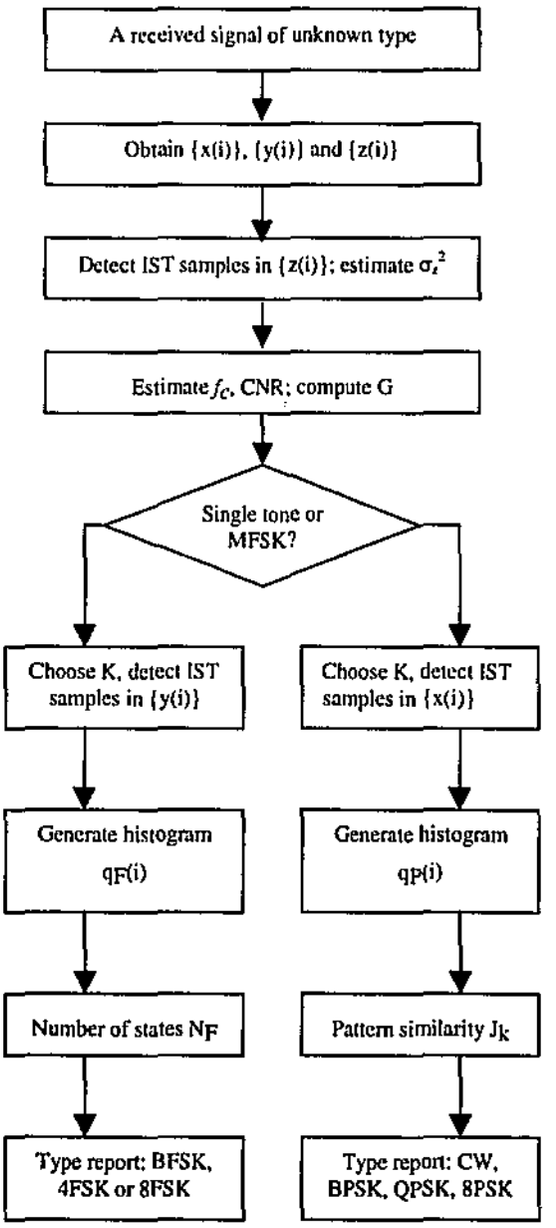

Figure 2.1 Functional flow chart for automatic classificationof digitally modulated signals.

As the received signal is sampled, the $\{x(i)\}$ sequence is formed and $\{y(i)\}$ and $\{z(i)\}$ are obtained. The $\{y_a(i)\}$ sequence is obtained by discarding those y(i) samples whose |z(I + 1) exceed 3.034 σ_{za} (a factor of 1/100 between samples). The length of the resultant $\{y_a(i)\}$ sequence is denoted by N_y. Signal parameters including carrier frequency, symbol rate, CNR and variances are estimated. Single-tone or multitone signals are then separated on the basis of the variance of $y_a(i)$. A phase histogram $q_P(i)$ is used for a single-tone signal to as a feature to identify CW and MPSK signals. A frequency histogram $q_F(i)$ is used to identify MFSK signals for a multitone signal.

2.3 Single Tone and FSK signals

The variance of $y_a(i)$ in the case of an MFSK signal can be written as in [1]:

$$\sigma_y^2(M) = \frac{1}{M}\sum_{m=1}^{M}\sigma_m^2(M) + \frac{1}{M}\sum_{m=1}^{M}\mu_m^2(M) - \frac{1}{M^2}\sum_{i=1}^{M}\sum_{k=1}^{M}\mu_i(m)\mu_k(M) \qquad (2-10)$$

where

$$\sigma_m^2(M) = \frac{1}{[2\pi f_c + (2m-1-M)\pi f_d]^2\gamma} \qquad (2-11)$$

$$\mu_m(M) = \frac{1}{2f_c + (2m-1-M)f_d} \qquad (2-12)$$

and f_d is the frequency deviation. Equation (2-12) can be used as a classification feature since it is monotonic increasing function in M for given values of f_d and f_c. Thus, separating single-tone signals from multitone signals can be reduced to discrimination between single-tone signals (M=1) and BFSK signals (M=2).

Equation (2-10) is only valid for infinite observation interval. In case of finite observation interval, the variance of $y_a(i)$ is estimated in [1]:

$$G = \frac{1}{N_s}\sum_{i=1}^{N_s}\left[y_a(i) - \frac{1}{N_s}\sum_{i=1}^{N_s} y_a(i) \right]^2 \qquad (2-13)$$

To discriminate between single-tone and BFSK signals, a hypothesis test is formulated based on the PDF of the variance G in a finite observation interval.

2.4 CW and PSK signals

For a reference continuous sine wave with zero initial phase, the nth zero-crossing point is $n/2f_c$. An unbiased estimate of the phase of the mth symbol ϕ_m, relative to the reference sine wave as in [1]:

$$\phi_m = 2\pi f_c \left[\frac{1}{L_m + 1} \sum_{n=k}^{k+L_m} \left(x(n) - \frac{n}{2f_c} \right) \right] \qquad (2-14)$$

where the identified beginning sample and end sample of the mth symbol is $x(k)$ and $x(k + L_m)$, respectively. This symbol phase estimate is meaningless since the initial phase of a received signal is not known. Symbol phase difference θ_m, the difference between the phases of the (m-1)th and mth symbols, is computed as:

$$\theta_m = \left[\phi_m - \phi_{m-1} + \frac{\pi}{8} \right] \bmod 2\pi \qquad (2-15)$$

A normalized phase difference histogram $q_P(i)$ generated by using the distribution of symbol phase differences θ_m is used as a feature for signal recognition.

2.5 FSK signal recognition

At high CNR, a zero crossing interval histogram $q_F(i)$ is obtained from the sample distribution of $\{y_a(i)\}$. The number N_F of hills in the histogram represents the number of states. Choosing the smallest integer D such that $N_F \leq 2^D$, a modulation type MFSK is reported with $M = 2^D$.

2.6 The Zero Crossing Classifier

Simulation results have shown that single-tone signals and multitone signals are separated successfully at a CNR of 15 dB or above. At lower CNR, the fluctuations in the variances yield an unsteady ratio between them, which causes difficulty in discriminating between single-tone and FSK signals.

The performance in discriminating between single-tone and MFSK signals, is dominated by the ratio, f_d/f_c. For f_d/f_c below certain value the receiver cannot distinguish between single-tone and FSK. The performance in discriminating between CW and MPSK, is dominated by the estimation accuracy of the carrier frequency. The computer simulation results demonstrated that the classification scheme provides successful modulation recognition for CNR $\geq$ 15 dB. A parallel processing scheme is recommended since the process of extracting the information from the three sequences $\{x(i)\}$, $\{y(i)\}$ and $\{z(i)\}$can be accomplished in parallel.

3. Detection and Classification using the Quasi-Log-Likelihood Ratio (qLLR) Rule

A classifier proposed by [3] Polydoros & Kim (1990) derived by approximating the likelihood-ratio functionals of phase-modulated digital signals in white Gaussian noise, hence named a quasi-log-likelihood ratio (qLLR) rule.

3.1 Detector Structures

Let the received waveform be described as:

$$r(t) = s(t) + n(t) \qquad 0 \le t \le T$$

where $n(t)$ is the standard AWGN with a two-sided power spectral density (PSD) of $N_0/2$ W/Hz. The signal part $s(t)$ described in [3] as:

$$\begin{aligned} s(t) &= s_c(t)\cos(\omega_c t + \theta_c) - s_s(t)\sin(\omega_c t + \theta_c) && (3-1a) \\ &= \sqrt{2S}a(t)\cos(\omega_c t + \theta(t) + \theta_c) && (3-1b) \\ &= \mathrm{Re}[\tilde{s}(t)e^{j(\omega_c t + \theta_c)}] && (3-1c) \end{aligned}$$

where $s_c(t)$ and $s_s(t)$ are the cosine and the sine coefficient functions respectively, $(2S)^{1/2}a(t)$ and $\theta(t)$ are the amplitude and phase processes of the polar representation with $a(t) \ge 0$. For constant-envelope modulation, $a(t) = 1$.

To derive detector structure for different modulations and to evaluate the associated performance, consider the standard noise complex envelope:

$$\tilde{n}(t) = \sqrt{2}[n_I(t) + jn_Q(t)]$$

where the white Gaussian noise processes $n_I(t)$, $n_Q(t)$ are independent, each with a two-sided PSD of $N_0/2$ W/Hz. The derivation of optimal LR (likelihood-ratio) detector can be performed for the synchronous or asynchronous, coherent or noncoherent cases as summarized below:

A. Coherent Case (θ_c known)

The LR $\Lambda(r(t))$ for QPSK in the synchronous coherent case as in [3]:

$$\Lambda(r(t)) = e^{-N\gamma_s}\prod_{n=1}^{N}\cosh\left(\frac{\sqrt{2S}}{N_0}r_{I,n}\right)\cosh\left(\frac{\sqrt{2S}}{N_0}r_{Q,n}\right) \qquad (3-2)$$

where $N=T/T_s$ is the number of observed symbols, $\gamma s = ST_s/N_0$ is the symbol or input SNR, and

$$r_{\begin{bmatrix} I \\ Q \end{bmatrix},n} = \int_{(n-1)T_s}^{nT_s} r(t) \begin{bmatrix} \cos \omega_c t \\ \sin \omega_c t \end{bmatrix} dt \qquad (3-3)$$

are the appropriate in-phase and quadrature matched-filter outputs during the nth symbol interval. If staggered observables are introduced:

$$r^{stag}_{\begin{bmatrix} I \\ Q \end{bmatrix},n} = \int_{(n-1/2)Ts}^{(n+1/2)Ts} r(t)\, p\left(t - nTs - \frac{Ts}{2}\right) \begin{bmatrix} \cos \omega_c t \\ \sin \omega_c t \end{bmatrix} dt \qquad (3-4)$$

then the theory can be extended to all the other modulations, with the resulting statistics, $Y = ln\ \Lambda$, to be used in the standard threshold comparison.

B. Bit-Noncoherent Case (Staggered Mod.)

The staggered modulation can be extended to the bit-noncoherent case [3]:

$$Y = -N\gamma_s + \sum_{k=1}^{2N} \ln I_0\left(\frac{\sqrt{2S}}{N_0} | \tilde{r}_k |\right) \qquad (3-5)$$

where $I_0(.)$ is the zeroth-order modified Bessel function, and $r_k = r_{I,k} + jr_{Q,k}$.

The bit noncoherent case is not realistic but can be useful for approximation the performance of any other detector and provide a lower bound on such performance.

