CHAPTER 5

DESIGN AND IMPLEMENTATION

5.1 CORRELATION EXTRACTOR

The correlation extractor reveals large peak in the low frequencies and is an indication that low frequency components are correlated [18]. These peaks are compensated by employing high frequency pro-emphasis filter that precedes the correlation extractor. The adaptive scheme for human motion estimation is designed in two blocks namely;

Block – I (refer Figure 5.1): This operates on the human motion input in the presence of noise and facilitate the extraction of the correlated samples using down sampler and unit delay elements.

Block – II (refer Figure 5.4): This consists of unsupervised network training stage and using the trained weights perform noise separation and constitutes the human motion estimation (MEF).

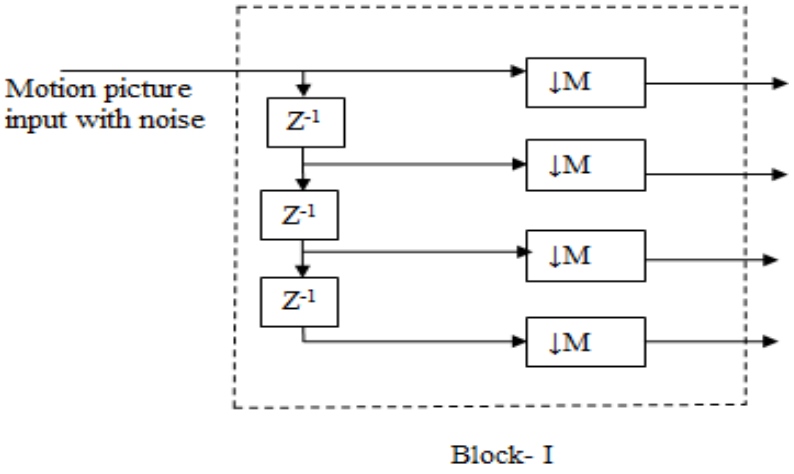


Figure 5.1 Correlation Extractor Module

Note: ↓M down sampler & z^{-1} delay by unit sample.

5.1.1 Implementation results of Correlation extractor

The output of block-I consists of the motion picture samples corresponding to$\sum_{n=0}^{N} x(4n)$, $\sum_{n=0}^{N} x(4n+1)$, $\sum_{n=0}^{N} x(4n+2)$ and $\sum_{n=0}^{N} x(4n+3)$ respectively. The correlation present among the different delayed samples i.e. x (n) to x (n-k) for k=1 to 3, of the human motion video input is depicted in Figure 5.2(a) and 5.2(b) for both Low Frequency and (ii) High Frequency Motion respectively.

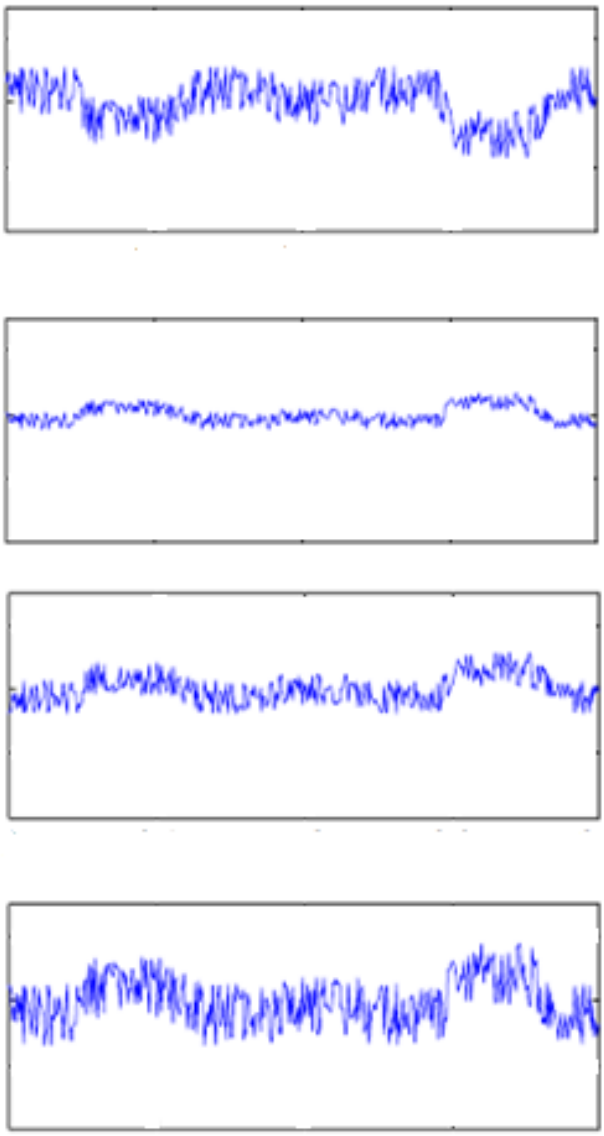


Figure 5.2(a) Block – I: Correlation among the samples and its delayed versions at lower frequency

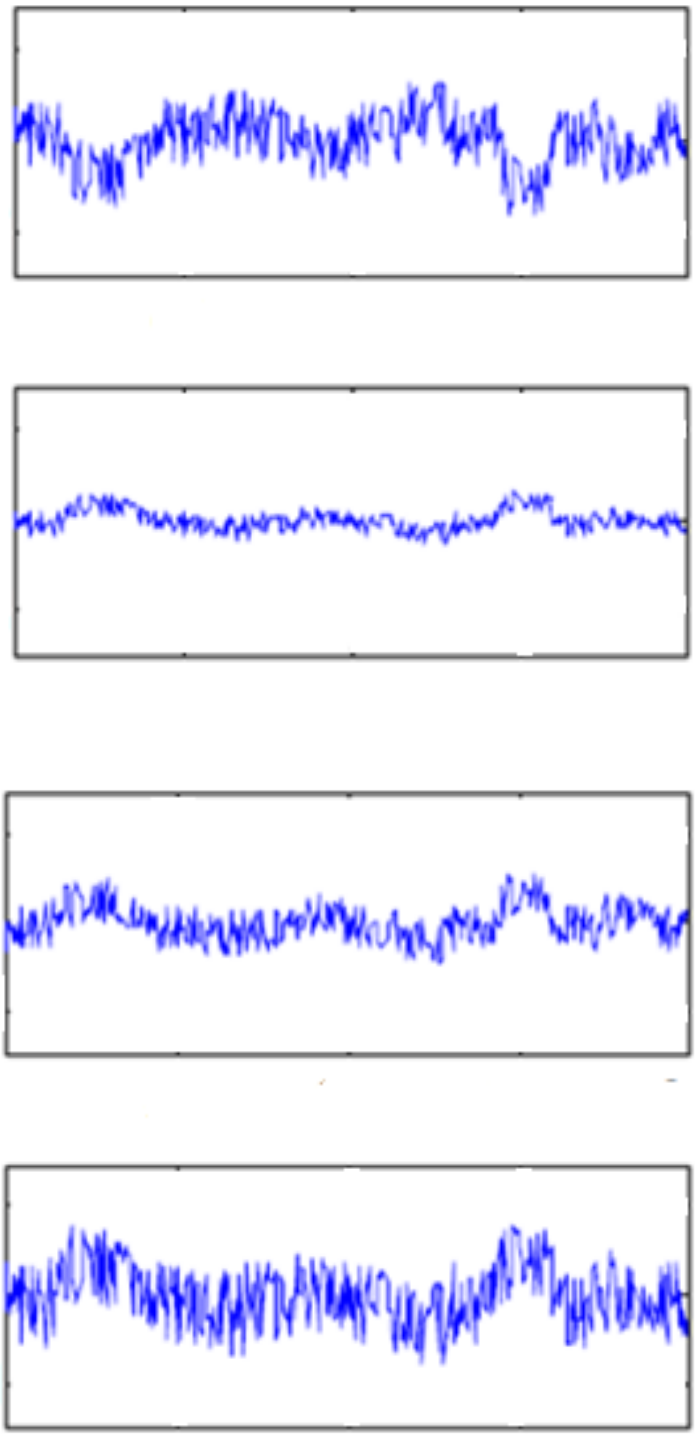


Figure 5.2(b) Block – I: Correlation among the samples and its delayed versions at higher frequency

