The application of neural networks to problems in fringe analysis.

David John Tipper

A thesis submitted in partial fulfilment of the requirements of the Liverpool John Moores University for the degree of Doctor of Philosophy.

July 1999

School of Engineering,
Liverpool John Moores University

Dedicated to the memory of my grandfather,
Harold Smith.

Application of Neural Networks to Problems in Fringe Analysis
Abstract of Thesis

This thesis describes the use of neural networks to address two problems which occur during the process of fringe analysis.

Phase Unwrapping

Due to the phase unwrapping problem being essentially one of recognition (i.e. What is a phase wrap and what is noise?), it was thought that a neural network would be ideally suited to the task of recognising the position of phase wraps in an image. Initial experimentation involved the use of small networks, typically containing less than 20 neurons for the unwrapping of simple phase distributions in one dimension. It was shown that backpropagation neural networks were capable of distinguishing phase wraps from noise spikes, so the idea was extended to use larger networks to process two dimensional "tiles" for the unwrapping of entire images. Experimentation with both supervised and unsupervised learning was carried out and the results showed that, again, backpropagation networks proved to be the most reliable.

It was successfully shown that a backpropagation neural network can form the basis of a reliable and robust phase unwrapping system.

Fringe optimisation.

Little work has been carried out in the field of optimisation of fringe patterns, as the process was largely impossible until the invention of the adaptive interferometer. The interferometer used twin optical fibres to produce a fringe pattern. If the relative position of the fibres is varied, this can vary characteristics of the fringe pattern, namely fringe spacing and orientation. The use of neural networks to optimise a fringe pattern before analysis takes place has been investigated. If the fringe pattern is optimised before measurement takes place, the suitability of that pattern for any given surface will be ensured. Neural networks were trained to analyse the parameters which are easily controllable, i.e. mean intensity, visibility, fringe number and fringe orientation. Two methods were investigated:

(a) The use of a separate network for each parameter, the outputs from each one being combined to produce a final decision.

(b) The use of a single network to analyse the pattern "globally".

Again, experiments were carried out using both supervised and unsupervised learning. The most accurate results were achieved using a General Regression Network for global analysis. The final result was a "closed loop" system, whereby the fringe pattern could be successfully optimised before mathematical analysis took place.

Acknowledgements

The author would like to thank his supervisors, Prof. David Burton and Prof. Michael Lalor for their support and guidance throughout this project. Also, the author would like to take this opportunity to mention his third supervisor, the late Dr. John Atkinson.

This work would not have reached completion without the unofficial help of the author's friends and colleagues of the Coherent and Electro-Optics Research Group, the Centre for Precision Measurement and the basement laboratories at the John Moores University. He would, therefore, like to thank them for making the impossible seem a little easier.

Contents

Chapter 1: Introduction

Chapter 1: Introduction.

Metrology is defined simply as "the science of measuring"[1]. While not a new science, its use is becoming more widespread in modern industry. The accurate measurement of engineering components is becoming increasingly important. In today's marketplace, companies are under pressure to produce better and more reliable components in order to have the advantage over their competitors. To be successful, manufacturers need to ensure that products comply with their specifications, which, as manufacturing technology improves, are becoming increasingly exacting. Modern designs are extremely complex and parts are manufactured to high tolerances. For example, the design of the combustion chamber within a motor vehicle cylinder head is far more involved than, say, thirty years ago. In the past, the combustion chamber was designed simply as a place for fuel to be compressed and ignited. Today, with ever-increasing demands for performance and fuel economy, the way in which the fuel-air mixture burns is important, so combustion chambers are designed to optimise the path of the flame-front during ignition. To achieve this, the curvature of the chamber must be measured extremely accurately. A number of methods exist which can accomplish this task, including both contact and non-contact techniques. This thesis is primarily concerned with a non-contact method of measurement, which uses some form of “structured light”, usually a pattern of interference fringes, to generate the information about an engineering component. The extraction of this information is often referred to as *fringe analysis*.

Fringe analysis has been a recognised technique for many years. For example, it enables surface profiles of objects to be measured with extreme accuracy without use of intrusive, tactile probes that may themselves affect the result of the measurement process. Chapter 2 describes in detail the process of fringe generation. Whichever method of fringe generation is

used, e.g. holographic, moiré, speckle etc., the net result is always the same. A set of fringes is produced, usually referred to as interferograms. Until the digital computer became a widely available tool, fringe analysis was an extremely labour intensive task. The process generally involved manual location of the fringes in an image and numbering of their position. This changed, however, as the cost of computers decreased and their power increased. Much effort has been made in making fringe analysis a viable metrological process by increasing its accuracy and speed. This thesis attempts to address two of the problems that arise during the process of fringe analysis. Chapter 4 introduces a novel approach to the problem of phase unwrapping. Whether the Fourier transform or the phase stepping method is used, the mathematics involved in the analysis lead to a distribution of phase values that are "wrapped" modulo 2π. This is explained in detail in chapter 4. In order to produce an accurate height map of the measured surface, the phase map must be reconstructed to give a continuous distribution. It is this stage of the process that is usually the most difficult to address. A number of algorithms have been developed to solve the phase unwrapping problem. These are discussed in detail in chapter 4. Even though the subject has been researched extensively, there is still no "generic" phase unwrapper. Most algorithms are not robust enough to cope with complex, noisy images. Some of the simpler algorithms have difficulty in differentiating between noise and genuine phase wraps, the result being errors that are propagated throughout the final unwrapped phase distribution. It was thought that a new approach to the phase unwrapping problem was needed, so this thesis describes a method using neural networks, a method into which little research has been carried out in this application.