C. Symbol-Noncoherent Case

The resulting detector statistics for this class are significantly more complicated in the case of staggered modulation. For the BPSK case [3]:

$$Y = -N\gamma_s + \sum_{n=1}^{N} \ln I_0\left(\frac{2\sqrt{S}}{N_0} | \tilde{r}_n |\right) \qquad (3-6)$$

D. Carrier-Noncoherent Case

The optimal structures for the carrier-noncoherent models are too complicated. With certain simplifying approximations, we can arrive at some quasi-optimal detector statistics, which for the BPSK case as in [3] is:

$$Y = \left(\Sigma_I + \Sigma_Q\right) + \sqrt{\left(\Sigma_I - \Sigma_Q\right)^2 + \left(2\Sigma_{IQ}\right)^2} \qquad (3-7)$$

where

$$\Sigma_a = \sum_{n=1}^{N} r_{a,n}^2, \qquad \Sigma_a^{stag} = \sum_{n=1}^{N} \left(r_{a,n}^{stag}\right)^2; \quad a = I, Q$$

$$\Sigma_{IQ} = \sum_{n=1}^{N} r_{I,n} r_{Q,n}, \quad \Sigma_{I^{st},Q^{st}} = \sum_{n=1}^{N} \left(r_{I,n} r_{I,n}^{stag} + r_{Q,n} r_{Q,n}^{stag}\right)$$

$$S_{a,a^{st}} = \Sigma_a + \Sigma_a^{stag}; \qquad a = I, Q$$

$$S_{I,Q^{st}} = \Sigma_I + \Sigma_Q^{stag}, \quad S_{I^{st},Q} = \Sigma_I^{stag} + \Sigma_Q.$$

For staggered modulation, the cross product term $\Sigma_{Ist,\,Qst}$ is negligible.

3.2 A qLLR Decision-Theoretic Classifier

In this section, a novel quasi-optimal classifier is derived. The carrier frequency and symbol timing are assumed to be known to the receiver. Let the complex envelope of the matched-filter output at the end of the nth symbol interval be:

$$\tilde{r}_n = r_{I,n} + jr_{Q,n}$$

The LR for an MPSK signal with known carrier frequency and unknown carrier phase can be written [3] as:

$$\Lambda(r(t)) = e^{-N_T S} E_{\alpha} \left\{ \exp \sum_{n=1}^{N} \ln E_{\theta_n} \left\{ \exp\left[\frac{2\sqrt{S}}{N_0} \mathrm{Re}\left[\tilde{r}_n e^{-j(\alpha+\theta_n)} \right] \right] \right\} \right\} \qquad (3-8)$$

where E_q denotes expectation with respect to the random variable q, and r_n is the aforementioned complex envelope at the nth symbol interval.

To distinguish between the $M = 2$ hyppthesis (BPSK) versus any other $M = 2^m \geq 4$, then the LR [3] is:

$$\left. \frac{\Lambda(r(t)/BPSK)}{\Lambda(r(t)/MPSK)} \right|_{M \geq 4} \approx I_0\left[\frac{S}{N_0^2} \sqrt{\left(\Sigma_I - \Sigma_Q\right)^2 + 4\Sigma_{IQ}^2} \right] \qquad (3-9)$$

And the statistic qLLR can be utilized as a classification module between, say, BPSK and QPSK, where qLLR [3]:

$$qLLR = \left(\Sigma_I - \Sigma_Q\right)^2 + 4\Sigma_{IQ}^2 \qquad (3-10)$$

The phase-based rule is actually an approximate version of the *qLLR*.

3.3 qLLR Classifier Performance

The standard measure of performance in this problem is the probability of correct classification Pc. Assuming that the prior probability of occurrence of BPSK and QPSK is the same, then for a BPSK/QPSK classifier, we have [3]:

$$P_c = \Pr[\text{correct class.}] = \frac{\Pr[2/2] + \Pr[4/4]}{2} \quad (3-11)$$

where Pr [*M/M*]; M = 2, 4 is the probability that the classifier will declare MPSK modulation given that this indeed is the correct modulation embedded in noise. Typically Pr[2/2] > Pr [4/4].

Since the *qLLR* is not a function of the carrier phase θ_c and the rotational phase θ_0, but only a function of phase differences, we can assume that $\theta_0 = \theta_c = 0$. Let $G[x; m_x, \sigma_x^2]$ be the one-dimensional Gaussian density function with mean m_x and variance σ_x^2, then the Gaussian-mixture probability density function (pdf) [3]:

$$G_{mix}[x; m_x, \sigma_x^2] = \frac{G[x; m_x, \sigma_x^2] + G[x; -m_x, \sigma_x^2]}{2} \quad (3-12)$$

For M = 2, 4, the Gaussian random variables $r_{I,n}$ and $r_{Q,n}$ are uncorrelated, and their joint density function as in [3] is:

$$p(r_{I,n}, r_{Q,n}) = \begin{cases} G_{mix}[r_{I,n}; \sqrt{2}m_I, \sigma^2] G[r_{Q,n}; 0, \sigma^2] \\ \qquad (BPSK) \\ G_{mix}[r_{I,n}; m_I, \sigma^2] G_{mix}[r_{Q,n}; m_Q, \sigma^2] \\ \qquad (QPSK) \end{cases} \quad (3-13)$$

where

$$m_I = m_Q = T_s\sqrt{S/2}; \sigma^2 = \frac{N_0}{2T_s}$$

The pdf expression for $q = qLLR$ as in [3]:

$$f_{BPSK}(q) = \frac{1}{2V}\exp\frac{\lambda+q}{-2V} I_0\left(\frac{\sqrt{q\lambda}}{V}\right); q \geq 0 \quad (3-14a)$$

where

$$\lambda = N^2; V = \frac{4N}{\gamma_s}\left(1+\frac{1}{\gamma_s}\right) \quad (3-14b)$$

and for QPSK

$$f_{QPSK}(q) = \frac{1}{\sqrt{ab}}\exp\left[-\left(\frac{1}{a}+\frac{1}{b}\right)\frac{q}{2}\right] I_0\left[\left(\frac{1}{b}-\frac{1}{a}\right)\frac{q}{2}\right];$$
$$q \geq 0 \quad (3-15a)$$

where

$$a = 2N\left(1+\frac{1}{\gamma_s}\right)^2; b = a - N. \quad (3-15b)$$

For small input SNR ($\gamma_s << 1$), (3-15a) can be approximated as in [3] by:

$$f_{QPSK}(q) \approx \frac{1}{a}\exp\left(\frac{-q}{a}\right); \quad q \geq 0 \quad (3-16)$$

It is possible to predict performance analytically from the density functions derived.

3.4 The qLLR Approach

The LR approach is conceptually straightforward and systematic, it provides insight into the problem, and more significantly, it suggests structures that operate directly on the sampled outputs of the quadrature matched filters. It is of theoretical as well as practical significance. It assumes a perfect knowledge of the symbol pulse shape at the receiver's front end so that perfect matched filtering and sampling can be performed. So pulse shapes other than rectangular can also be accommodated. LR methods are well equipped to address MPSK classification for $M \geq 4$, whereas spectral-correlation methods are not.

4. Signal Classification using Statistical Moments

In this section, a moments-based algorithm to classify general M-ary PSK signals ($M=2^{\alpha}$, $\alpha=0, 1, 2 \ldots$) is introduced. Soliman & Hsue developed this algorithm in [4].

4.1 Developement

The signal component of the received signal *s(t)* is assumed to be either CW or *M*PSK, and it contains the phase information. The noise component is an additive white Gaussian noise, *w(t)*, with zero-mean and power spectral density. The phase information can be extracted and represented as in [4]:

$$\phi_a(i) = \theta_a(i) + v(i) \quad -\infty < i < \infty. \quad -\pi < \phi_a(i) \le \pi \qquad (4-1)$$

where $\theta_a(i)$ is the sampled phase component of the transmitted MPSK signal *s(t)* and *v(t)* is the random phase attributed to the received noise *w(t)* and any other measurement errors. Also $\{\phi_a(i)\}$ and $\{\theta_a(i)\}$ are assumed to be independent and identically destributed with zero-means.

4.1.1 Probability Density Functions

The probability density function of the phase ϕ_0 for a CW signal of amplitude A, in which $\alpha = 0$, is given by Bennett (1956), cited in [4] is:

$$f_{\phi}(\phi_0) = \frac{\ell^{-\gamma}}{2\pi} + \sqrt{\frac{\gamma}{\pi}}\cos(\phi_0)\ell^{-\gamma\sin^2(\phi_0)} \cdot Q\left[-\cos(\phi_0)/\sqrt{2\gamma}\right] \quad -\pi < \phi_0 \le \pi \qquad (4-2)$$

where

$$Q[x] = \frac{1}{2\pi}\int_{x}^{\infty} \ell^{-y^2/2} dy \qquad (4-3)$$

$$\gamma = \frac{A^2}{2\sigma_w^2} \qquad (4-4)$$

Assuming that $\sigma_w 2 \rightarrow 0$, equation (4-2) above can be approximated by Tikhonov probability density function according to Leib & Pasupathy (1988) cited in [4] as:

$$f_\phi(\phi_0) \approx \frac{\exp[2\gamma\cos(\phi_0)]}{2\pi I_0[2\gamma]} \quad -\pi < \phi_0 \leq \pi \qquad (4-5)$$

where $I_0[.]$ is the zero-order modified Bessel function of the first kind. This approximation is good for $\gamma > 6$ dB and fair for values of γ around 0 dB. For values above 6 dB, then equation (4-2) can be approximated by a Gaussian pdf as in [4]:

$$f_\phi(\phi_0) \approx N\left(0, 1/\sqrt{2\gamma}\right) \qquad (4-6)$$

This approximation is quite accurate and simple and thus would be used here. Then for equally likely M phases, the pdf of θ_α as in [4]:

$$f_\theta(y{:}\alpha) = \frac{1}{2^\alpha}\sum_{k=1}^{2^\alpha} \delta\left(y - \eta_k(\alpha)\right) \qquad (4-7)$$

where $\eta_k(\alpha)$ is the phase of kth phase states and can be expressed as [4]:

$$\eta_k(\alpha) = \frac{\left(2k - 2^\alpha - 1\right)}{2^\alpha} \quad k = 1, 2, \ldots, 2^\alpha \qquad \alpha = 0, \ldots, \log_2 M. \qquad (4-8)$$

The pdf of the ϕ_α is then as in [4]:

$$f_\phi(y;\alpha) = \frac{1}{2^\alpha}\sum_{k=1}^{2^\alpha} f_Y(y-\eta(\alpha))$$
$$= \frac{1}{2^\alpha}\sum_{k=1}^{2^\alpha} \frac{\exp\left[2\gamma\cos\left(y-\eta_k(\alpha)\right)\right]}{2\pi I_0[2\gamma]} \quad -\pi \le y < \pi. \qquad (4-9)$$

If pdf is plotted against the phase angle, there would be some peaks in the graph indicating the number of signal phase states. Those peaks are related to the CNR, decreasing CNR causes the peaks to smear off until finally the pdf reaches that of a uniformly distributed random variable, and $f_\phi(y;\alpha)$ approaches $1/2\pi$ as CNR goes to -∞ dB or α goes to ∞.