5.2 HUMAN MOTION ESTIMATION NETWORK ARCHITECTURE

A class of human motion estimation strategies, that are ideally suited for in a real time environment favors a model in which the secondary source $n_1(k)$ is absent, while retaining the constraints that

(i) Input video (x(k)) and noise ($n_1(k)$) can be non-stationary and

(ii) No apriori statistical information is required about x(k) and $n_1(k)$.

The human motion estimation network architecture is shown in Figure 5.3. The delay path serves to generate a reference signal, that can be used to estimate x(k). It has the advantage, that, it can remove the noise by observing the mixed signal alone. No apriori statistical information about the desired signal is required to be given as input to the network. Also, the source signal x(k) need not be stationary. This motivates the use of blind signal separation (BSS) algorithms to estimate the human motion video. A neural network based independent component analysis (ICA) algorithm is incorporated into the existing system to estimate the human motion.

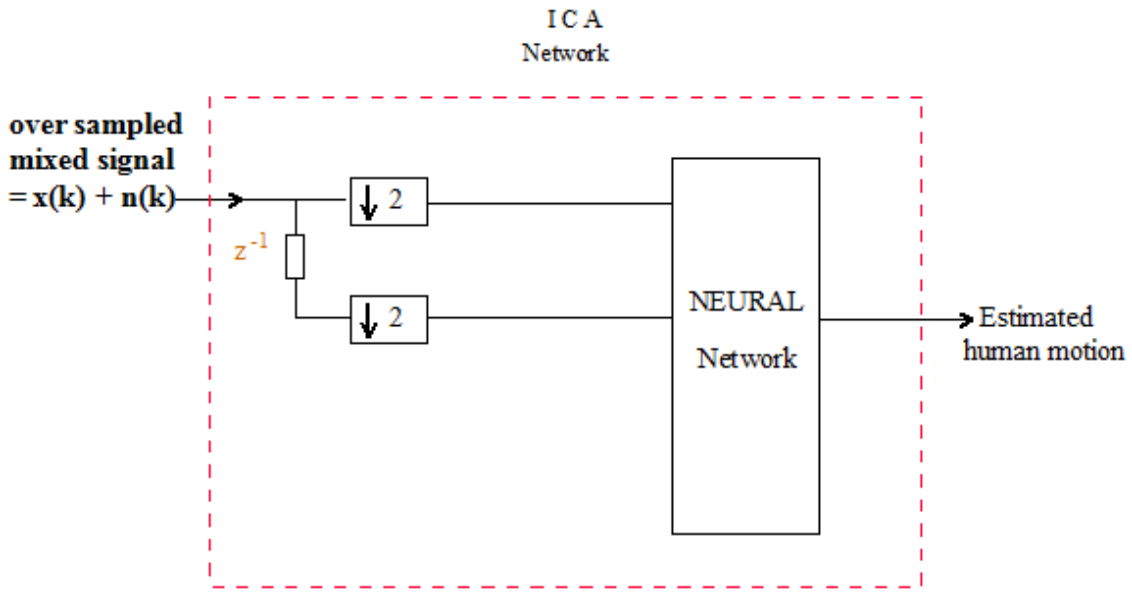


Figure 5.3 Human Motion Estimation Network Architecture

Note: ↓M M-stage decimator

5.2.1 Motion Estimation Filter (MEF)

The MEF is designed to process the output of block – I and perform lower and higher frequency human motion estimation. The main role of MEF is to separate the uncorrelated samples from the correlated ones and perform estimation of human motion as shown in Figure 5.4.

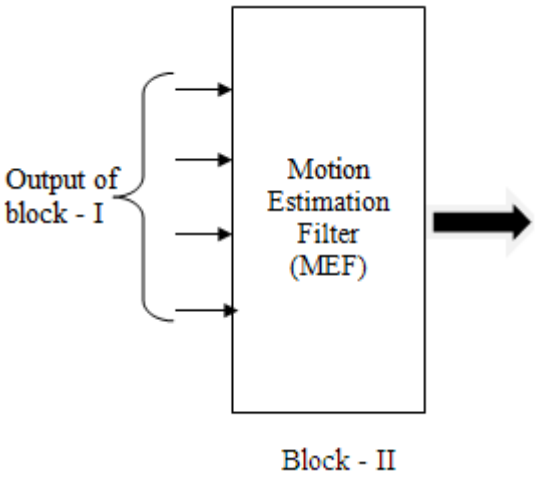


Figure 5.4 Block diagram of Motion Estimation Filter

5.2.2 Stages of MEF

The human motion estimation filter is processed in three stages and is shown in Figure 5.5.

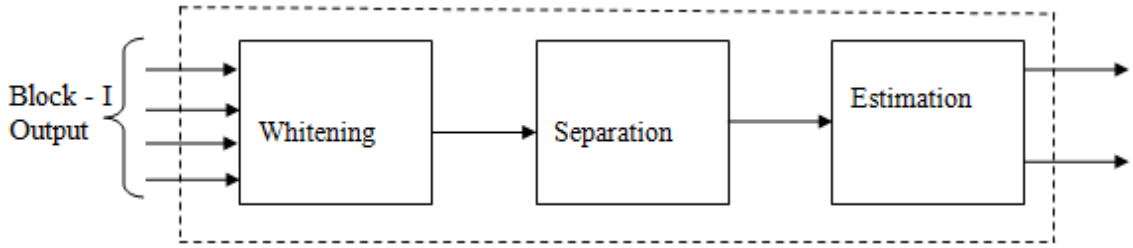


Figure 5.5 Stages of Motion Estimation Filter

Stage 1: Whitening: This stage normalizes the human motion video samples to unit variance.

Stage 2: Separation: After normalization, this stage separates the uncorrelated samples (noise) and the correlated samples (i.e. the human motion).

Stage 3: Estimation: Finally, using the trained weights, the motion estimation filter estimates the human motion video.

The human motion estimation analysis consists of three stages and it is used to remove the motion signals in the ICA network as shown in Figure 5.6.