The thesis also outlines an approach to a further problem: that of fringe optimisation. Until recently, it was not possible to have great control over the quality of the projected fringe pattern. With the development of the adaptive interferometer, which is described in detail in chapter 2, it has

become possible to have a greater degree of control over such factors as fringe spacing and orientation. With this adaptability comes the prospect of optimising the fringe pattern to ensure a good result will be achieved *before* the measurement process takes place. Again, this is a subject into which little research has been carried out. The second part of this thesis describes an approach that utilises a neural network to analyse the quality of a fringe pattern to ascertain whether it conforms to a series of optimisation criteria.

References:

1. Chambers Scientific and Technical Dictionary, 1996

Chapter 2: Fringe analysis techniques

Chapter 2: Fringe analysis techniques

2.1 Interference

The phenomenon of interference is an important basis to the science of fringe analysis. Consider two waves that can be described

$$u_1 = U_1 e^{i\phi_1}$$

and

$$u_2 = U_2 e^{i\phi_2}$$

where u = spatial complex amplitude and

U = amplitude

ϕ = phase.

When these two waves overlap each other in space, electromagnetic wave theory states that the resulting field is given by the summation of the complex amplitude terms, thus:

$$u = u_1 + u_2$$

The observable quantity is, however, not amplitude but *intensity*, which is given by

$$I = |u|^2$$

The observed intensity of the overlapping waves is, therefore, given by

$$I = |u_1 + u_2|^2$$

$$= U_1^{\,2} + U_2^{\,2} + 2U_1U_2\cos(\phi_1 - \phi_2)$$

$$= I_1 + I_2 + 2\sqrt{(I_1 I_2)}\cos\Delta\phi$$

where

$$\Delta\phi = \phi_1 - \phi_2$$

The two waves interfere and $\Delta\phi = \phi_1 - \phi_2$ is referred to as the *interference term*. Also, when

$$\Delta\phi = (2n + 1)\pi \qquad \text{for } n = 0, 1, 2,...$$

$\cos\Delta\phi = -1$ and I reaches its minimum. In this case, the two waves interfere destructively as they are in *antiphase.* The two waves interfere constructively when they are in phase. This occurs when

$$\Delta\phi = 2n\pi \qquad \text{for } n = 0, 1, 2,...$$

when $\cos\Delta\phi = 1$ and I reaches its maximum.

For two waves of equal intensity, i.e. where $I_1 = I_2 = I_0$ the intensity equation becomes

$$I = 2I_o(1 + \cos\Delta\phi)$$

$$= 4I_o\cos^2\left(\frac{\Delta\phi}{2}\right)$$

where the intensity varies between 0 and $4I_0$.

The result of this interference is a fringe pattern of the type shown in figure 2.1.