4.1.2 Ensemble Moments

Let $I_k[.]$ be the modified Bessel function of the first kind and order k, then the nth moment of ϕ_α, in the presence of noise can be expressed as [4]:

$$m_n(\alpha) = \begin{cases} \dfrac{\pi^n}{n+1} + \dfrac{1}{2^\alpha \pi I_0[2\gamma]}\displaystyle\sum_{i=1}^{2^\alpha}\sum_{k=1}^{\infty} I_k[2\gamma]\int_{-\pi}^{\pi} y^n \cos\left(k\left(y-\eta_i(\alpha)\right)\right)dy, & n \text{ even} \\ 0, & n \text{ odd} \end{cases} \qquad (4-10)$$

Even moments of ϕ_α, is a monotonic increasing function with respect to α according to [4]. They can be used as a feature to classify CW and general *M*PSK signals. Signals with large M may require higher order moments to discriminate between.

4.1.3 Measured Moments

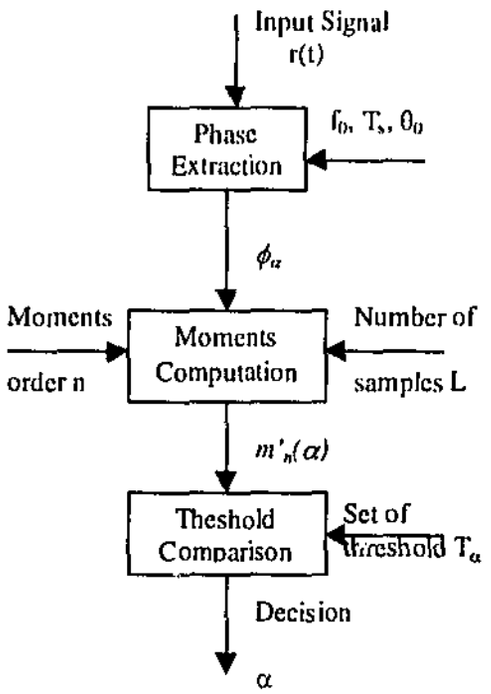

Figure 4-1 Block diagram of the MPSK classifier using moments.

The unbiased sample averages of even powers of the extracted phase is given my [4] as:

$$\hat{m}_n(\alpha) = \frac{1}{L}\sum_{i=1}^{L} \phi_\alpha^n(i) \qquad (4-11)$$

where L is the number of observed samples in a finite interval.

Let the pdf of $m_n(\alpha)$ be a $N(\mu_n(\alpha), \sigma_n(\alpha))$ where as in [4]:

$$\mu_n(\alpha) = E\{\hat{m}_n(\alpha)\} = \frac{1}{L}\sum_{i=1}^{L} E\{\phi_\alpha^n(i)\} = m_n(\alpha) \qquad (4-12)$$

and

$$\sigma_n^2(\alpha) = \frac{1}{L^2}\sum_{i=1}^{L} E\left\{\left[\sigma_a^n(i) - E\left\{\sigma_a^n(i)\right\}\right]^2\right\} = \frac{m_{2n}(\alpha) - m_n^2(\alpha)}{L} \qquad (4-13)$$

This suggests that since the random variables in the pdf have means that are monotonic increasing functions of α, then they can be used as a discriminating feature to classify MPSK signals. Figure 4-1 shows a block diagram of the MPSK classifier using moments, [4].

4.2 Statistical Moments Classifier Performance

Based on the densities of the phase moments, a hypothesis testing problem may be formulated so as to evaluate the performance of the statistical moments classifier performance. This hypothesis testing problem can be formulated [4] as:

$$\begin{aligned} &H_0: \text{pdf of } \hat{m}_n(0) = N\left(\mu_n(0), \sigma_n(0)\right) \\ &H_1: \text{pdf of } \hat{m}_n(1) = N\left(\mu_n(1), \sigma_n(1)\right) \\ &\vdots \\ &H_\alpha: \text{pdf of } \hat{m}_n(\alpha) = N\left(\mu_n(\alpha), \sigma_n(\alpha)\right) \end{aligned} \qquad (4-14)$$

A simple test can be made as [4]:

$$\begin{aligned} &\hat{m}_n(\alpha) < T1 && \text{CW signaling.} \\ &T_\alpha < \hat{m}_n(\alpha) < T_{\alpha+1} && M\text{PSK signal,} \quad M = 2^\alpha \end{aligned} \qquad (4-15)$$

Tα is given in equation (4-16) as in [4].

$$T_\alpha = \frac{\mu_n(\alpha-1)\sigma_n^2(\alpha)-\mu_n(\alpha)\sigma_n^2(\alpha-1)}{\sigma_n^2(\alpha)-\sigma_n^2(\alpha-1)} + \frac{\sigma_n^2(\alpha-1)\sigma_n(\alpha)\left[\left(\mu_n(\alpha)-\mu_n(\alpha-1)\right)^2+\left(\sigma_n^2(\alpha)-\sigma_n^2(\alpha-1)\right)\ln\left(\sigma_n^2(\alpha)/\sigma_n^2(\alpha-1)\right)\right]^{1/2}}{\sigma_n^2(\alpha)-\sigma_n^2(\alpha-1)} \quad (4-16)$$

The quality of the classifier can be measured using the nth moment, in terms of two probabilities of misclassification as in [4]:

$$\begin{aligned} \mathrm{Pe}(n,0) &= \mathrm{Prob}\{\text{misclassification}|H_0\} \\ &= \int_{T1}^{\infty} N\left(\mu_n(0),\sigma_n(0)\right)dy = Q\left[\frac{T_1-\mu_n(0)}{\sigma_n(0)}\right] \end{aligned} \quad (4-17)$$

and

$$\begin{aligned} \mathrm{Pe}(n,\alpha) &= \mathrm{Prob}\{\text{misclassification}|H_\alpha\} \\ &= \int_{-\infty}^{T_\alpha} N\left(\mu_n(\alpha),\sigma_n(\alpha)\right)dy + \int_{T_{\alpha+1}}^{\infty} N\left(\mu_n(\alpha),\sigma_n(\alpha)\right)dy \\ &= Q\left[\frac{\mu_n(\alpha)-T_\alpha}{\sigma_n(\alpha)}\right] + Q\left[\frac{T_{\alpha+1}-\mu_n(\alpha)}{\sigma_n(\alpha)}\right]. \end{aligned} \quad (4-18)$$

For each value of L, there is an optimum value of n. Increasing n would result in increasing the variance and the probability of classification.

4.3 The Statistical Moments Classifier

The statistical moments classifier has been developed to classify CW and MPSK signal with $M = 2^\alpha$. The nth moment is a monotonic increasing function of α, and there is an optimum value of n for the best performance. The performance of this classifier is not

better than that of qLLR, but the qLLR is limited to in a way that it is only valid for low CNR less than zero, and can discriminate between BPSK and QPSK signals only. The statistical moments classifier is better than the Square Law Classifier (SLC) for positive values of CNR, and also better than Phase-Based Classifier (PBC) for all values of CNR.

5. Classification of Signals in Unknown ISI Environments using the Average and Generalized Likelihood Ratio Test

Lay & Polydoros have developed two classification techniques for digitally modulated signals affected by intersymbol interference (ISI) in [5]. In an ISI environment, the received signal is degraded by intersymbol interference because of a finite impulse response (FIR) channel and AWGN. The carrier phase and channel coefficients are assumed to be known or can be obtained. For a transmitted sequence a, the structure of the observed receive sequence is then as in [5]:

$$r_k = \sum_{n=0}^{L-1} h_n . a_{k-n} + n_k \qquad (5-1)$$

For a known channel impulse response, two different likelihood based tests are developed, the Average Likelihood Ratio Test (ALRT) and the Generalized Likelihood Ratio Test (GLRT).

5.1 Average Likelihood Ratio Test

The ISI channel is assumed to be known. An N symbol long sequence is received from one of two constellations, C_0 and C_1, whose occurrences are assumed to be equiprobable. The a *priori* probabilities are given by the inverse of the constellation size. By averaging over unknown parameters, the average *posteriori* densities should be found so as to form a likelihood ratio test as in [5]:

$$\begin{aligned} pr(r|C_i) &= \int_a pr(r|a, C_i) p(a|C_i) da \\ &= \left\langle pr(r|a, C_i) p(a|C_i) \right\rangle_{a \in C_i} \end{aligned} \qquad (5-2)$$

where,

$\langle\ \rangle_{a\in C_i}$ = *average over all data sequences in* C_i.

The standard maximum a posteriori decision rule is given by [5] as:

$$\Lambda(r)=\frac{pr(r|C_0)}{pr(r|C_1)}\begin{vmatrix} H_0 \\ > \\ < \\ H_1 \end{vmatrix} \eta \equiv \frac{P(C_1)}{P(C_0)} \qquad (5-3)$$

The joint density of a received sequence which has been corrupted by an ISI channel of length L [5]:

$$\begin{aligned} pr(r|a) &= p(r_N, r_{N-1},\ldots,r_1|a_N,\ldots,a_1) \\ &= \prod_{k=1}^{N} p(r_k|a_k,\ldots,a_{k-L+1}) \qquad (5-4) \\ where,\ a_k &= 0,\ k \le 0. \end{aligned}$$

In order to compute the average likelihood, the likelihood function for the $|C_i|^N$ data sequences for each constellation must be calculated. Then it can be calculated as in [5]:

$$pr(r|C_i)=\frac{1}{|C_i|^N}\sum_{j=1}^{|C_i|^N}\prod_{k=1}^{N} p(r_k|a_{j,k},\ldots,a_{j,k-L+1}) \qquad (5-5)$$

$$where,\ p(r_k|a_{j,k},\ldots,a_{j,k-L+1})=\left(\frac{1}{\pi N_0}\right)^{\frac{1}{2}}.e^{-\frac{1}{N_0}\left[r_k-\sum_{l=0}^{L-1}h_l\cdot a_{j,k-l}\right]^2}$$

$$and,\ a_j=[a_{j,1},\ldots,a_{j,N}] \equiv one\ of\ |C_i|^N\ sequences,\ a_{j,k}\in C_i$$

5.2 Generalized Likelihood Ratio Test

A generalized likelihood ratio test (GLRT) is given by [5] as:

$$\Lambda_g(r) = \frac{pr\left(r|\hat{a}_{ML(C_0)}, C_0\right)}{pr\left(r|\hat{a}_{ML(C_1)}, C_1\right)} \begin{matrix} C_0 \\ > \\ < \\ C_1 \end{matrix} \eta \qquad (5-6)$$

$$where,\ pr\left(r|\hat{a}_{ML(C_i)}, C_i\right) = \max_{a \in C_i} pr(r|a, C_i) \quad and,\ \eta \equiv threshold$$

$$\hat{a}_{ML(C_i)} \equiv Max.\ Likelihood\ Data\ Sequence\ Estimate\ of\ ontellation\ C_i$$

To calculate the test statistic, the maximum likelihood estimate of the data sequence is required. This estimate can be obtained using the Viterbi algorithm for known channels. For two modulations, C_0 and C_1, and an L length channel, the GLRT is given by [5] as:

$$\Lambda_g(r) = \frac{\prod_{k=1}^{N} e^{-\frac{1}{N_0}\left[r_k - \sum_{l=0}^{L-1} h_l . \hat{a}_{ML(C_0),k-l}\right]^2}}{\prod_{k=1}^{N} e^{-\frac{1}{N_0}\left[r_k - \sum_{l=0}^{L-1} h_l . \hat{a}_{ML(C_1),k-l}\right]^2}} \begin{matrix} C_0 \\ > \\ < \\ C_1 \end{matrix} \eta \qquad (5-7)$$

The simplified log generalized likelihood ratio appears as given by [5]:

$$\log\left(\Lambda_g(r)\right) = -\sum_{k=1}^{N}\left[r_k - \sum_{l=0}^{L-1} h_l . \hat{a}_{ML(C_0),k-l}\right]^2 + \sum_{k=1}^{N}\left[r_k - \sum_{l=0}^{L-1} h_l . \hat{a}_{ML(C_1),k-l}\right]^2 \qquad (5-8)$$

The performance of the GLRT is suboptimal compared to ALRT, but it has some implementation advantages over it. Computational and numerical precision problems are avoided and the knowledge of the noise variance is not required.