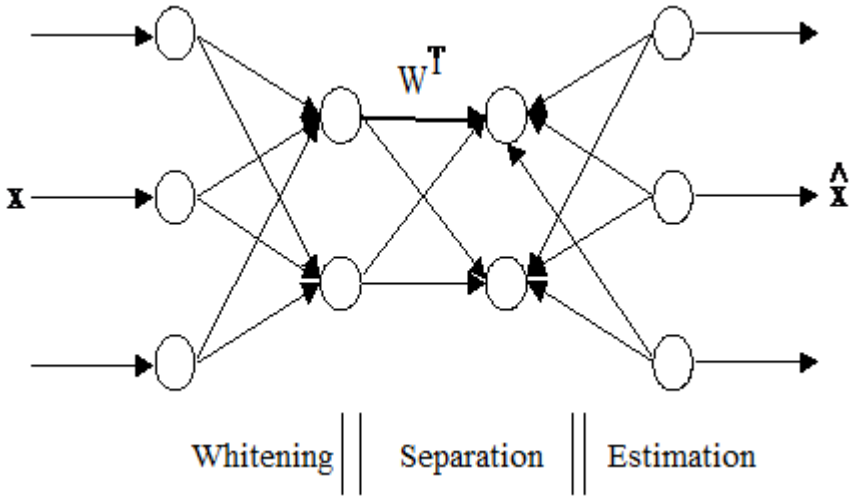


Figure 5.6 Extracting motion signals using unsupervised network

If the observed data vectors x_k have a nonzero mean, then it is desirable to remove the mean. This process is called whitening. The effects of second-order statistics can be removed by using the whitening transformation. The whitening matrix is chosen so that the covariance matrix of the whitened vectors $\{v_k v_k^T\}$ equals the unit matrix I_M. After whitening, the components of the whitened vectors v_k are mutually uncorrelated with zero mean and unit variance.

Uncorrelatedness is a necessary prerequisite for the stronger independence condition, so that after pre-whitening, the noise removal task from the observed mixed data usually becomes easier. There exist infinitely many solutions for whitening the input data, provided that the number of observations of the noisy sensor readings exceeds or atleast equals the number of source signals.

5.2.2.1 Principal component analysis (PCA) based whitening algorithm

In this research work, the standard PCA based approach is used for whitening and simultaneously compress the information optimally in the mean-square error sense and filter possible motion signal. The following steps are used in the PCA based pre-whitening process:

1) Remove the mean value from the noisy sensor readings s[k]
2) Form the covariance matrix of the observed motion signal $E\{s_k s_k^T\}$.

It is to be noted that for monotonically decreasing covariance functions, the covariance matrix is sparse.

3) Extract the matrices D and E,

where $D = diag[\lambda(1),\ldots\ldots,\lambda(M)]$ is a $M \times M$ diagonal matrix,

$E = [c(1),\ldots\ldots,c(M)]$ is a $L \times M$ matrix,

$\lambda(i)$ is the i_{th} largest eigenvalue of the data covariance matrix $E\{s_k s_k^T\}$ and

$c(i)$ is the respective i_{th} principal eigenvector.

4) Form whitening matrix V using $V = D^{-1/2} * E^T$
5) Pre-whiten the inputs using the transformation $v[k] = V*s[k]$

5.2.2.2 Separation process

The second stage in the ICA network is the separation of the source signals. This can be achieved by using suitable higher-order statistics. The separation process can be modeled as a single-layer neural network with an equal number of input and output nodes, where the coefficient w_{ij} of the separation matrix W, are simply the weights from the input to output nodes. In the present work, the noise is removed from the measured sensor readings in this stage. The nonlinear PCA sub-space learning algorithm is used in the present work, and has

the advantage that it can be realized using a simple modification of the one-layer standard symmetric PCA network.

5.2.2.2.1 Neural network based separation algorithm

The neural network based separation algorithm was implemented by using the pseudo-code of the algorithm is given below.

Step 1:

```
while (count < number of training epochs)
    {
      while (i < number of observed samples)
      {
          assign y[k] =W^T * v[k]
          assign μ = 1/( γ / μ[k-1] +  | v[k] | ^2 )
           assign W[k+1] = W[k] + μ[k] { v[k]  - W[k] * g(y[k]) } g(y^T[k])
          i++
      }
      count++
    }
```

where γ is a constant called the forgetting factor and is chosen as 0.6,

μ is the learning parameter

W is the separation matrix and

g(.) is a suitable nonlinearity.

Step 2: Estimate the source signals using $y[k] = W^T * v[k] * \sqrt{}$(number of observed samples)

5.2.2.2.2 Choice of activation function 'g(.)' used in the separation process

As a non-parametric model, the performance is clearly related to the activation function and a proper choice of the activation function g(.) is important for effective removal of noise present in the sensor readings. The activation function at the output nodes is used for the training mode only and

plays a central role in blind signal separation (BSS). The activation function provides a metric to measure the similarity between the inputs [68] and this metric is adjusted dynamically to adapt to different local regions. Its nature is defined by objective or contrast or score function. The maximum likelihood approach defines the score function as

$$g_i(u_i) = -\frac{\partial \log p_s(u_i)}{\partial u_i} = \frac{-p'_s(u_i)}{p_s(u_i)}, \qquad i = 1,\ldots,M_s \quad \ldots$$

(5.1)

where $p_s(u_i)$and $p'_s(u_i)$ are the probability density function (pdf) and its derivative, respectively, of the source signals.

5.3 MEF ALGORITHM

The human motion estimation algorithm is designed to extract the mean value of the human motion picture and identify the correlation, eigenvalues in the diagonal elements of the human motion components and the correlation coefficients between the human motion picture samples. The MEF algorithm is described in the flowchart of Figure 5.7.