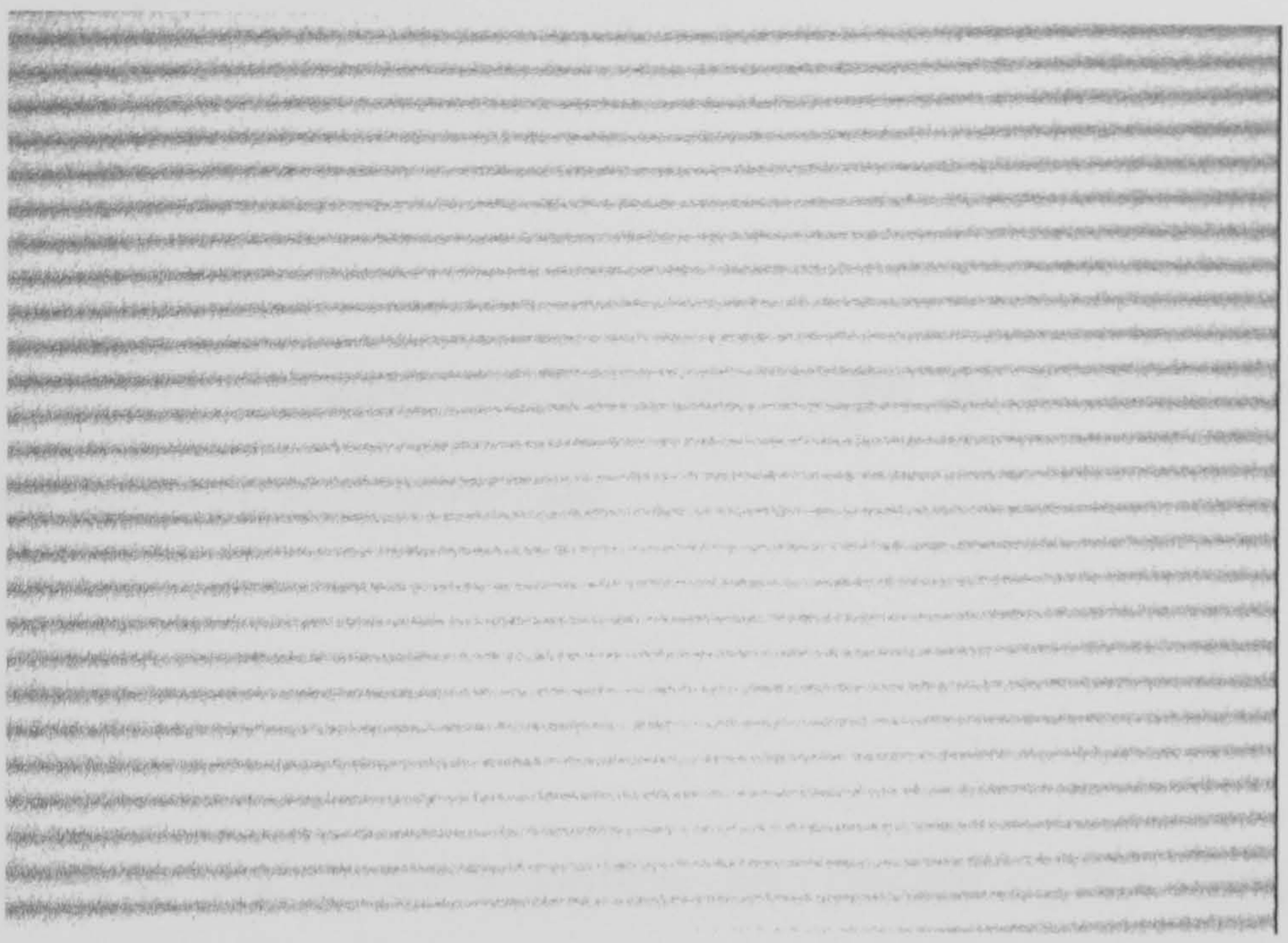


Figure 2.1: Cosinusoidal interference fringe pattern

A fringe pattern such as this can be a useful measurement tool. Consider the structured light system as shown in figure 2.2.