5.3 ALRT and GLRT

Two classification techniques for digitally modulated signals affected by intersymbol interference have been developed, the Average Likelihood Ratio Test (ALRT) and the Generalized Likelihood Ratio Test (GLRT). The initial development of the classification tests is derived assuming a known-channel impulse response. In an unknown ISI environment, per-survivor processing (PSP) is employed. " *PSP is a technique for estimating both the data sequence and the unknown parameter of a communications signal which exhibits memory*" [5]. Both likelihood based classification tests indicate substantial potential in discriminating between three different 16-ary symbol contellations. The ALRT produces better performance than GLRT. ALRT requires knowledge of the signal power and the noise variance of the channel. Through the use of the Viterbi algorithm in GLRT, the decision statistics computation is reduced.

6. Digital Quadrature and Offset Modulation Classification by LF and Mth-law Classifiers

Hwang & Polydoros developed two approaches for the classification of QAM and OQPSK signals in [6], the likelihood functionals (LF) and the M^{th}-law classifiers. Optimal solutions to a variety of classification problems can be provided by the likelihood-ratio tests. The M^{th}-law, M = 2, 4, 8,..., is based on the signal's properties to classify between MPSK and M'PSK, where M' > M, by detecting the presence or absence of a spectral line around the M^{th} multiple of carrier frequency. Some assumptions are made as follows:

- An AWGN channel with a two sided PSD of $(N_0/2)$ W/Hz.
- Symbol timing is known.
- The carrier frequency is known.
- An observation time of NT_s seconds (T_s is the sample time).

6.1 Likelihood Approaches

A two dimensional modulation scheme whose constellation S'_k can be any set on the complex plane, is represented in [6] as:

$$s(t) = \Re\left\{\sqrt{2\alpha S}\sum_{k=1}^{N}\hat{S}_k u_{T_s}(t - kT_s)e^{j(2\pi f_c + \theta_c)}\right\} \qquad (6-1)$$

The power of the signal is adjusted to S by a normalizing factor [6]:

$$\alpha = \left[\frac{1}{Q}\sum_{n=1}^{Q}\left|\hat{S}_i\right|^2\right]^{-1}.$$

The carrier phase θ_c, is modeled as a r.v. uniformly distributed in $[0, 2\pi]$, and $u_{Ts}(t)$ is the standard unit pulse of duration T_s seconds.

The log-likelihhod functional (LLF) of the observation r(t) with respect to the signal st) in AWGN is given in [6] as:

$$l(r(t)) = \ln << \exp\left\{\frac{2}{N_0}\int_0^{NT_s} r(t).s(t)dt\right\} >_{\hat{s}_k} >_{\theta_c} \qquad (6-2)$$

where, $<>_{\hat{s}_k}$: averaging w.r.t signal constellation

$<>_{\theta_c}$:averaging w.r.t carrier phase.

For specific cases, the following results can be obtained [6]:

6.1.1 MPSK

An approximate version of the LLF for MPSK is given as [6]:

$$l_{MPSK} \cong \sum_{p=1}^{M/2} \frac{1}{(p!)^2}\left(\frac{\sqrt{S}}{N_0}\right)^{2p} \sum_{k=1}^{N} \left|\tilde{r}_k\right|^{2p} + \frac{2}{M!}\left(\frac{\sqrt{S}}{N_0}\right)^{M} \left|\sum_{k=1}^{N} (\tilde{r}_k)^M\right| \qquad (6-3)$$

where, $\tilde{r}_k = r_{I,k} + jr_{Q,k} = \int_{(k-1)Ts}^{kTs} r(t).\sqrt{2}e^{-j2\pi f_c t}dt.$

MPSK and M'PSK can be distinguished by the statistic as in [6]:

$$q_M = \left|\sum_{k=1}^{N} (\tilde{r}_k)^M\right| = \left[\left(\sum_{k=1}^{N} A_{M,k}\right)^2 + \left(\sum_{k=1}^{N} B_{M,k}\right)^2\right]^{1/2} \qquad (6-4)$$

A BPSK signal can be distinguished from MPSK signals (M ≥ 4) with q_2. A recursive algorithm for calculating the statistic q_M is [6]:

$$\begin{aligned} A_{M,k} &= A_{M/2,k}^2 - B_{M/2,k}^2; A_{1,k} = r_{I,k} \\ B_{M,k} &= 2A_{M/2,k}.B_{M/2,k}; B_{1,k} = r_{Q,k} \end{aligned} \qquad (6-5)$$

For large N, the arguments of equation (6-4), can be approximated by jointly Gaussian random variables. For small SNR ($\gamma s << 1$), then the pdf for $q = q_M^2.(T_s^2 S)^{-M/2}$ is given in [6]:

$$f_{q_M}^{MPSK}(q) = \frac{1}{2NV_M}\exp\left(\frac{N^2+q}{-2NV_M}\right)I_0\left(\frac{\sqrt{q}}{V_M}\right) \qquad (6-6)$$

$$f_{q_M}^{M'PSK} \approx \frac{1}{2NV_M}\exp\left(\frac{-q}{-2NV_M}\right);\ q \geq 0$$

$$\text{where}\quad V_M = \sum_{l=1}^{M}\frac{(M!)^2}{2l![(M-l)!]^2}\cdot\gamma_s^{-l}$$

These pdfs can be used to express the probability of correct classification.

6.1.2 QAM

An approximate version of the LLF for QAM is given as [6]:

$$l_{QAM} \cong \sum_{p=1}^{\infty}\left\{\left(\frac{\sqrt{\alpha S}}{N_0}\right)^p \sum_{l=0}^{\lfloor p/2\rfloor}\left[\frac{\varepsilon_{p-2l}\left|\overline{\left|\tilde{S}_i\right|^{2l}\left(\tilde{S}_i\right)^{p-2l}}\right|}{l!(p-l)!}\left|\sum_{k=1}^{N}\left|\tilde{r}_k\right|^{2l}\left(\tilde{r}_k\right)^{p-2l}\right|\right]\right\} \qquad (6-7)$$

$$\text{where,}\ \overline{\left|\tilde{S}_i\right|^{2l}\left(\tilde{S}_i\right)^{p-2l}} = \frac{1}{Q}\sum_{i=1}^{Q}\left|\tilde{S}_i\right|^{2l}\left(\tilde{S}_i\right)^{p-2l}.$$

The recursive algorithm (6-5), is generalized when M is odd as [6]:

$$\begin{aligned} A_{M,k} &= A_{M-1,k}r_{I,k} - B_{M-1,k}r_{Q,k} \\ B_{M,k} &= A_{M-1,k}r_{Q,k} + B_{M-1,k}r_{I,k} \end{aligned} \qquad (6-8)$$

The pdf expressions (6-6), can also be generalized for all *QAM* signals, and the correct probability of classifying QAM_i and QAM_j by the q_M rule is [6]:

$$P_{q_M,c}^{QAM} = \frac{1}{2}\left[1 - Q(\xi_i,\zeta_i) + Q(\xi_j,\zeta_j)\right] \tag{6-9}$$

$$\xi_\eta = \sqrt{\frac{a_{M,\eta}^2 N}{V_{M,\eta}}},\ \zeta_\eta = \sqrt{\frac{T_0}{NV_{M,\eta}}};\ a_{M,\eta} = \frac{\left|\overline{\left(\tilde{S}_\eta\right)^M}\right|}{\left(\overline{\left|\tilde{S}_\eta\right|^2}\right)^{M/2}};\qquad \eta = i, j$$

$$V_{M,\eta} = \sum_{l=0}^{M} \frac{(M!)^2}{2l!\left[(M-l)!\right]^2} \cdot \frac{\overline{\left|\tilde{S}_\eta\right|^{2(M-l)}}}{\left(\overline{\left|\tilde{S}_\eta\right|^2}\right)^{M-l}} \cdot \gamma_s^{-l} - \frac{a_{M,\eta}^2}{2}$$

6.1.3 OQPSK

Similarly, an approximate version of the LLF of OQPSK given by [6] is:

$$l_{QPSK} \cong \sum_{p=1}^{\infty}\left\{\left(\frac{\sqrt{S}}{N_0}\right)^{2p} \sum_{m=0}^{p}\left[\frac{\varepsilon_m}{(p-m)!(p+m)!}\cdot\prod_{k=1}^{N}\left(\left|\tilde{r}_k\right|^{2(p-m)}\left(\tilde{r}_k\right)^{2m} + (-1)^m\left|\tilde{r}_{stag,k}\right|^{2(p-m)}\left(\tilde{r}_{stag,k}\right)^{2m}\right)\right]\right\}$$
$$= \frac{S}{N_0^2}\left\{\sum_{k=1}^{N}\left[\left|\tilde{r}_k\right|^2 + \left|\tilde{r}_{stag,k}\right|^2\right] + \left|\sum_{k=1}^{N}\left(\tilde{r}_k\right)^2 - \left(\tilde{r}_{stag,k}\right)^2\right|\right\} + \ldots \tag{6-10}$$

where, $\tilde{r}_{stag,k} = \int_{(k-1/2)Ts}^{(k+1/2)Ts} r(t).\sqrt{2}e^{-j2\pi fct}dt.$

For classifying *OQPSK* versus *QPSK*, the statistic is used as [6]:

$$q_{stag} = \left|\sum_{k=1}^{N}\left(\tilde{r}_k\right)^2 - \left(\tilde{r}_{stag,k}\right)^2\right| \tag{6-11}$$

And the correct probability of classifying *OQPSK* and *QPSK* by the q_{stag} rule [6] is:

$$P_{q_{stag},C}^{OQPSK} = \frac{1}{2}\left[1-\exp\left(-\frac{T_0}{2NV_Q}\right)+Q\left(\sqrt{\frac{N}{4V_0}},\sqrt{\frac{T_0}{NV_0}}\right)\right] \qquad (6-12)$$

where, $V_0 = \frac{1}{16}+\gamma_s^{-1}+\gamma_s^{-2}, V_Q = \frac{1}{4}+\gamma_s^{-1}+\gamma_s^{-2}$

and $T_0 \approx 4V_s^2\left(I_0^{-1}\left[\exp\left(\frac{N}{4(V_O+V_Q)}\right)\right]\right)^2$.