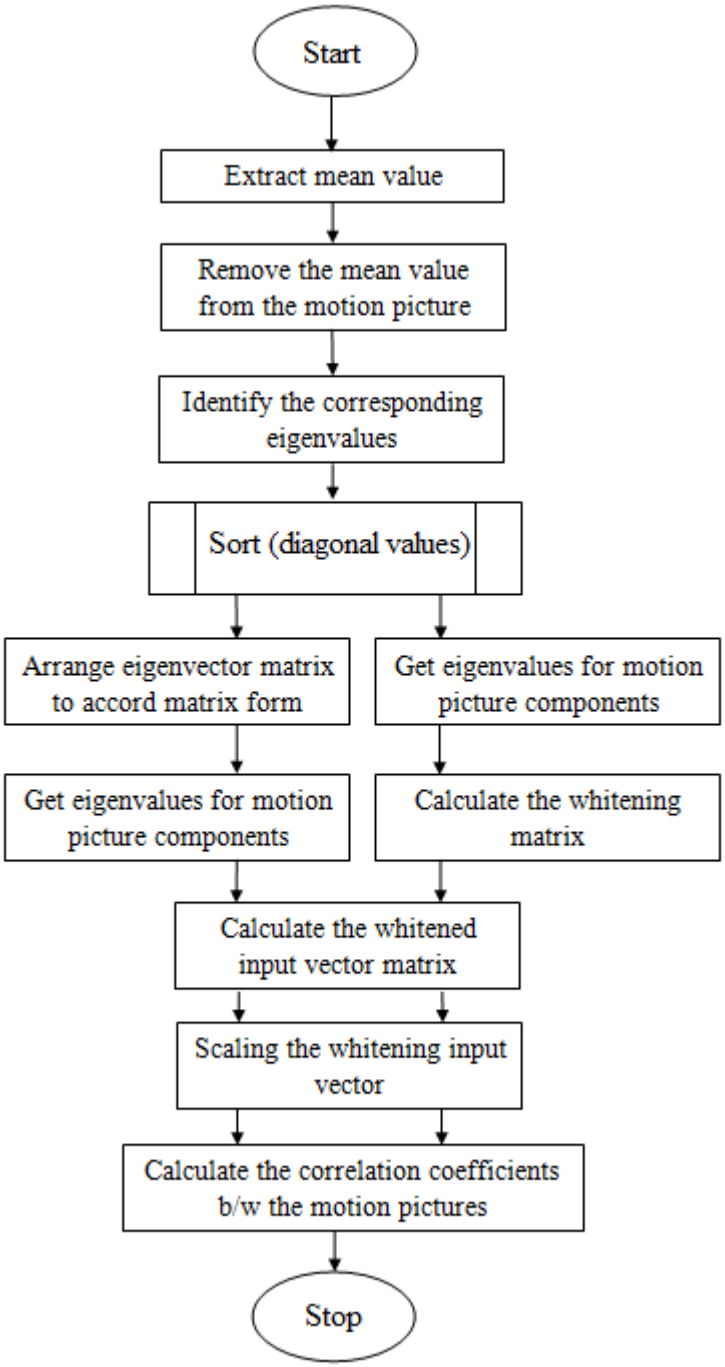


Figure 5.7 Flowchart of MEF

5.4 PHASE COMPENSATOR INTEGRATED MEF

At higher frequencies, the MEF introduces an unintentional phase shift of 180° in the estimated human motion. To compensate for phase shift, the MEF block is modified with the decision and 180° phase shift module and is shown in Figure 5.8.

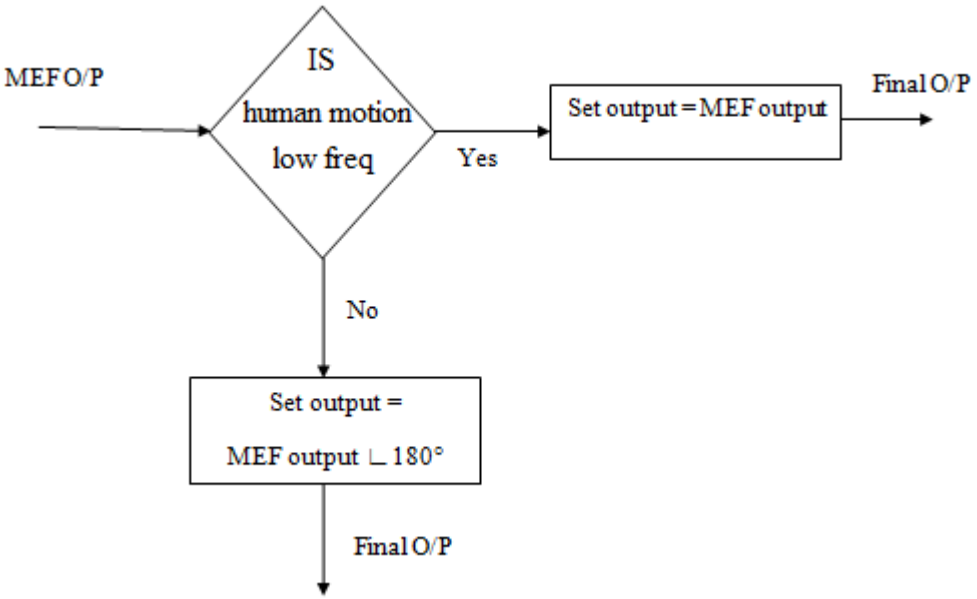


Figure 5.8 Phase compensator integrated MEF

5.4.1 Design of Phase compensator

The human motion estimation is integrated with phase compensator and the data's are observed as:

$$y(t)= A\cos(\omega_c t + \theta) + n(t), \qquad 0 \le t \le T \qquad \ldots (5.2)$$

where T is the observation interval. Expanding $A\cos(\omega_c t + \theta)$ as

$$A\cos\omega_c t \cos\theta - A\sin\omega_c t \sin\theta$$

$$\emptyset_1(t) = \sqrt{\frac{2}{T}}\cos\omega_c t, \qquad 0 \le t \le T \qquad \ldots (5.3)$$

and

$$\emptyset_2(t) = \sqrt{\frac{2}{T}}\, sin\omega_c\, t, \qquad 0 \le t \le T \qquad \ldots (5.4)$$

Thus, the decision is

$$z(t) = \sqrt{\frac{T}{2}} A\, cos\theta\, \emptyset_1(t) - \sqrt{\frac{T}{2}} A\, sin\theta\, \emptyset_2(t) + N_1\emptyset_1(t) + N_2\emptyset_2(t) \quad \ldots (5.5)$$

where

$$N_i = \int_0^T n(t)\, \emptyset_i(t)\, dt, \quad i = 1,2 \qquad \ldots (5.6)$$

Because y(t) – z(t) involves only motion signal, which is independent of z(t) and it is not relevant to making the estimation. Thus, the estimation on the vector is

$$\mathrm{Z} \triangleq (\mathrm{Z}_1, \mathrm{Z}_2) = \sqrt{\frac{T}{2}} A\, cos\theta + N_1, - \sqrt{\frac{T}{2}} A\, sin\theta + N_2 \qquad \ldots (5.7)$$

where

$$\mathrm{Z_i} = \big(y(t), \emptyset_i(t)\big) = \int_0^T \big(y(t), \emptyset_i(t)\big)\, dt \qquad \ldots (5.8)$$

The likelihood function $f_{z|\theta}(\mathrm{z_1}, \mathrm{z_2}|\theta)$is obtained by noting that the variance of $\mathrm{Z_1}$ and $\mathrm{Z_2}$ is simply $1/2\mathrm{N_0}$. Thus the likelihood function is

$$f_{z|\theta}(\mathrm{z_1}, \mathrm{z_2}|\theta) = \frac{1}{\pi \mathrm{N_0}} \exp\left\{-\frac{1}{\mathrm{N_0}}\left[(\mathrm{z_1} - \sqrt{\frac{\mathrm{T}}{2}}\mathrm{A}\cos\theta)^2 + (\mathrm{z_2} + \sqrt{\frac{\mathrm{T}}{2}}\mathrm{A}\sin\theta)^2\right]\right\} \quad (5.9)$$

which reduces to

$$f_{z|\theta}(\mathrm{z_1}, \mathrm{z_2}|\theta) = C \exp 2\sqrt{\frac{T}{2}\frac{A}{N_0}}(\mathrm{z_1}\, cos\theta - \mathrm{z_2}\, sin\theta) \qquad \ldots (5.10)$$

where the coefficient C contains all factors that are independent of 'θ'. The logarithm of the likelihood function is

$$ln\, f_{z|\theta}(z_1, z_2|\theta) = ln\, C + \sqrt{2T}\, \frac{A}{N_0}\, (\mathrm{z_1}\, cos\theta - \mathrm{z_2}\, sin\theta) \qquad \ldots (5.11)$$

which, when differentiated and set to zero, yields a necessary condition for the maximum – likelihood estimation of 'θ' in accordance with (Eq. 5.5). The result is

$$-Z_1 sin\theta - Z_2 cos\theta \big|_{\theta=\dot{\theta}_{ML}} = 0 \qquad \ldots (5.12)$$

where Z_1 and Z_2 signify a particular (random) observation

$$Z_1 = (y, \emptyset_1) = \sqrt{\frac{2}{T}} \int_0^T y(t)\, cos\omega_c\, t\, dt \qquad \ldots (5.13)$$

and

$$Z_2 = (y, \emptyset_2) = \sqrt{\frac{2}{T}} \int_0^T y(t)\, sin\omega_c\, t\, dt \qquad \ldots (5.14)$$