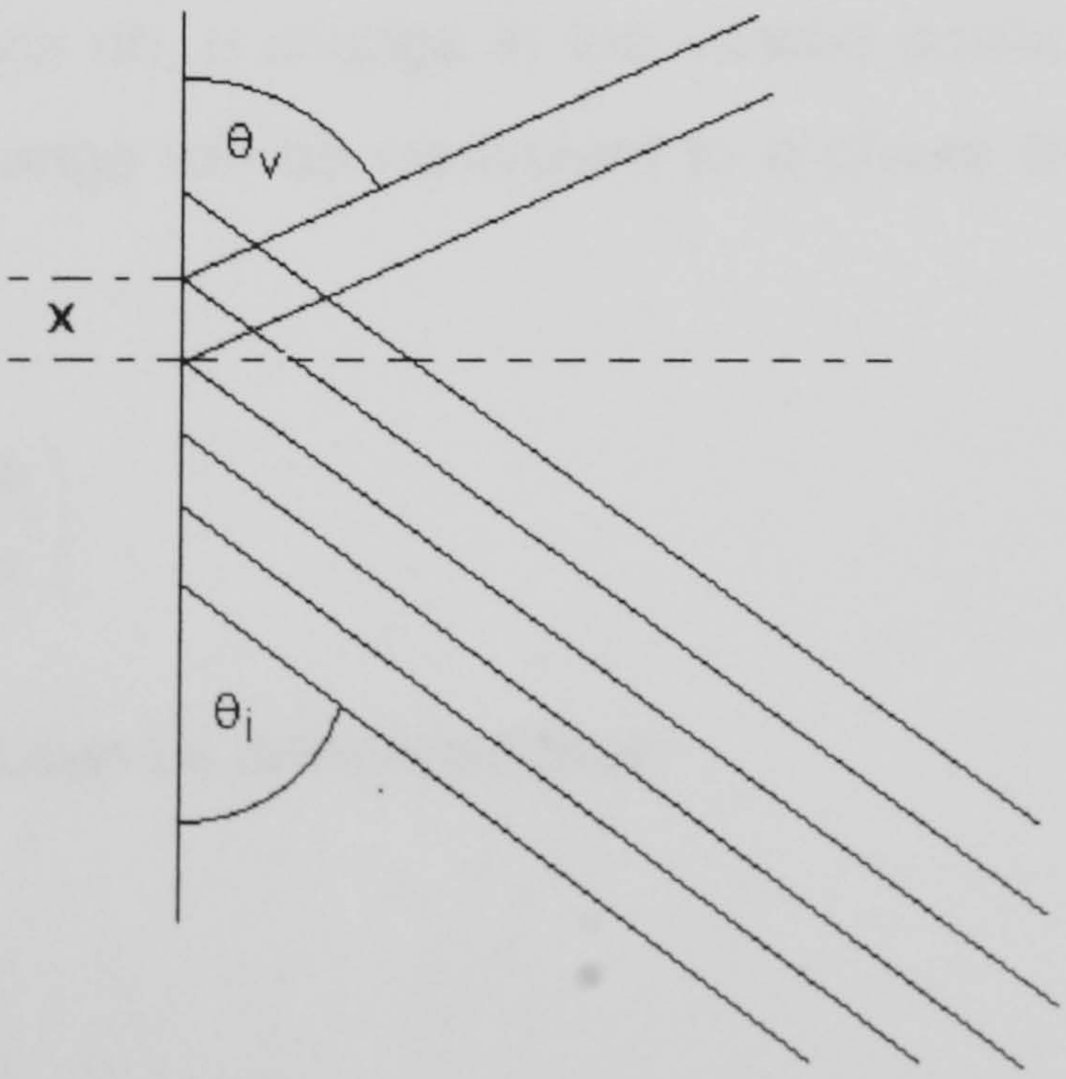


Figure 2.2: Structured light system

The plane surface is illuminated by a fringe pattern whose intensity profile is sinusoidal, the fringes having a period x. If the angle of illumination is θ_i, the fringes on the surface will have a period, y, given by

$$y = \frac{x}{\cos\theta_i}$$

If the fringe pattern is observed with an angle of θ_v, the period of the observed fringe pattern, z, is given by

$$z = y \cos \theta_v$$

Substituting for y in the above equation, the period of the observed fringe pattern can be described by the equation

$$z = \frac{x \cos\phi_v}{\cos\theta_i}$$

Consider the situation shown in figure 2.3. If the position of the surface moves a distance dh, a change in the viewed position of the light, dz, will occur. This change will be equivalent to a phase shift of $d\phi$. This phase shift is given by

$$d\phi = 2\pi\left(\frac{dz}{z}\right)$$

The distance dz can be calculated thus:

$$a = dh \tan \theta_i$$

$$b = dh \tan \theta_v$$

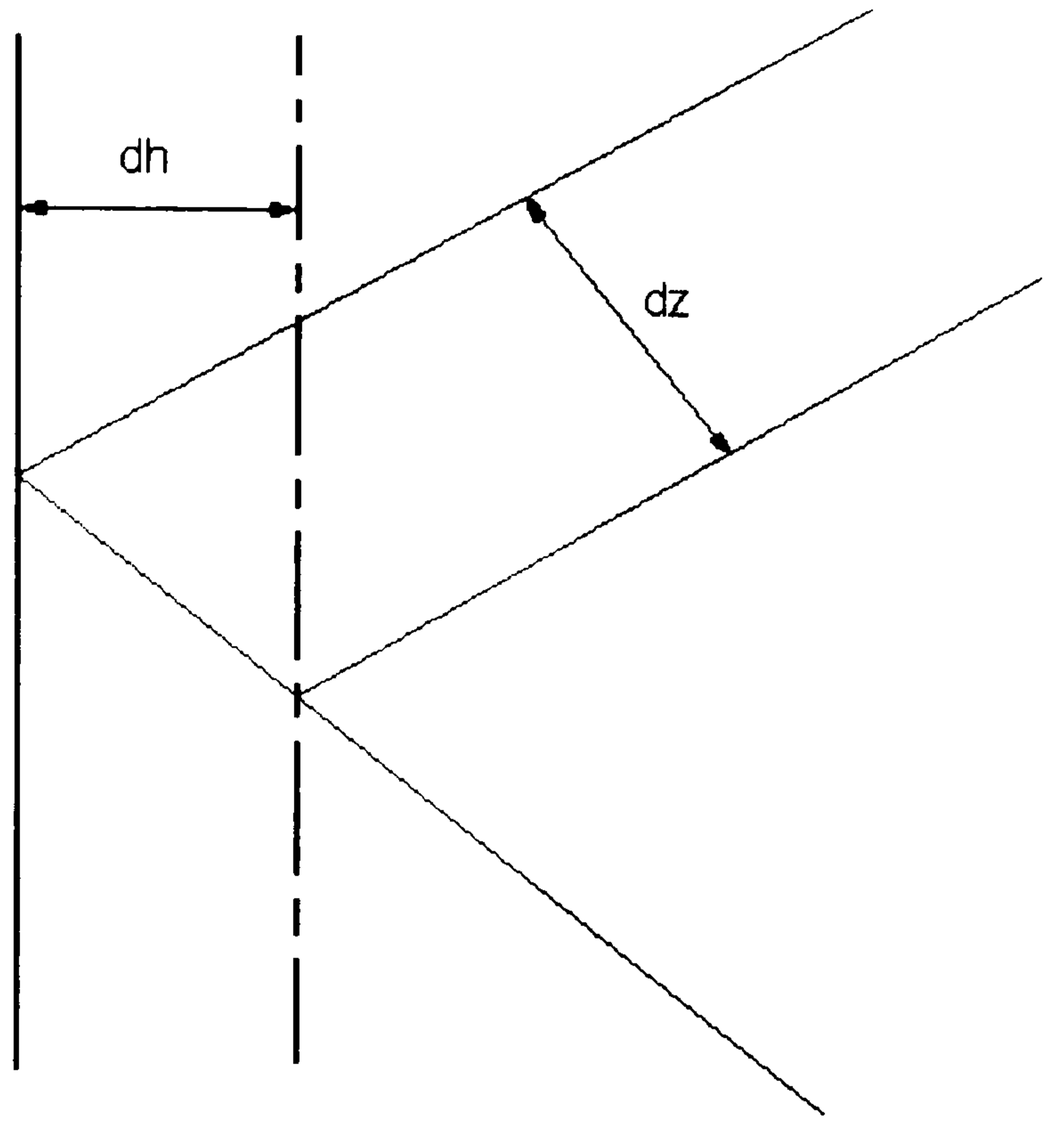


Figure 2.3

Adding the above equations gives

$$a + b = dh(\tan\theta_i + \tan\theta_v)$$

However,

$$dz = (a + b)\cos\theta_v$$

$$= dh(\tan\theta_i + \tan\theta_v)\cos\theta_v$$

Substituting gives

$$d\phi = \frac{(2\pi\ dh(\tan\theta_i + \tan\theta_v)\cos\theta_v)}{x}$$

If the image is viewed with the lens normal to the plane of the surface, θ_v becomes zero, reducing the above equation to

$$d\phi = \frac{(2\pi\ dh \sin\theta_i)}{x}$$

This shows that the height at any point on the surface of an object is proportional to the phase of the structured light reflected from the object. It is this principle which makes the science of fringe analysis possible.

There are a number of ways in which a fringe pattern can be generated.

2.2 Interferometry

Interferometers are measurement instruments in which interference can be observed and generally consist of

- a light source
- an element for splitting the light into two or more partial waves
- separate propagation paths for each of the partial waves
- an element for superposing the partial waves
- a detector to observe the interference pattern

Generally, interferometers are divided into two groups, depending on how they split the light.

2.2.1 Wavefront division

There are a number of wavefront dividing interferometers, including the Fresnel biprism, Lloyd's mirror and Michelson's stellar interferometer. Undoubtedly the best known of this type of interferometer is the apparatus

used by Young in the earliest of interference experiments. In 1801, Young experimented with the system shown schematically in figure 2.4(a). The wavefront incident on the screen S1 is divided by the two small holes P1 and P2. Spherical wavefronts emerge from the holes and will interfere, with the interference being observed on the second screen, S2. The path length difference s of the light reaching a point x can be calculated as shown in figure 2.4(b). If the distance z is much greater than distance D, the distance s can by approximated by

$$s = \frac{Dx}{z}$$

The phase difference then becomes

$$\Delta\phi = \frac{2\pi}{\lambda} s$$

$$= \frac{2\pi\ D}{\lambda\ z} x$$

If this is inserted into the general expression for the resulting intensity distribution, the intensity becomes

$$I(x) = 2I\left(1 + \cos\left(^{2}\pi\ \frac{D}{\lambda\ z} x\right)\right)$$

Interference fringes occur which are parallel to the y axis with a spatial period of

$$\frac{\lambda\ z}{D}$$

which decreases as the distance D increases. This explanation assumes that the ideal case holds, where the light waves from P_1 and P_2 are coherent.

However, this case becomes more difficult to fulfil as D increases. The contrast of the fringes on S1 is a measure of the degree of coherence.