6.2 M[th]-law Approaches

According to Hwang& Polydoros (1991), a sufficient and necessary condition for generating a spectral line around Mf_c after an M^{th}-law transform for a QAM signal is that the expectation $(Si)^M$ is not zero, which becomes part of the coefficient of the q_M term in the LLF of *QAM*. This result is summarized in the following lemma [6]:

"When applying a QAM signal to an M[th]-law device, a spectral line is present around the Mf_c-band iff the LLF of this QAM signal contains the q_M term. "

To classify between two *QAM* signals, more than one M can be found, by choosing the smallest value of M so as to reduce the variance of the q_M statistic.

The correct classification probability is [6]:

$$V_{M,\eta} = \alpha_\eta^M\left\{\frac{\overline{\left|\tilde{S}_\eta\right|^{2M}}-\left|\overline{\left(\tilde{S}_\eta\right)^M}\right|^2}{2}+\sum_{k=1}^{M}\frac{2^{k-2}(M!)^2\left(0.9038\alpha_\eta\gamma_s\right)^{-k}U_k}{k![(M-k)!]^2}\right\},\ \eta=i,j \qquad (6-13)$$

where,

$$U_k = \left(\overline{\left| \tilde{S}_\eta \right|^{2(M-k)}} - \left| \overline{\left(\tilde{S}_\eta \right)^{M-k}} \right|^2 \right) P_k + \left| \overline{\left(\tilde{S}_\eta \right)^{M-k}} \right|^2 S_k$$

$$P_k = 2\int_{-1}^{1} \left[\frac{\sin(2\pi y)}{2\pi y} \right]^k (1-|y|)dy;\ S_k = \frac{1}{\pi}\int_{-\infty}^{\infty} \left(\frac{\sin x}{x} \right)^k dx.$$

6.3 q_M-type and M^{th}-law Classifiers

Both methods share some similarities in terms of implementation and performance. The q_M classifier could be interpreted as a synchronous, pulse-shape matched-filter, M^{th}-law classifier.

The correct classification probability can be approximated by the Gaussian-tail integral (Q-finction) in a low SNR environment as [6]:

$$P_c \approx 1 - Q\left(\frac{\left| a_{M,1} - a_{M,2} \right|}{\sqrt{bM!}} \cdot \sqrt{N(c\gamma_s)^M} \right) \qquad (6-14)$$

where, (b, c) = (2, 1) for q_M and (b, c) = (S_M, 0.452) for M^{th}-law.

The gain of the q_M classifier over the M^{th}-law classifier, which is the additional SNR required by the M^{th}-law classifier to achieve the same performance as the q_M classifier is [6]:

$$G_{\gamma_s} = 3.45 - \frac{10}{M}\log\left(\frac{2}{S_M} \right)$$

The qM theory is reasonable at low input SNR and can be extended to medium SNR.

7. Modulation Classification of *M*FSK Signals using the Higher-Order Correlation Domain

In this section, a class of classifiers that use higher-domain correlation (HOC) proposed by Beidas & Weber, [7] & [8], are discussed. Different cases that exhaust all possible relations between symbol duration and bandwidth are considered. These classifiers are of the synchronous kind that they assume perfect knowledge of the signal arrival.

7.1 Same Symbol Duration

Comparing the difference between the associated log-ALF rules can classify between M_0FSK and M_1FSK. The optimal statistic to be compared to a theshold is given by [8] as:

$$Z_{opt} = \sum_{i=1}^{N} \ln\left(1 + \frac{\lambda_L^{(i)} + \lambda_U^{(i)}}{\lambda_M^{(i)}}\right) \qquad (7-1)$$

where,

$$\lambda_L^{(i)} = \sum_{m=-\frac{M_1}{2}+1}^{-\frac{M_0}{2}} I_0\left(\frac{\sqrt{2S}}{N_0}\left|R_{2m-1}^{(i)}\right|\right) \qquad (7-2)$$

$$\lambda_M^{(i)} = \sum_{m=-\frac{M_0}{2}+1}^{\frac{M_0}{2}} I_0\left(\frac{\sqrt{2S}}{N_0}\left|R_{2m-1}^{(i)}\right|\right) \qquad (7-3)$$

$$\lambda_U^{(i)} = \sum_{m=\frac{M_0}{2}+1}^{\frac{M_1}{2}} I_0\left(\frac{\sqrt{2S}}{N_0}\left|R_{2m-1}^{(i)}\right|\right) \qquad (7-4)$$

$$R_m^{(i)} = \int_{(i-1)T_s}^{iT_s} \tilde{r}(t) e^{-j2\pi\frac{m}{2T_s}t}\, dt \qquad (7-5)$$

and $N = T/T_s$,(number of observed symbols).

The optimal classifier is in the frequency domain, and requires the evaluation of the Fourier spectrum of the received waveform at a set of candidate frequencies, that are then separated into two sets. A set of those used by the M_0FSK and those that are not.

In a non-channelized approach, the signal frequency band is divided into three sub-bands., lower, middle and upper sub-bands. Three parallel streams of data is extracted from the received waveform $r(t)$ labeled $r'_L(t)$, $r'_M(t)$ and $r'_U(t)$, each having its spectral content solely within on of the sub-bands.

Defining:

$$\tilde{r}_j^{(i)}(t) = \tilde{r}_j\left(t + (i-1)T_s\right);\ \ j = L, M, U \qquad (7-6)$$

as given by [8], then the resultant per-branch per-symbol discrete-time sequence, [8], is:

$$\tilde{r}_j^{(i)}[k] = \tilde{r}_j^{(i)}\left(\frac{k}{\frac{B_1 - B_0}{2}}\right);\ \ j = L, U \qquad (7-7)$$

$$\tilde{r}_M^{(i)}[k] = \tilde{r}_M^{(i)}\left(\frac{k}{B_0}\right) \qquad (7-8)$$

where, $(B_1\text{-}B_0)/2$ is the sampling rate for the lower and upper sub-bands, and B_0 is for the middle sub-band in complex-valued samples / second. The per-symbol correlators through which each branch is processed [8]:

$$c_{1,L}^{(i)}[l] = \frac{2}{B_1 - B_0} \sum_{k=0}^{\frac{M_1 - M_0}{2} - l - 1} \tilde{r}_L^{(i)}[k]^* \tilde{r}_L^{(i)}[k + l];\quad l = 0,1,\ldots,\frac{M_1 - M_0}{2} - 1 \qquad (7-9)$$

$$c_{1,M}^{(i)}[l]=\frac{1}{B_0}\sum_{k=0}^{M_0-l-1}\tilde{r}_M^{(i)}[k]^*\tilde{r}_M^{(i)}[k+l];\qquad l=0,1,\ldots,M_0-1 \tag{7-10}$$

$$c_{1,U}^{(i)}[l]=\frac{2}{B_1-B_0}\sum_{k=0}^{\frac{M_1-M_0}{2}-l-1}\tilde{r}_U^{(i)}[k]^*\tilde{r}_U^{(i)}[k+l];\qquad l=0,1,\ldots,\frac{M_1-M_0}{2}-1 \tag{7-11}$$

Classification which utilizes 2nd-order correlation domain, another correlator on the 1st order correlation sequences is used to generate $c^{(i)}_{2,j}[l]$ for $i = 1, 2, \ldots, N$ and $j = L, M, U$.

The family members of the HOC-based procedure are:

- The 1st-order correlation-based classifier Z_1 given by [8] as:

$$Z_1=\sum_{i=1}^{N}\ln\left(1+\frac{2B_0}{B_1-B_0}\frac{\zeta_{1,L}^{(i)}+\zeta_{1,U}^{(i)}}{\zeta_{1,M}^{(i)}}\right) \tag{7-12}$$

where

$$\zeta_{1,j}^{(i)}=\sum_l\left|c_{1,j}^{(i)}[l]\right|^2. \tag{7-13}$$

is the energy of the 1st-order correlation of the ith symbol in the jth branch.

- The 2nd-order correlation-based classifier of the first kind [8]:

$$Z_2^{(1)}=\sum_{i=1}^{N}\ln\left(1+\frac{2B_0}{B_1-B_0}\frac{\zeta_{2,L}^{(i)}+\zeta_{2,U}^{(i)}}{\zeta_{2,M}^{(i)}}\right) \tag{7-14}$$

where,

$$\zeta_{2,j}^{(i)} = \sum_{l} c_{1,j}^{(i)}[l] c_{2,j}^{(i)}[l]^{*}. \qquad (7-15)$$

- The 2nd-order correlation-based classifier of the second kind [8]:

$$Z_2^{(1)} = \sum_{i=1}^{N} \ln\left(1 + \frac{2B_0}{B_1 - B_0} \frac{\eta_{2,L}^{(i)} + \eta_{2,U}^{(i)}}{\eta_{2,M}^{(i)}}\right) \qquad (7-16)$$

where,

$$\eta_{2,j}^{(i)} = \sum_{l} \left| c_{2,j}^{(i)}[l] \right|^2 \qquad (7-17)$$

is the energy contained in the 2nd-order correlation domain of the *i*th symbol in the *j*th branch.

The energy-based classifier is used for comparative purposes and is given by [8] as:

$$Z_e = \sum_{i=1}^{N} -\left[\frac{1}{B_0} - \frac{1}{B_1}\right]\left(\frac{c_{1,M}^{(i)}[0]}{2N_0T_s}\right) + \frac{1}{B_1}\left(\frac{c_{1,L}^{(i)}[0] + c_{1,U}^{(i)}[0]}{2N_0T_s}\right) \qquad (7-18)$$

It ensemble mean and variance are [8]:

$$E\{Z_e | H_0\} = -N\gamma_s\left(\frac{1}{M_0} - \frac{1}{M_1}\right) \qquad (7-19)$$

$$VAR\{Z_e | H_0\} = N\left[\begin{array}{l}\left(\frac{1}{M_0} - \frac{1}{M_1}\right) \\ +2\gamma_s\left(\frac{1}{M_0} - \frac{1}{M_1}\right)^2\end{array}\right] \qquad (7-20)$$

$$E\{Z_e|H_1\} = 0 \qquad (7-21)$$

$$VAR\{Z_e|H_1\} = N[\left(\frac{1}{M_0} - \frac{1}{M_1}\right) + \left(2\gamma_s + \gamma_s^2\right) \times \left(\frac{1}{M_0} - \frac{1}{M_1}\right)^2 \frac{M_0}{M_1 - M_0}] \qquad (7-22)$$