Therefore, Eq.5.12 can be put in the form

$$-\sin\dot{\theta}_{ML} \int_0^T y(t)\, cos\omega_c\, t\, dt - cos\, \dot{\theta}_{ML} \int_0^T y(t)\, sin\omega_c\, t\, dt = 0 \qquad \ldots (5.15)$$

or

$$\int_0^T y(t)\, sin(\omega_c\, t + \dot{\theta}_{ML})\, dt = 0 \qquad \ldots (5.16)$$

Eq. 5.16 can be interpreted as the feedback structure shown in Figure 5.9 except for the integrator replacing a loop filter to the phase compensator. A lower bound for the variance of '$\dot{\theta}_{ML}$' is obtained. Applying Eq. (5.11) the first differentiation is given by

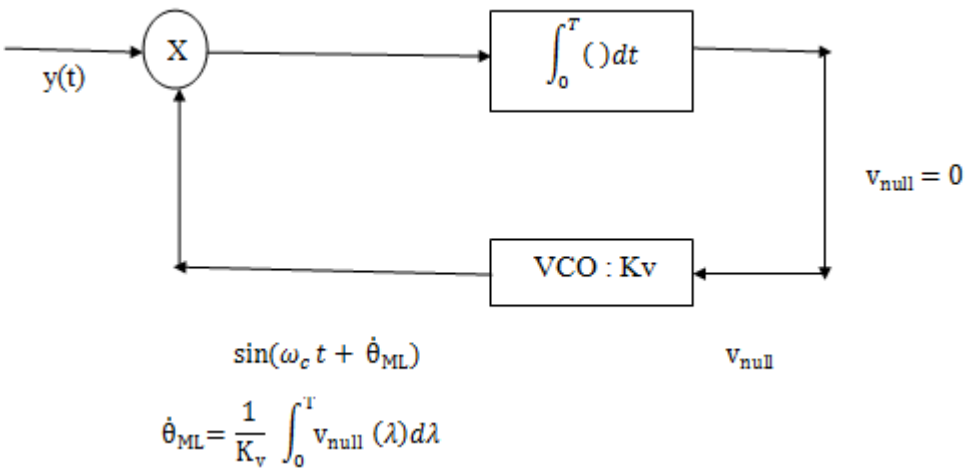


Figure 5.9 ML estimator for phase compensator

$$\frac{\partial \ln f_{z|\theta}}{\partial \theta} = \sqrt{2T}\,\frac{A}{N_0}\,(-\mathrm{Z}_1\, sin\theta - Z_2\, cos\theta) \qquad \ldots (5.17)$$

and for the second differentiation is

$$\frac{\partial^2 \ln f_{z|\theta}}{\partial \theta^2} = \sqrt{2T}\,\frac{A}{N_0}\,(-\mathrm{Z}_1\, cos\theta - Z_2\, sin\theta) \qquad \ldots (5.18)$$

Comparing Eq. 5.17 and 5.18 will get

$$\mathrm{var}\{\dot{\theta}_{\mathrm{ML}}\,(\mathrm{Z})\} \geq \frac{1}{\sqrt{2\mathrm{T}}}\,\frac{N_0}{A}\,(E\{Z_1\}\cos\theta - E\{Z_2\}\sin\theta)^{-1} \qquad \ldots (5.19)$$

The expectations of Z_1 and Z_2 (i.e. 1st statistical moment) are

$$E\{Z_i\} = \int_0^T E\{y(t)\}\,\theta_i(t)dt$$

$$= \int_0^{\mathrm{T}} \sqrt{\frac{\mathrm{T}}{2}}\,A[\cos\theta\ \emptyset(t) - \sin\theta\ \emptyset_2\,(t)]\,\emptyset_i(t)dt$$

$$= \int_0^{\mathrm{T}} \sqrt{\frac{\mathrm{T}}{2}}\,A cos\,\theta,\ \ i = 1 \qquad \ldots (5.20)$$

where Eq.5.5 is used. Substitution of these results into Eq.5.19, yields

$$\mathrm{var}\{\dot{\theta}_{\mathrm{ML}}\,(\mathrm{Z})\} \geq \frac{1}{\sqrt{2\mathrm{T}}}\,\frac{\mathrm{N}_0}{\mathrm{A}}\left[\sqrt{\frac{\mathrm{T}}{2}}\,A\,(cos^2\theta + sin^2\,\theta\right]^{-1} = \frac{N_0}{A^2T} \qquad \ldots (5.21)$$

Noting that the average motion signal is $P_s = ½A^2$ and defining $B_L = (2T)^{-1}$ as the equivalent human motion bandwidth of the estimator structure, Eq. 5.21 can be written as

$$\mathrm{var}\{\dot{\theta}_{\mathrm{ML}}\} \geq \frac{N_0 B_L}{P_s} \qquad \ldots (5.22)$$

Eq.5.22 is identical to the result of the non-linearity in the estimator and additionally a lower bound for the variance is obtained.

5.4.2 Implementation

The performance of phase compensator integrated with human motion estimation filter is implemented successfully with the following steps:

Step 1: Generate two independent Gaussian random values 'Z_1' and 'Z_2' respectively.

Step 2: Estimate the human motion values 'θ'.

Step 3: Call the first iteration of motion 'θ_0'and estimate the next estimation and obtain 'θ_1'.

Step 4: Iteratively estimate the human motion using first iteration as:

$$\theta_1 = \theta_0 + \epsilon \tan^{-1}\left(\frac{Z_{2.0}}{Z_{1.0}}\right)$$

Step 5: Generate again two values of 'Z_1' and 'Z_2' (call it is '$Z_{1.1}$' and '$Z_{2.1}$') respectively.

Step 6: Then, estimate the next human motion as

$$\theta_2 = \theta_1 + \epsilon \tan^{-1}\left(\frac{Z_{2.1}}{Z_{1.1}}\right)$$

Step 7: Continue step 3 to step 6, till 'θ' converges to zero phase.

5.5 RECURSIVE IMPLEMENTATION OF DIFFERENT CLASSIFIERS

5.5.1 Bayesian Classifier (for Gaussian Processes)

Inputs:

(a) Mean vectors

(b) Covariance matrices of the class distributions of a c-class problem

(c) Apriori probabilities of the c classes, and

(d) a matrices X containing column vectors that stem from the above classes.