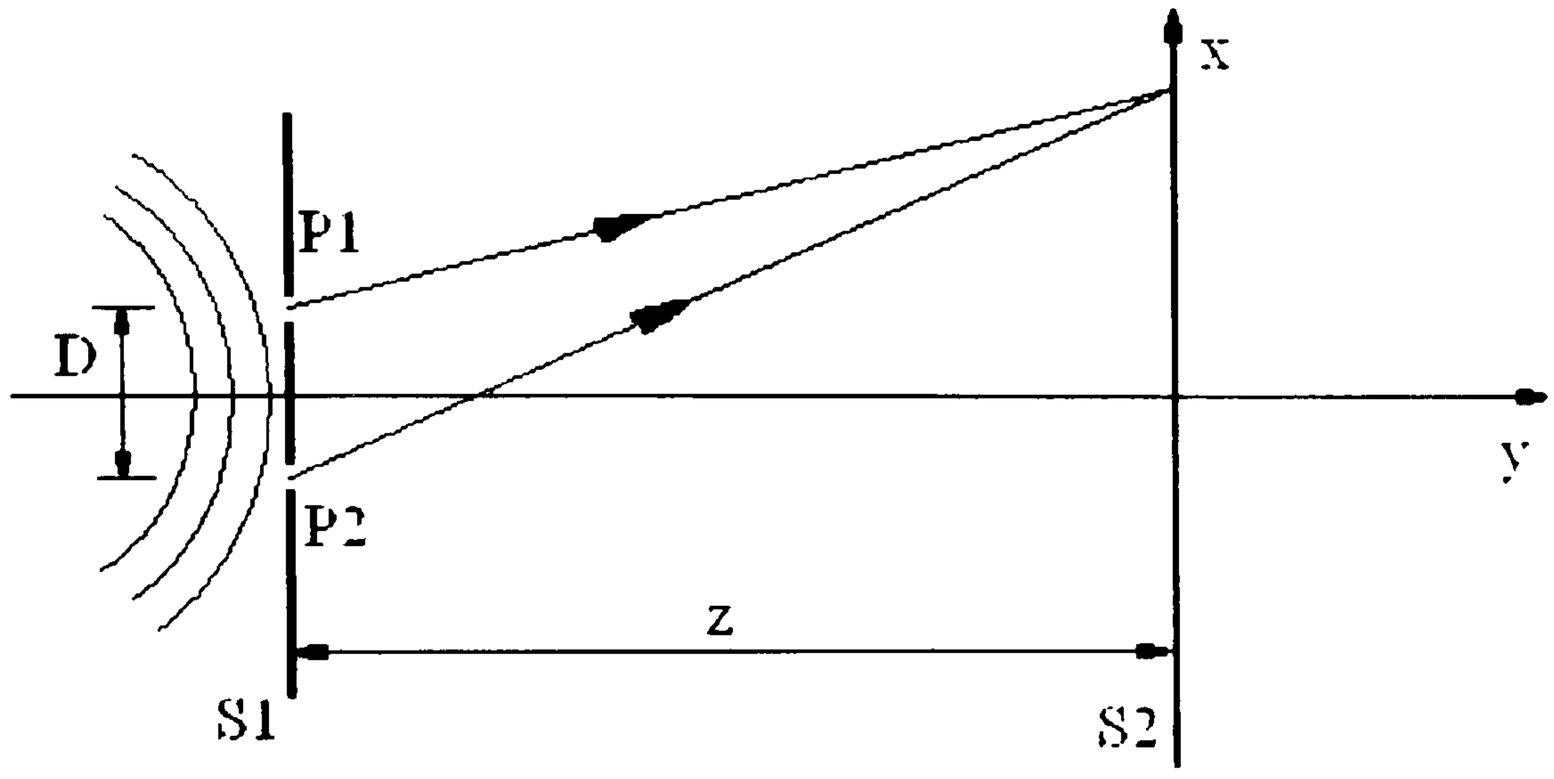


Figure 2.4(a): Young's experiment

Subsequently, the pinholes were replaced by slits to produce straighter fringes.

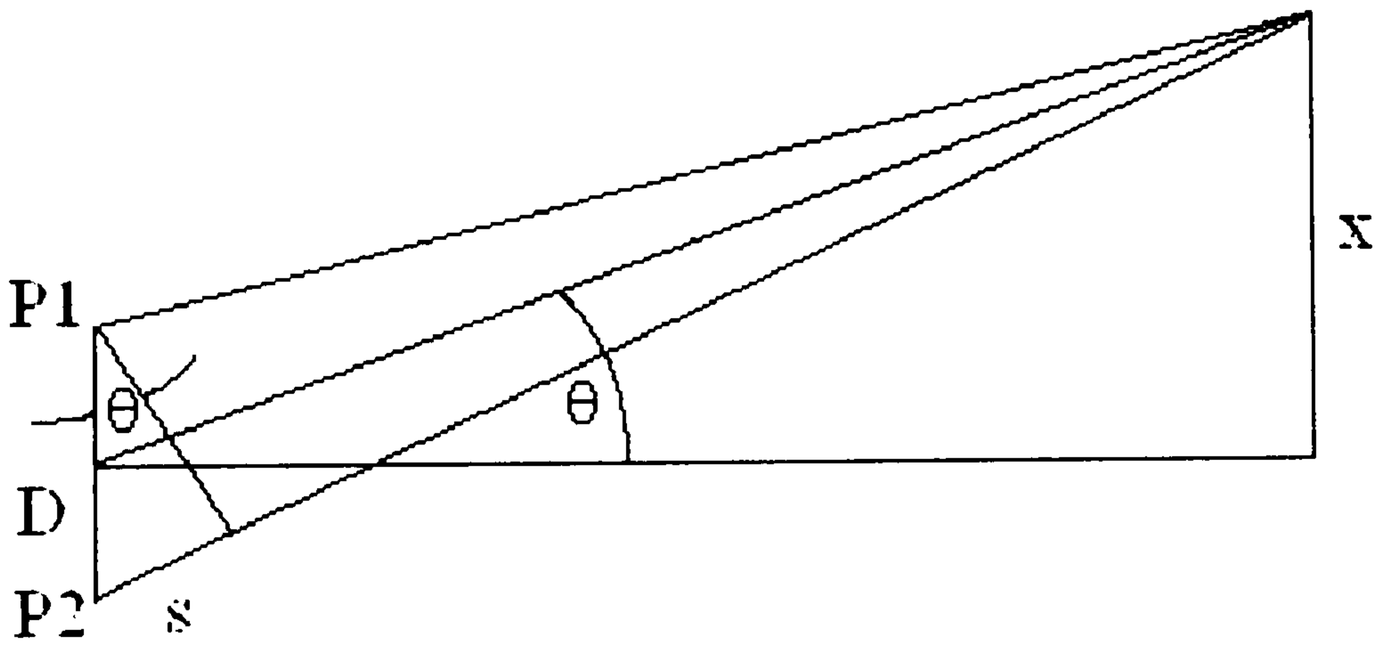


Figure 2.4(b)