7.2 Same Bandwidths

Assuming that $M_1 = 2M_0$, and $T_{s,1} = 2T_{s,0} = 2T_s$, and let $T = NT_s$, then the optimal statistic is [8]:

$$Z_{opt} = -\sum_{i=1}^{N/2} \ln\left(\frac{L_0^{(2i-1)} \times L_0^{(2i)}}{L_1^{(i)}}\right) \qquad (7-23)$$

where,

$$L_0^{(i)} = \frac{1}{M_0} \sum_{m=\frac{-M_0}{2}+1}^{\frac{M_0}{2}} I_0\left(\frac{\sqrt{2S}}{N_0}\left|R_{0,2m-1}^{(i)}\right|\right) \qquad (7-24)$$

$$L_1^{(i)} = \frac{1}{M_1} \sum_{m=\frac{-M_1}{2}+1}^{\frac{M_1}{2}} I_0\left(\frac{\sqrt{2S}}{N_0}\left|R_{1,2m-1}^{(i)}\right|\right) \qquad (7-25)$$

$$R_{0,m}^{(i)} = \int_{(i-1)T_s}^{iT_s} \tilde{r}(t) e^{-j2\pi\frac{m}{2T_s}t dt} \qquad (7-26)$$

$$R_{1,m}^{(i)} = \int_{2(i-1)T_s}^{2iT_s} \tilde{r}(t) e^{-j2\pi\frac{m}{4T_s}t dt} \qquad (7-27)$$

In a non-channelized approach, the tone generated by the signal under hypothesis H_1 is twice as thin and four times as tall in the spectral domain than that under the H_0 hypothesis. A pre-processor is used to demodulate the observation down to baseband and then it is put through an ideal low-pass filter of bandwidth $B/2$. The resultant is sampled at B (complex-valued samples/second), and processed through a per-symbol correlator operating at the ith symbol to produce as in [8]:

$$c_{1,a}^{(i)}[l] = \frac{1}{B}\sum_{k=0}^{M_0-1} \tilde{r}_a^{(i)}[k]^* \tilde{r}_a^{(i)}[k+l]; \qquad l = 0,1,\ldots,M_0-1 \qquad (7-28)$$

$$c_{1,b}^{(i)}[l] = \frac{1}{B}\sum_{k=0}^{M_1-1} \tilde{r}_b^{(i)}[k]^* \tilde{r}_b^{(i)}[k+l]; \qquad l = 0,1,\ldots,M_1-1 \qquad (7-29)$$

where,

$$\tilde{r}_a^{(i)}[k] = \tilde{r}[k+(i-1)M_0] \qquad (7-30)$$

$$\tilde{r}_b^{(i)}[k] = \tilde{r}[k+(i-1)M_1] \qquad (7-31)$$

A family of HOC-based procedures is obtained as [8]:

- The 1st-order correlation-based classifier Z_1:

$$Z_1 = -\sum_{i=1}^{N/2} \ln\left(\frac{\zeta_{1,a}^{(2i-1)} \times \zeta_{1,a}^{(2i)}}{\zeta_{1,a}^{(i)}}\right) \qquad (7-32)$$

where,

$$\zeta_{1,a}^{(i)} = 1 + \alpha_1 c_{1,a}^{(i)}[0] + \frac{\alpha_2}{B} \sum_{l=-(M_0-1)}^{M_0-1} \left| c_{1,a}^{(i)}[l] \right|^2 \quad (7-33)$$

$$\zeta_{1,b}^{(i)} = 1 + \alpha_1 c_{1,b}^{(i)}[0] + \frac{\alpha_2}{B} \sum_{l=-(M_1-1)}^{M_1-1} \left| c_{1,b}^{(i)}[l] \right|^2 \quad (7-34)$$

where α_1 and α_2 are some constants.

- The 2nd-order correlation-based classifier of the first kind:

$$Z_2^{(1)} = -\sum_{i=1}^{N/2} \ln\left(\frac{\zeta_{2,a}^{(2i-1)} \times \zeta_{2,a}^{(2i)}}{\zeta_{2,b}^{(i)}} \right) \quad (7-35)$$

where,

$$\zeta_{2,a}^{(i)} = \zeta_{1,a}^{(i)} + \frac{1}{B}\alpha_3 \sum_{l=-(M_0-1)}^{M_0-1} c_{1,a}^{(i)}[l]\left(c_{2,a}^{(i)}[l]\right)^* \quad i = 1,2,\ldots,N, \text{and } \alpha_3 \text{ is a constant.} \quad (7-36)$$

$$\zeta_{2,b}^{(i)} = \zeta_{1,b}^{(i)} + \frac{1}{B}\alpha_3 \sum_{l=-(M_1-1)}^{M_1-1} c_{1,b}^{(i)}[l]\left(c_{2,b}^{(i)}[l]\right)^* \quad (7-37)$$

- The 2nd-order correlation-based classifier of the second kind:

$$Z_2^{(2)} = -\sum_{i=1}^{N/2} \ln\left(\frac{\eta_{2,a}^{(2i-1)} \times \eta_{2,a}^{(2i)}}{\eta_{2,b}^{(i)}} \right) \quad (7-38)$$

where,

$$\eta_{2,a}^{(i)} = \zeta_{1,a}^{(i)} + \frac{1}{B}\alpha_4 \sum_{l=-2(M_0-1)}^{2(M_0-1)} \left| c_{2,a}^{(i)}[l] \right|^2 \quad i = 1,2,\ldots,N, \text{and } \alpha_4 \text{ is a constant.} \quad (7-39)$$

$$\eta_{2,b}^{(i)} = \zeta_{1,b}^{(i)} + \frac{1}{B}\alpha_4 \sum_{l=-2(M_1-1)}^{2(M_1-1)} \left| c_{2,b}^{(i)}[l] \right|^2 \qquad (7-40)$$

The energy-based rule cannot be used in this case (same bandwidths case) as a classification alternative. One more correlation computer stage needs to be employed to produce 3rd-order correlation-based statistic Z_3.

7.3 The HOC-based Method

The HOC-based method delivers a performance which tightly lowerbounds that of the optimal likelihood-ratio test and it posses an immunity to imperfect knowledge of exact frequency locations. The performance of Z_3 is very close to that of Z_{opt} and thus using HOC of higher than the third degree will not be commensurate with the effort.

8. Modulation Classification using a Neural Tree Network

In this section the Neural Tree Network (NTN) classifier for estimating modulation type for digitally modulated signals is considered. This classifier implements a sequential linear decision strategy and does not require statistical analysis of the features. The features are obtained from an autoregressive model of the signal. The modulation types considered here are continuous wave (CW), phase shift keying (PSK), and frequency shift keying (FSK). Farrell & Mammone proposed the NTN classifier in [9].

8.1 Feature Extraction

The modulation types considered have the general form of:

$$s(k) = A\cos\left(\omega_c k + \phi(k) + \theta\right) \qquad (8-1)$$

where A is the amplitude, ω_c is the carrier frequency, θ is the phase, and $\phi(k)$ corresponds to the modulation type as follows [9]:

$$\phi(k) = \begin{cases} 0 & CW \\ 0,\pi & BPSK \\ 0,\pm\frac{\pi}{2},\pi & QPSK. \\ \pm\omega_d k & BFSK \\ \pm\frac{\omega_d}{2}k,\pm\omega_d k & QFSK \end{cases} \qquad (8-2)$$

where, ω_d is the frequency deviation for the FSK modulation type.

The instantaneous frequency is derived using the autoregressive approach so as to determine the modulation types in equation (8-2). The autoregressive method used here is

the autocorrelation method, an alternative to Fourier analysis for obtaining the frequency spectrum of a signal. Estimates of the autocorrelation terms of an input signal of x(k) can be found by [9]:

$$\hat{R}_{xx}(k) = \sum_{n=0}^{M} x(n)x(n+k) \qquad (8-3)$$

where M is the number of samples in the analysis frame. The parametric model of the spectrum is obtained from solving [9]:

$$\begin{bmatrix} \hat{R}_{xx}(0) & \hat{R}_{xx}(1) & \cdots & \hat{R}_{xx}(N-1) \\ \hat{R}_{xx}(1) & \hat{R}_{xx}(0) & \cdots & \hat{R}_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{R}_{xx}(N-1) & \hat{R}_{xx}(N-2) & \cdots & \hat{R}_{xx}(0) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} \hat{R}_{xx}(1) \\ \hat{R}_{xx}(2) \\ \vdots \\ \hat{R}_{xx}(N) \end{bmatrix} \qquad (8-4)$$

where the vector a represents the polynomial coefficients whose inverse best fits the frequency spectrum, and whose Z-domain expression given by [9] is:

$$1 - a_1 z^{-1} - a_2 z^{-2} - \ldots - a_N z^{-N},$$

and the corresponding factored form is:

$$\prod_{i=1}^{N/2} \left(1 - z_i z^{-1}\right)\left(1 - z_i^* z^{-1}\right).$$

The frequency F_i and the bandwidth BW_i for a sampling frequency of F_s within an analysis frame are given as [9]:

$$F_i = \frac{F_s}{2\pi} \tan^{-1}\left[\frac{\mathrm{Im}(Z_i)}{\mathrm{Re}(Z_i)}\right] \qquad (8-5)$$

$$BW_i = -\frac{F_s}{\pi} 10\log_{10}\left[\frac{1}{\mathrm{Im}(Z_i)^2 + \mathrm{Re}(Z_i)^2}\right]. \quad (8-6)$$

The statistics of these features can be used for classification of modulation type. The standard deviation of the instantaneous frequency *(IF)* for example can be used to distinguish between *FSK* and non-*FSK* signals. The standard deviation of the instantaneous bandwidth *(IB)* can be used to separate *CW* and *PSK* signals. The first order difference of the instantaneous frequency can be used to distinguish between *BPSK* and *QPSK*, and also *BFSK* and *QFSK* signals and it is given in [8] as:

$$\Delta F(k) = F(k) - F(k-1) \quad (8-7)$$

The feature vector to be used by the classifier is:

$$X = \left[\sigma_{IF}, \sigma_{IB}, \mu_{\Delta IF(peaks)}\right]. \quad (8-8)$$

8.2 NTN Algorithm

NTN uses a tree architecture to implement a sequential linear decision strategy. Each level has some nodes. Each node corresponds to a decision and uses an exclusive subset of the training data. Each node consists of a maximum of five neurons corresponding to the five modulation types in this case.

The feature vector is applied to the root node of the trained NTN. The output of each neuron is computed, and the selected path corresponds to the neuron with the largest output. This process goes on until the feature vector arrives at a leaf node where it is assigned the label of that node.

An M-class NTN is trained by training a neuron for each class in a binary fashion, in which the data for the class corresponding to that neuron is labeled '1' and for other classes as a '0'. For the training a gradient descent algorithm is used where the neuron output is found by [8]:

$$y_i = f\left(\left(< W, X_i >\right)\right) \qquad (8-9)$$

where,

$$f(x) = \frac{1}{1+e^{-x}} \qquad (8-10)$$

is the sigmoid function. An error measure is [9]:

$$\varepsilon_i = t_i - y_i \qquad (8-11)$$

where t_i is the target label ('1' or '0').

The gradient of an error function is evaluated with respect to the weights to obtain the weight update according to [9]:

$$W^{k+1} = W^k - \lambda \frac{\partial E_i}{\partial W^k} \qquad (8-12)$$

where λ is a scaling parameter and

$$E_i = |t_i - y_i| \qquad (8-13)$$

The update equation for the weight vector is [9]:

$$W^{k+1} = W^k - \lambda y_i (1-y_i)\,\mathrm{sgn}(\varepsilon_i) X_i \qquad (8-14)$$

where for a positive ε, sgn(ε) outputs a (+1) and a (-1) otherwise.