Output:

N-dimensional vector whose i^{th} component contains the class where the corresponding vector is assigned, according to the Bayesian classification rule.

Code:

```
function z=bayes_classifier (m,S,P,X)
    [1,c]=size(m);          %1=dimensionality, c=no. of classes
    [1,N]=size(X);          %N=no. of vectors
    for i=1:N
        for j=1:c
            t(j)=P(j)*comp_gauss_dens_val(m(:,j),…..
            S (:,:,j),X(:,i));
        end
        % Determining the maximum quantity Pi*p(x | wi)
        [num,z(i)]=max (t);
    End
```

5.5.2 Euclidean Classifier

Inputs:

(a) the mean vectors and

(b) a matrix X containing column vectors that stem from the above classes.

Output:

N-dimensional vector whose i^{th} component contains the class where the corresponding vector is assigned, according to the minimum Euclidean distance classifier

Code:

```
Function z=eucliden_classifier(m,X)
        t(j)=sqrt(()X(:,i)-m(:,j))'*(X(:,i)-m(:,j)));
      (Computation of Euclidean distance from all class representatives)
[num,z(i)]=min(t);
(Determination of the closest class mean)
```

5.6 CLASSIFICATION ERROR FOR THE CLASSIFIERS

The performance of the different classifiers illustrated in this thesis, is evaluated by computing the classification error defined as the percentage of the places where the two vectors i.e. one vector containing the class where the corresponding data vector belongs and the second vector containing the class where corresponding data vector is assigned from a certain classifier. The recursive computation of this measure is performed as follows:

Input:

(a) An N-dimensional vector, each component of which contains the class where the corresponding data vector belongs and

(b) A similar N-dimensional vector each component of which contains the class where corresponding data vector is assigned from a certain classifier.

Output:

Classification error of the classifier.

Code:

```
Function class_error=compute_error(y,y_est)
    [q.N]=size(y);          %N=no. of vectors
    C=max(y);               % Determining the number of classes
    Class_error=0;          % Counting the misclassified vectors
    For i=1:N
        If(y(i) ~= y_est(i))
            Clas_error=clas_error+1;
        End
    End
    %computing the classification error
    Class_error=clas_error/N;
```

5.7 CLASSIFICATION MODELS

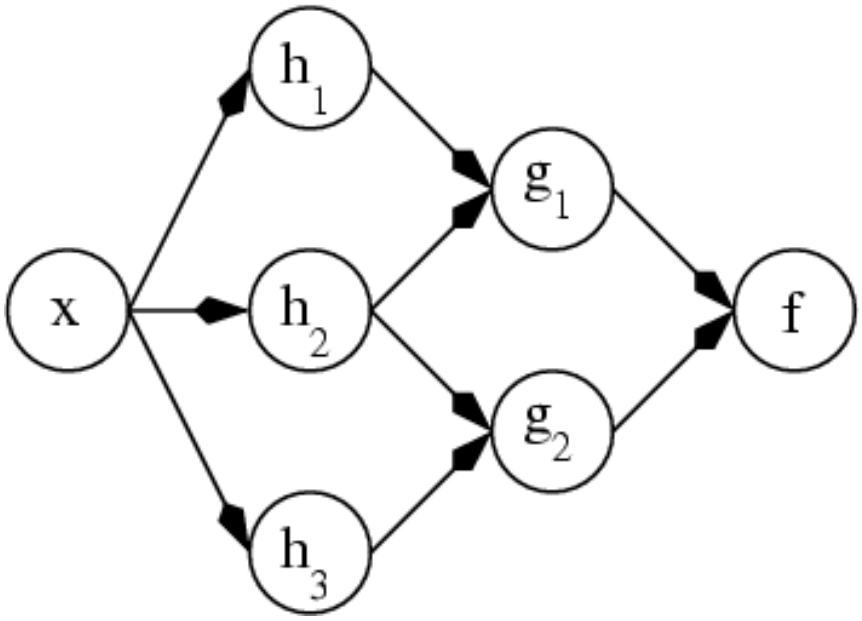
These are essentially simple mathematical models defining a function or a distribution. Sometimes models are also intimately associated with a particular learning algorithm or learning rule. A common use of the phrase neural network model really means the definition of a *class* of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity).

5.7.1 Network Function

Figure 5.10 depicts such decomposition, with dependencies between variables indicated by arrows. These can be interpreted in two ways. The first view is the functional view: the input is transformed into a 3-dimensional vector, which is then transformed into a 2-dimensional vector. This view is most commonly encountered in the context of optimization. The second view is the probabilistic view: the random variable depends upon the random variable, which depends upon, which depends upon the random variable. This view is most commonly encountered in the context of graphical models. The two views are largely equivalent.

In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of are independent of each other given their input). This naturally enables a degree of parallelism in the implementation.

http://en.wikipedia.org/wiki/File:Ann_dependency_graph.png



http://en.wikipedia.org/wiki/File:Ann_dependency_graph.png

Figure 5.10 Neural Network dependency graph

5.7.2 Design issues in Neural Network

Before a neural network is trained to learn a classification task, the following design issues must be considered.

1. The number of nodes in the input layer should be determined. Assign an input node to each numerical or binary input variable. If the input variable is categorical, either create one node for each categorical value or encode the K-ray variable using $[\log_2 k]$ input nodes.

2. The number of node in the output layer should be established. For two-class problem, it is sufficient to use single output node. For *k* - class problem, there are *k* output nodes.

3. The network topology (e.g., the number of hidden layers and hidden nodes, and feed-forward are recurrent network architecture) must be selected.

4. The target function representation depends on the weight of the links, the number of hidden nodes and hidden layers, biases in the nodes, and type of activation functions.

5. Alternatively, instead of repeating the model-building procedure, some of the nodes could be removed and depict the model evaluation procedure to select the right model complexity.

6. The weights and biases need to be initialized. Random assignment is usually acceptable.

7. Training examples with missing values should be removed or replaced with most likely value.

8. Neural Network can handle redundant features because the weights are automatically learned during the training step. The values of weights for redundant features tend to be very small.

5.7.3 Inference results from Neural Network

The error curve, neural network mapping curve and the convergence plots are shown in Figure 5.11 to Figure 5.13. Figure 5.11, the x-axis denotes the range of variations of parameter about its nominal value and in this work, the variation was chosen as x $\sum$ [-6, 6]. The results are shown for first 500 epochs in Figure 5.13. Figure 5.12 indicates that, the neural network is able to map with the required data with lesser error for most of the data points. However, when the data points are much closer to the nominal value, the network exhibits an increased error and therefore, the number of epochs must be increased.