2.2.2 Amplitude division

Here, the amplitude of the incident light is divided. This is usually done with the aid of a *beam splitter*. This is normally a transparent plate coated with a partially reflecting film, allowing part of the light to be transmitted and part to

be reflected. A commonly used amplitude division instrument is the Michelson interferometer, a schematic diagram of which is shown in figure 2.5. The light passes through the beamsplitter and the reflected and transmitted partial waves propagate to the mirrors M1 and M2. From here they are reflected back and recombined to form an interference fringe pattern at the detector D. The path difference between the two partial waves can be varied by moving one of the mirrors. Movement of a mirror through a distance x gives a path difference of 2x and a phase difference of

$$\Delta\phi = \left(\frac{2\pi}{I}\right)2x$$

The resultant intensity distribution is given by

$$I(x) = 2I\left(1+\cos\left(\frac{4\pi\, x}{\lambda}\right)\right)$$

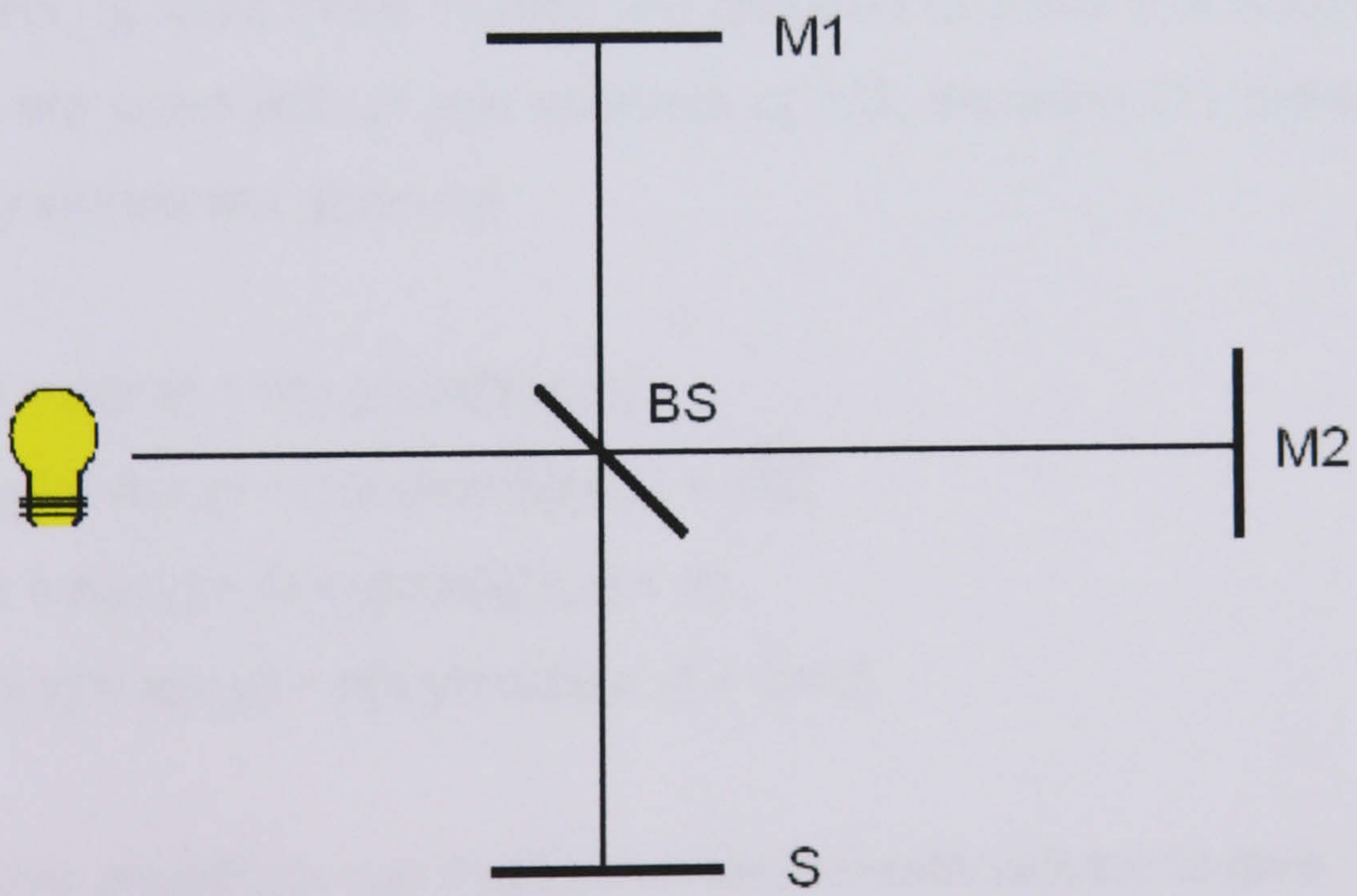


Figure 2.5: The Michelson Interferometer

2.3 Fringe analysis techniques

For many years, the analysis of interferograms was a laborious job. Until fairly recently, analysis was very much a labour-intensive operation, whereby an operator would manually number fringes and locate positions. However, as the power of computers and image processing equipment has increased and its cost decreased, automatic fringe analysis has become a major area for research.

There are two generally recognised techniques for the mathematical analysis of fringe patterns; phase stepping and Fourier fringe analysis.

2.3.1 Phase stepping

Phase stepping relies on the use of multiple fringe images. Intensity values of several images are recorded where the phase value of each one is shifted by a known amount. Because the intensity equation (1) contains three unknowns, at least three images are required to solve that equation. If four images are used with phase intervals of $\pi/2$, equation (1) dictates that the intensity values are given by:

$$I(0)(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y)]$$

$$I(\pi/2)(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y) + \pi/2]$$

$$I(\pi)(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y) + \pi]$$

$$I(3\pi/2)(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y) + 3\pi/2]$$

The above equations can then be solved simultaneously to give

$$\tan\phi(x,y) = \frac{I\left(\frac{\pi}{2}\right) - I\left(\frac{3\pi}{2}\right)}{I(0) - I(\pi)}$$

The phase, ϕ, at any point (x,y) is, therefore, given by

$$\phi(x,y) = \arctan\left(\frac{I\left(\pi_{/2}\right) - I\left(3\pi/2\right)}{I(0) - I(\pi)}\right)$$

Recently, further developments have been made in this field, utilising 5, 6 and 7 step algorithms[1].

2.3.2 Fourier fringe analysis

The use of the Fourier transform for the analysis of fringe patterns was pioneered in the early 1980s by Takeda et. al. [2] and requires only one image. A number of steps are involved in the calculation, thus:

1. An image of a fringe pattern is taken.
2. A two-dimensional Fourier transform is performed on the image.