8.3 NTN Classification

The neural tree network is a self-organizing, hierarchical classifier that implements a sequential linear decision strategy. It has several advantages over the traditional statistical and decision tree based approaches. It does not require information regarding the statistical properties of the features. According to Farrell & Mammone (1993), both the NTN and decision tree are applied to several experiments for classifying the modulation type of digitally modulated signals. The NTN performs well for CNRs as low as 10 dB besides consistently outperforming the decision tree.

9. Automatic Modulation Recognition using Neural Networks

Automatic Modulation Recognition (AMR) approach using back-error propagation neural networks are introduced in this section. This approach is based on the characteristics of modulated signals that have the general form as in [10] of:

$$s(t) = A_c A(t)\cos\left(2\pi f_c t + \varphi(t) + \theta_0\right), \qquad (9-1)$$

where $A(T)$ is the signal envelope, $\varphi(t)$ is the zero phase, A_c controls the carrier power, f_c is the carrier frequency and θ_0 is the initial phase angle. The baseband message determines $A(t)$ and $\varphi(t)$ according to the modulation type.

9.1 Classification Algorithm

Preprocessing is used first to remove noise and irrelevant information from the signals. Spectral techniques were chosen for preprocessing due to the nature of the signals considered.

Inspired from the biological neural system, an artificial neural network is a set of interconnected computing units (neurons), organized in a layer structure. Two layers are usually sufficient, and the first layer is called *hidden* as it is in between the inputs and outputs. Typically, the network is fully connected, each unit in any layer is connected to all units in all layers before and after. In backpropagation networks, an input vector is presented to the network, the output is computed and then compared to the desired output generating an error signal. This error signal is propagated backwards updating the weights. The aim of this is to minimize the squared error between the generated and desired outputs.

Choosing the right number of hidden layer units is very important in successfully applying the backpropagation networks to a problem. This can be achieved by a pruning

network technique devised by Cun, Dender & Solla (1990) cited in [10], called the Optimal Brain Damage (OBD), and is based on the second derivative of the error function. OBD works as follows:

Let $u_k(n)$ be a weight in the network [10]:

$u_k(n) = w_{ji}(n)$ for all $(j,i) \in V_k$,
(V_k is a complete set of index pairs).

Haykin cited in [10] defined $\delta u_k(n)$ as the perturbation in the value of parameter $u_k(n)$. The Taylor series expansion of the network error $E(n)$ given by [10] is:

$$\delta E(n) = \sum_k \frac{\delta E(n)}{\delta u_k(n)} u_k(n) + \frac{1}{2}\sum_k h_{kk}(n)\delta u_k^2(n) + \frac{1}{2}\sum_j \sum_{kj\neq k} h_{jk}(n)\delta u_j(n)\delta u_k(n) + \ldots \qquad (9-2)$$

where $h_{jk}(n)$ is the jk-th element of the Hessian matrix of E(n) given in [10] as:

$$h_{jk}(n) = \frac{\delta^2 E(n)}{\delta u_j(n)\delta u_k(n)}. \qquad (9-3)$$

If $E(n)$ is quadratic around the minimum and the weights are detected after the network has converged, then (9-2) can be simplified [10] as:

$$\delta E(n) = \frac{1}{2}\sum_k h_{kk}(n)\delta u_k^2(n) \qquad (9-4)$$

A "saliency" measure is proposed for the weights whose saliency is below a certain thershold are deleted until a satisfactory performance level in attained. The saliency is given in [10] as:

$$s_{uk}(n) = \frac{1}{2}\sum_{k} h_{kk} u_k^2(n). \qquad (9-5)$$

Extensive experimentations are required to determine the saliency threshold. A small value results in a limited gain while a large value results in a sharp decrease in performance.

9.2 Backpropagation Neural Network Classifiers

Backpropagation neural network classifiers were implemented for the AMR problem according to [10] with 10 signal classes. Regular periodogram, Welch's periodogram and bispectrum preprocessing paradigms were investigated. The best performance was that of the truncated scaled-log Welch periodogram spectrum estimator of 98.6%.

This technique was compared with the k-nearest neighbour classifier (k-NN), which is a modification of the Parzen-window estimator for arbitrary density functions. It determines which density is larger than the others at a given point. It can be concluded from the comparison between the two methods that the optimum backpropagation networks gave lower error rates on a test set over most classes and a lower overall error. The disadvantage however is that the training time is 3 times of that required to compute 1-nearest neighbour in the k-NN method for 10000 test points. Once a network is trained, forward passes for strict classification is faster.

10. A Decision-Theoretic Approach and Artificial Neural Algorithms for Modulations Recognition

Two algorithms for analog and digital modulations recognition are introduced in this section, a decision-theoretic approach and an artificial neural network approach (ANN). Both algorithms were introduced by Nandi & Azzouz (1998), in [11].

10.1 The Decision-Theoretic Approach

In this method, the intercepted signal frame of length K is divided into M (= Kf_s/N_s where fs is the sampling rate) successive segments of N_s = 2048 samples in length each. This is equivalent of 1.76 ms.

10.1.1 Classification of Each Segment

Key features extraction and modulation classification are required to discriminate between different types of modulation from each of the segments available.

A) Key Features Extraction:

Only the signal spectrum symmetry feature is derived from the RF signal spectrum, while all the other key features are derived from the instantaneous amplitude, phase, and frequency of the intercepted signal. The key features used in the proposed modulation recognizer are:

1. The maximum value of the spectral power density of the normalized-centered instantaneous amplitude γ_{max}, given by [11] as:

$$\gamma_{\max} = \max DFT\left[A_{cn}(i)\right]^2 / N_s \qquad (10-1)$$

where, $A_{cn}(i)$ is the value of the normalized-centered instantaneous amplitude at time instants $t = i/f_s$, in which the $i = 1, 2, \ldots, N_s$, f_s is the sampling rate (1200kHz), as given in [12]:

$$A_{cn}(i) = A_n(i) - 1 \qquad (10-2)$$

where,

$$A_n(i) = \frac{A(i)}{m_a} \qquad (10-3)$$

is the normalized instantaneous amplitude at time instants $t = i/f_s$, and m_a is the average value of the instantaneous amplitude over one frame given in [12] as:

$$m_a = \frac{1}{N_s}\sum_{i=1}^{N_s} A(i). \qquad (10-4)$$

2. The standard deviation of the absolute of the nonlinear component of the instantaneous phase in the nonweak segments of a signal σ_{ap}.

$$\sigma_{ap} = \left[\frac{1}{C}\left(\sum_{A_n(i)>a_t}\phi_{NL}^2(i)\right) - \left(\frac{1}{C}\sum_{A_n(i)>a_t}\left|\phi_{NL}(i)\right|\right)^2\right]^{1/2} \qquad (10-5)$$

where $\phi_{NL}(i)$ is the value of the non-linear component of the instantaneous phase at time instants $t = i/f_s$ and C is the number of samples in $\{\phi_{NL}(i)\}$ for which $A_n(i) > a_t$.

3. The standard deviation of the direct value of the nonlinear component of the instantaneous phase in the nonweak segments of a signal σ_{dp}.

$$\sigma_{dp} = \left[\frac{1}{C} \left(\sum_{A_n(i) > a_t} \phi_{NL}^2(i) \right) - \left(\frac{1}{C} \sum_{A_n(i) > a_t} \phi_{NL}(i) \right)^2 \right]^{1/2} \qquad (10-6)$$

4. The ratio P, which measures the spectrum symmetry of the RF signal, given in [11] as:

$$P = (P_L - P_U) / (P_L + P_U) \qquad (10-7)$$

where,

$$P_L = \sum_{i=1}^{f_{cn}} X_c(i)^2, \qquad P_U = \sum_{i=1}^{f_{cn}} X_c(i + f_{cn} + 1)^2$$

and $(f_{cn} + 1)$ is the sample number corresponding to a carrier frequency of 150 kHz).

5. The standard deviation of the absolute value of the normalized-centered instantaneous amplitude of a signal σ_{aa}, given by [13] as:

$$\sigma_{aa} = \sqrt{\frac{1}{N} \left(\sum_{i=1}^{N} A_{cn}^2(i) \right) - \left(\frac{1}{N} \sum_{i=1}^{N} |A_{cn}(i)| \right)^2}. \qquad (10-8)$$

6. The standard deviation of the absolute value of the normalized instantaneous frequency of a signal σ_{af} or σ_{fa} given by [13] as:

$$\sigma_{fa} = \sqrt{\frac{1}{C} \left(\sum_{A_n(i) > a_t} f_N^2(i) \right) - \left(\frac{1}{C} \sum_{A_n(i) > a_t} |f_N(i)| \right)^2} \qquad (10-9)$$

where

$$f_N(i) = f_c(i) / r_b,$$

$$f_c(i) = f(i) - m_f ; \; m_f = \frac{1}{N}\sum_{i=1}^{N} f(i),$$

and r_b is the bit rate.

7. The standard deviation of the normalized-centered instantaneous amplitude in the nonweak segment of a signal defined as in [11]:

$$\sigma_a = \sqrt{\frac{1}{L}\left[\sum_{A_n(i)>a_t} A_{cn}^2(i)\right] - \left[\frac{1}{L}\sum_{A_n(i)>a_t} A_{cn}(i)\right]^2} \qquad (10-10)$$

where L is the number of samples in $\{C_{an}(i)\}$ for which the band-limitation is more affected on the intantaneous
amplitude of the band-limited PSK2 and the band-limited PSK4 signals.

8. The kurtosis of the normalized instantaneous amplitude
given in [11] as:

$$\mu_{42}^a = E\{A_n^4(t)\} / \{E\{A_n^2(t)\}\}^2 \qquad (10-11)$$

9. The kurtosis of the normalized instantaneous frequency as in [11]:

$$\mu_{42}^f = E\{f_n^4(t)\} / \{E\{f_n^2(t)\}\}^2 \qquad (10-12)$$

where, $f_n(t)$ is the normalized instantaneous frequency defined as: $f_n(t) = f(t) /$ max $\{f(t)\}$, and $f(t)$ is the instantaneous frequency of the intercepted signal

B) Modulation Classification Procedure

In the decision theoretic approach, a group of modulations is compared with a threshold by every key feature and is then divided into two possible sets. The modulation classification procedure is shown as a flow chart in Figure 10-1 as in [11].

10.1.2 Classification of a Signal Frame

Different classifications can be obtained from these M-segments and thus the majority logic rule, in which the classification with the largest number of repetitions is selected, is used. When more than one classification have the same number of repetitions, then all of those are regarded as candidates for optimal decision. The procedure then works as follows:

- Segments corresponding to each of the candidate decisions are grouped together.
- For every frame within a group, the number of samples of instantaneous amplitude falling below the threshold a_t is determined, and the total numbers of these samples over the group.
- The decision whose corresponding group has minimum number of samples below the threshold a_t is adopted.

10.2 Artificial Neural Network Approach

As shown in Figure 10-2, the ANN recognizer is composed of three blocks:

1. The preprocessing in which the key features are extracted.
2. The training and learning phase to adjust the classifier structure.
3. The test phase to obtain the modulation type.