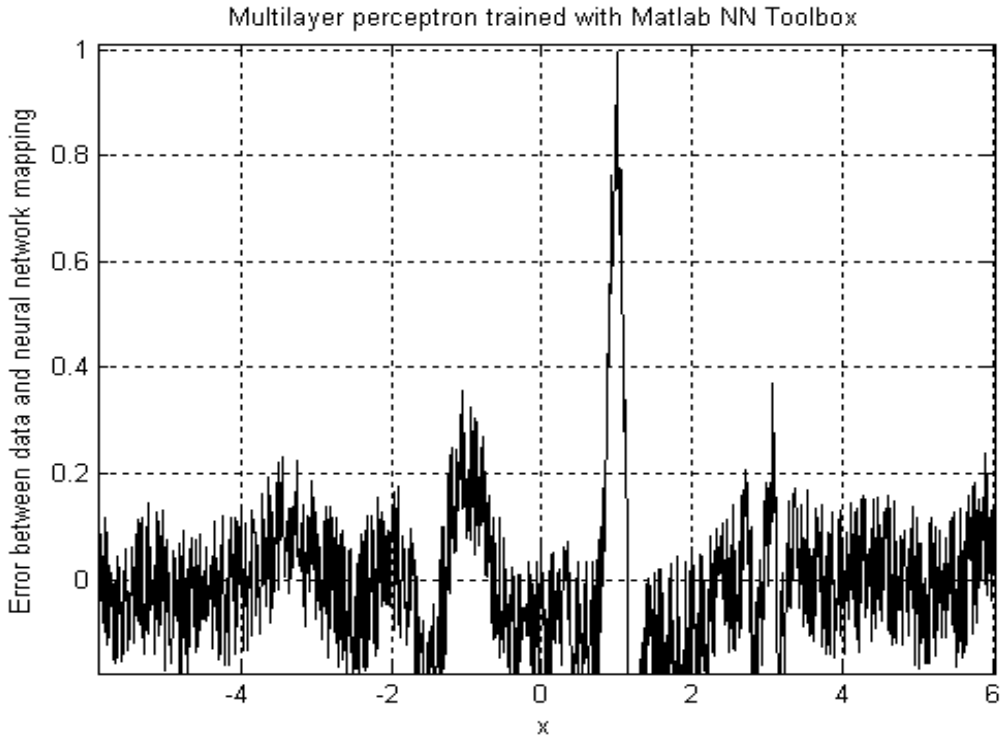


Figure 5.11 Error plot between target data and neural output

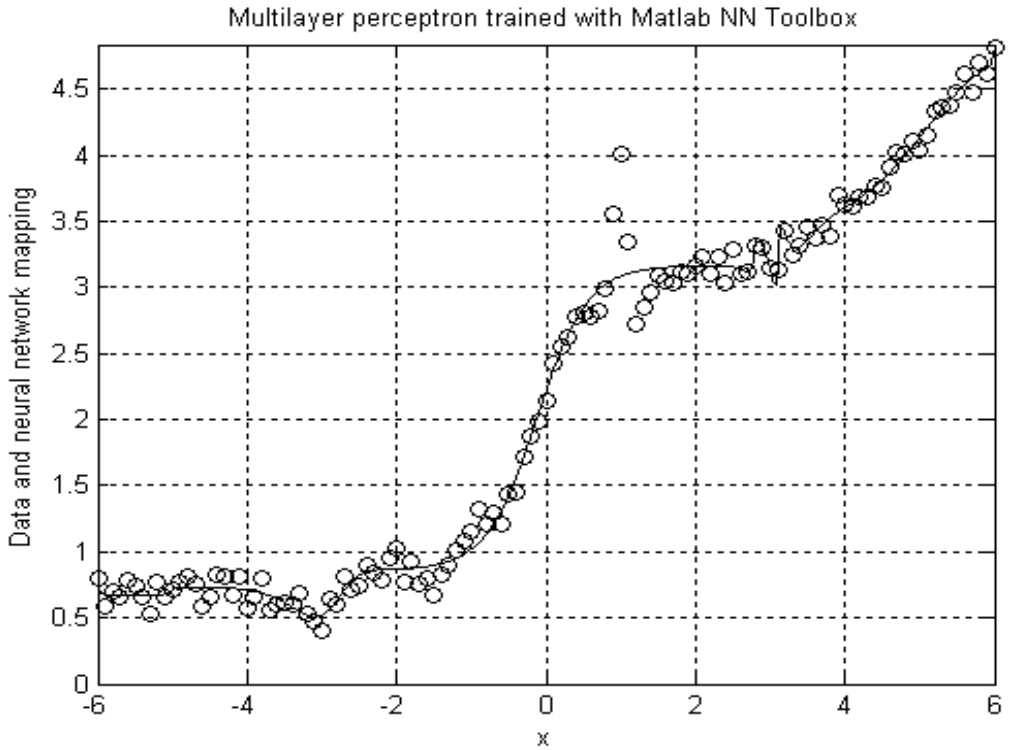


Figure 5.12 Mapped output points over the range of trained input data set

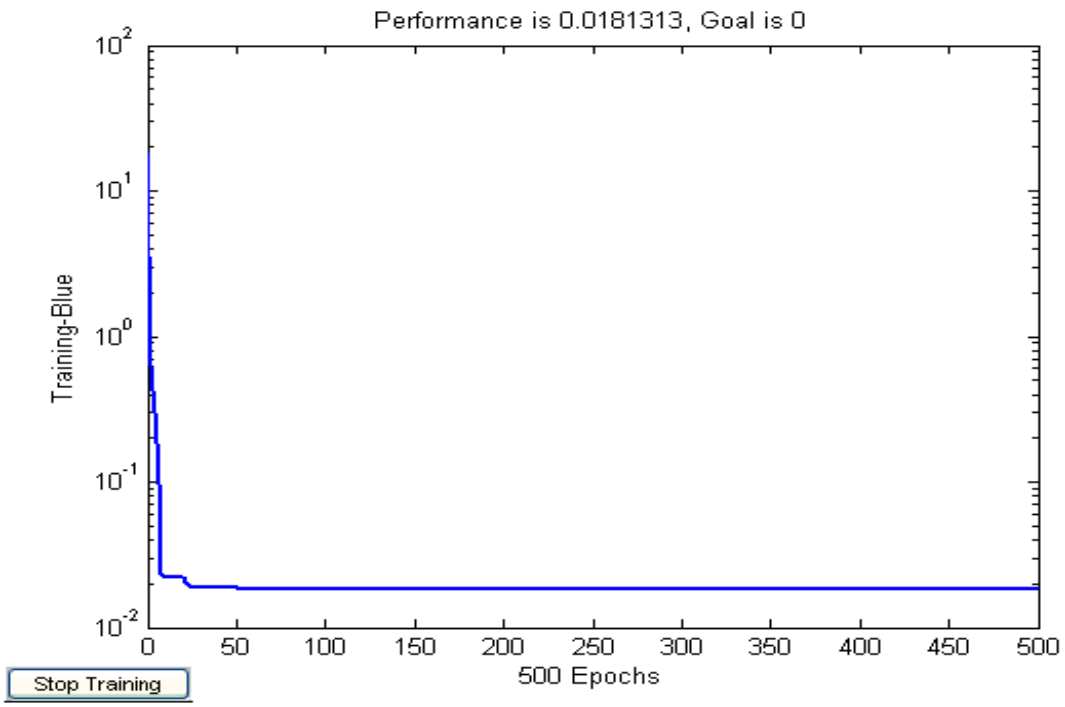


Figure 5.13 Training curve for the first 500 epochs

5.8 CLASSIFIERS WITH MINIMIZED ERROR PROBABILITY

a) The Bayesian classifier that minimizes the error probability is given in this case by minimizing Euclidean distance classifier. Thus, assign $x \in \omega_1$if

$$\|x - \mu_1\| < \|x - \mu_2\|$$

b) In this case *x* is classified t :o $\omega_1$if

$$\frac{p(x/\omega_1)}{p(x/\omega_2)} > \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21}}{\lambda_{12}}$$

where $\lambda_{12} = 1$ and $\lambda_{21} = 0.5$. Thus, following similar arguments as in theory, for Bayesian classification for normal distributions, the decision hyper plane is

$$g_{12}(x) = w^T(x - x_0)$$

$$w = \mu_1 - \mu_2$$

$$\text{and } x_0 = \frac{1}{2}(\mu_1 - \mu_2) - \sigma^2 \text{In}\ \frac{P(\omega_2)}{P(\omega_1)}\frac{\lambda_{21}}{\lambda_{12}}\frac{\mu_1 - \mu_2}{\|\mu_1 - \mu_2\|^2}$$