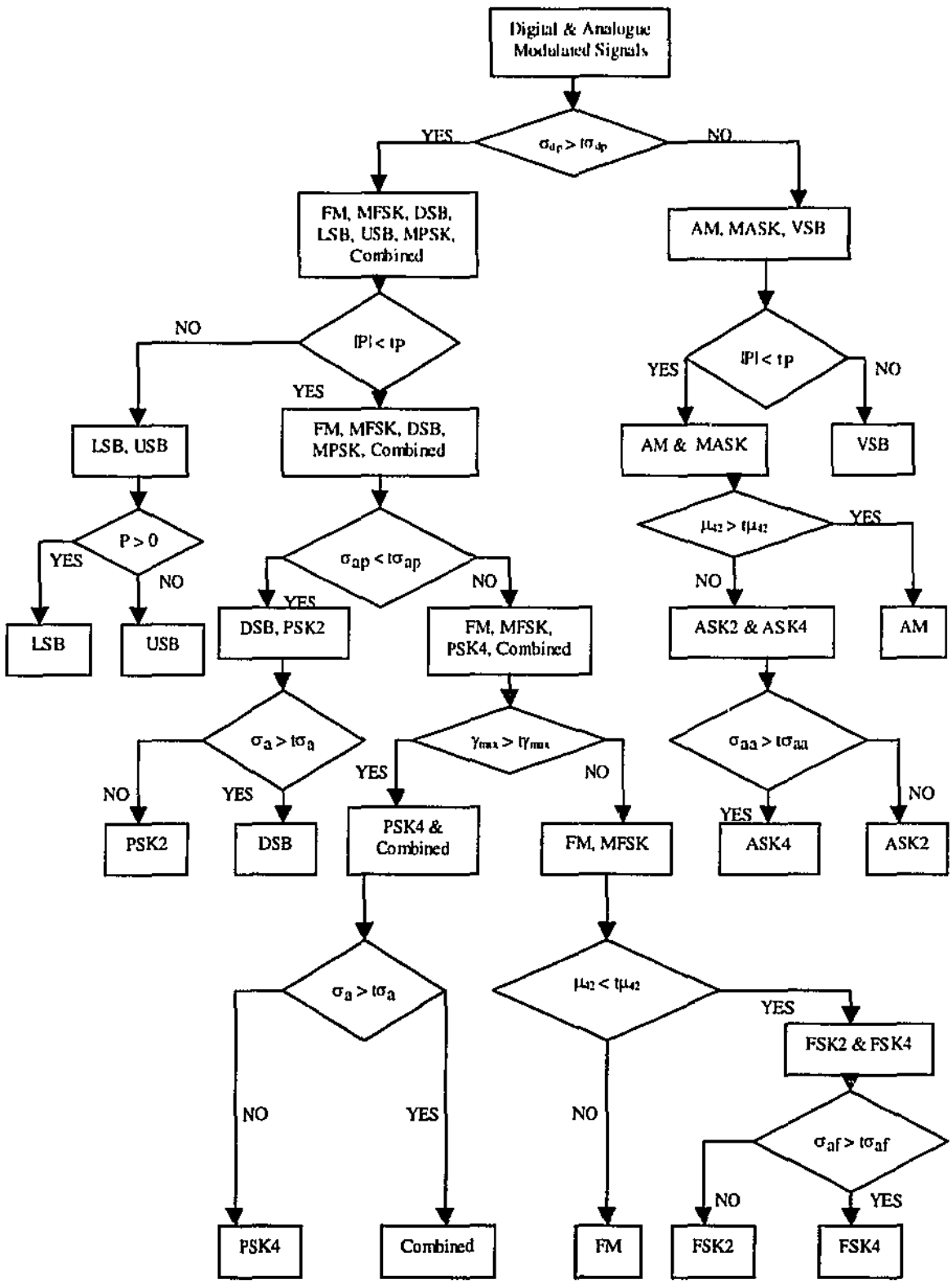

Figure 10-1 Functional Flowchart for the Decision-Theoretic Approach.Functional Flowchart for the Decision-Theoretic Approach.

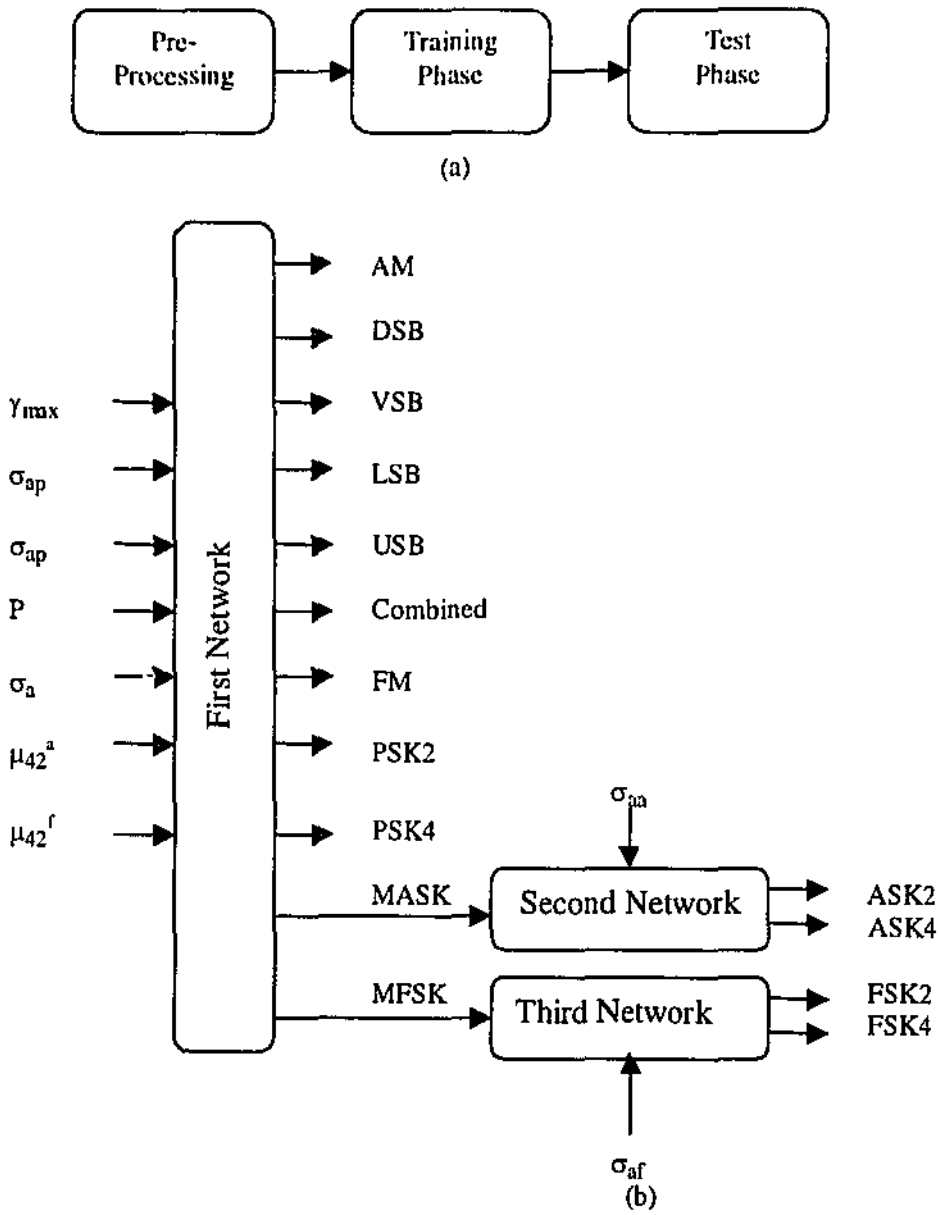

Figure 10-2 Functional blocks of the ANN algorithm.

Part (b) of shows that the algorithm requires three ANNs. One network is required to discriminate between all of the modulation types except for MASK and MFSK, which require an additional network. So one more network is required for the discrimination between ASK2 and ASK4, and another one for FSK2 and FSK4.

1) The First Network

The first network structure contains a seven-node input layer, eleven-node output layer and two hidden layers.

2) The Second Network

This network is used when the output decision of the first network is MASK. It contains one-node input layer and two-node output layer with no hidden layers.

3) The Third Network

This network is used when the output decision of the first network is MFSK. It contains one-node input layer and two-node output layer with no hidden layers, the same as the second network.

10.3 Both Recognizers

The optimum values for the key features were obtained in [11]. Extensive simulations has been carried outout different SNR's in [11]. Results for both algorithm are found to be encouraging. The overall success rate in the decision-theoretic algorithm is found to be over 94% at SNR of 15 dB while that of the ANN algorithm is over 96%.

References

[1] Hsue, S., & Soliman, S. S., "Automatic modulation recognition of digitally modulated signals". Southern Methodist University, TX, USA, Tech. Report EE 89003, July 1989.

[2] Hsue, S., & Soliman S. S., "Automatic modulation classification using zero crossing". *IEEE PROC*, Vol. 137, No. 6, Dec. 1990.

[3] Polydoros, A., & Kim, K., "On the Detection and Classification of Quadrature Digital Modulations in Broad-Band Noise". *IEEE Trans. Commun.*, Vol. 38, No. 8, pp. 1199-1211, August 1990.

[4] Soliman, S. S., & Hsue, S., "Signal Classification Using Statistical Moments". *IEEE Trans. Commun*, Vol. 40, No. 5, May. 1992.

[5] Lay, N. E., & Polydoros A., "Modulation Classification of Signals in Unknown ISI Environments". *IEEE 1995 Conference Proceedings.*

[6] Hwang, C. Y., & Polydoros A., "Advanced Methods For Digital Quadrature And Offset Modulation Classification". *IEEE 1991 Conference Proceedings.*

[7] Beidas, B. F., & Weber, C. L., "Higher-Order Correlation-Based Approach to Modulation Classification of Digitally Frequency-Modulated Signals". *IEEE 1995 Conference Proceedings.*

[8] Beidas, B. F., & Weber, C. L., "Modulation Classification of *M*FSK Signals Using the Higher-Order Correlation Domain". *IEEE 1995 Conference Proceedings.*

[9] Farrell, K. R., & Mammone, R. J., "Modulation Classification Using A Neural Tree Network". *IEEE 1993 Conference Proceedings*, pp. 1028-1032.

[10] Ghani, N., & Lamontagne R., "Neural Networks Applied to the Classification of Spectral Features for Automatic Modulation Recognition". *IEEE 1993 Conference Proceedings.*

[11] Nandi, A. K., & Azzouz, E. E., "Algorithms for Automatic Modulation Recognition of Communication Signals". *IEEE Trans. Commun.* Vol. 46, No. 4, April 1998.

[12] Nandi, A. K., & Azzouz, E. E., "Automatic Analogue Modulation Recognition". *Signal Processing.* Vol. 46, No. 2, pp. 211-222, May 1995.

[13] Nandi, A. K., & Azzouz, E. E., "Automatic Identification of Digital Modulation Types". *Signal Processing.* Vol. 47, No. 1, pp. 55-69, March 1995.