5.8.1 Recursive implementation of ML Estimate

A recursive implementation of ML (Machine Likelihood) estimate is presented in this section.

$$\hat{\mu}_{N+1} = \hat{\mu}_N + \frac{1}{N+1}(x_{N+1} - \hat{\mu}_N)$$

$$\hat{\sum}_{N+1} = \frac{N}{N+1}\hat{\sum}_N + \frac{N}{(N+1)^2}(x_{N+1} - \hat{\mu}_N)(x_{N+1} - \hat{\mu}_N)^T$$

where the subscript in the notation of the estimates, $\hat{\mu}_N, \hat{\sum}_N$ indicates the number of samples used for their computation. Since,

$$\hat{\mu}_{N+1} = \frac{1}{N+1}\sum_{k=1}^{N+1} x_k = \frac{1}{N+1}\sum_{k=1}^{N+1} x_k + \frac{1}{N+1}x_{N+1}$$

$$= \frac{1}{N+1}\hat{\mu}_N + \frac{1}{N+1}x_{N+1} = \hat{\mu}_N + \frac{1}{N+1}(x_{N+1} - \hat{\mu}_N)$$

For the covariance matrix,

$$\sum_{N+1} = \frac{1}{N+1}\sum_{k=1}^{N+1}(x_k - \hat{\mu}_{N+1})(x_{N+1} - \hat{\mu}_N)^T$$

$$= \frac{1}{N+1}\sum_{k=1}^{N+1} x_k x_k^T - \frac{1}{N+1}\hat{\mu}_{N+1}\sum_{k=1}^{N+1} x_k^T - \frac{1}{N+1}\left(\sum_{k=1}^{N+1} x_k\right)\hat{\mu}_{N+1}^T + \frac{1}{N+1}\sum_{k=1}^{N+1}\hat{\mu}_{N+1}\hat{\mu}_{N+1}^T$$

$$= \frac{1}{N+1}\left(\sum_{k=1}^{N+1} x_k x_k^T + x_{N+1}x_{N+1}^T\right) - \hat{\mu}_{N+1}\hat{\mu}_{N+1}^T$$

$$= \frac{1}{N+1}(N\sum_N \hat{\mu}_N\hat{\mu}_N^T + x_{N+1}x_{N+1}^T) - \hat{\mu}_{N+1}\hat{\mu}_{N+1}^T \quad \ldots (5.23)$$

$$\sum_{N+1} = \frac{N}{N+1}\sum_{N} + \frac{N}{N+1}\hat{\mu}_N\hat{\mu}_N^T + \frac{1}{N+1}x_{N+1}x_{N+1}^T - \frac{N^2}{N+1}\hat{\mu}_N\hat{\mu}_N^T - \frac{1}{(N+1)^2}x_{N+1}x_{N+1}^T - \frac{2N}{(N+1)^2}\hat{\mu}_N x_{N+1}^T$$

which results in

$$\sum_{N+1} = \frac{N}{N+1}\sum_{N} + \frac{N}{N+1}(x_{N+1} - \hat{\mu}_N)(x_{N+1} - \hat{\mu}_N)^T \quad \ldots (5.24)$$

5.8.2 Recursive implementation of posterior pdf

The posterior pdf (Probability Density Function) estimate in the Bayesian inference task can be computed recursively, i.e.

$$p(\theta|x_1, \ldots, x_N) = \frac{p(x_N|\theta)p(\theta|x_1,\ldots,x_{N-1})}{p(x_N|x_1,\ldots,x_N)} \quad \ldots (5.25)$$

The intrinsic recursive nature of the Bayesian inference task is revealed. From theory,

$$p(\theta|x_1, \ldots, x_N) = \frac{p(x_1,\ldots,x_N|\theta)p(\theta)}{p(x_1,\ldots,x_N)} \quad \ldots (5.26)$$

Using the Bayes theorem

$$p(\theta|x_1, \ldots, x_N) = \frac{p(x_1, \ldots, x_{N-1}, \theta)p(x_1, \ldots, x_{N-1}|\theta)p(\theta)}{p(x_1, \ldots, x_N)}$$

$$= \frac{p(x_1, \ldots, x_{N-1}, \theta)p(\theta|x_1, \ldots, x_{N-1})p(x_1, \ldots, x_{N-1})}{p(x_N|x_1, \ldots, x_{N-1})p(x_1, \ldots, x_{N-1})}$$

Taking into account the mutual independence of $x_1, \ldots, x_N$,

$$p(\theta|x_1, \ldots, x_N) = \frac{p(x_N|\theta)p(x_1,\ldots,x_{N-1})}{p(x_N|x_1,\ldots,x_{N-1})} \quad \ldots (5.27)$$

5.8.3 Approximation of classifiers using Gaussian functions

In this work, the approximation of classifiers is done using Gaussian kernel function. The closeness of the approximation is studied by varying the smoothing parameter. The code is shown as follows:

Input:

Smoothing parameter h [varies between 0.05 and 0.2].

N: the no. of points generated from a pseudorandom generator according to $p(x)$.

Output:

Plot of the approximation varied w.r.t. (N,h)

Code:

```
n=rand(1,N)*2;
Privy=0;
p=[];
for m=-0.5:h:2.5
newp=0;
for a=1:size(n,2)
X=(n(a)-m)/h;
newp=newp+(1/sqrt(2*pi))*exp(-(x)^2/2);
end
newp=(1/h)*(1/size(n,2))*newp;
p=[p newp];
end
x=[zeros(1,length(-0.5:0.01:-0.01)),ones(1,length(0:0.01:-0.01)),
ones (1, length (0:0.01:2))*0.5, zeros (1, length (2.01:0.01:2.5))];
xax=-0.5:0.01:2.5;
m=-0.5: h:2.5;
plot (m,p,'r',xax,'k');
```

The output obtained by executing the above code snippet for different values of smoothing parameter 'h' and 'N' is shown in Table 5.1.

Table 5.1 Approximated pdf for ‘h’ and ‘N’

Approximated pdf for ‘h’	Approximated pdf for ‘N’
0.02	32
0.02	256
0.02	5000
0.2	32
0.2	256
0.2	5000

5.9 CHAPTER SUMMARY

In this thesis chapter, a human motion estimation filter is designed and performance metrics were evaluated. Unsupervised network architecture was used to reduce the artifacts from the motion input and extract the human motion.