3. The result is filtered in the Fourier plane to remove the d.c. term (a(x,y)), which eventually allows the term b(x,y) to be eliminated.
4. The remaining peak may be shifted to the origin. This step can be omitted, as discussed by Burton *et. al.* [3].
5. An inverse Fourier transform is performed on the data.
6. The inverse transform yields a phase distribution, which is wrapped modulo 2π. This step is explained in detail in chapter 4.
7. The phase map is unwrapped to give a continuous phase distribution.
8. The unwrapped phase values are proportional to height.

Mathematically, the process can be described thus:

A fringe pattern, as explained earlier in this chapter, can be described by

$$I(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y)]$$

where $I(x,y)$ = intensity at a point (x,y),

$a(x,y)$ is the dc offset term, caused by effects such as background illumination,

$b(x,y)$ = signal amplitude and

$\phi(x,y)$ = signal phase.

2.4 Problems encountered in fringe analysis

Although the phenomenon of interference fringes is one which has been long recognised, the use of automatic fringe analysis as a metrology tool is a more recent development. Its development as a science has only been feasible since the development of the modern digital computer. This is due to the complexity and scale of the calculations involved.

Even though much research has been done in the past fifteen years into the subject, a number of problems still recur at points during the analysis of fringe patterns. Two such problems are the subject of this thesis: phase unwrapping and fringe optimisation. In spite of this research in the field of fringe pattern analysis, the two problems addressed here do not have a generic solution.

Phase unwrapping still continues to be the most complex part of the fringe analysis process. As shown in the equations above, both Fourier techniques and phase stepping cause the value of phase to be calculated as a tangent function. As the calculation of height is dependent on phase, it is desirable to compute the inverse of the tangent function to retrieve the necessary

phase values. Due to the nature of the tangent function, as shown in figure 2.4, if its inverse is calculated, the resulting values will be "wrapped" between the values of $-\pi$ and $+\pi$. This means that, instead of achieving a smooth phase distribution, the result will resemble a "sawtooth" waveform, with values lying only in the region of $-\pi$ and $+\pi$. It is, therefore, necessary to "unwrap" the resulting phase distribution, that is remove the 2π phase discontinuities. Due to the amount of noise that is invariably present in a wrapped phase distribution, the achievement of a correctly unwrapped phase map can be problematic. This thesis attempts to outline a novel approach to the problem that deals with excessive amounts of noise in a phase distribution and still achieves an acceptable result.

Fringe optimisation is a relatively new field of research. Since the development of the adaptive interferometer, it has been possible to have automatic control of various parameters of a projected fringe pattern and, therefore, control its suitability for measurement of a given surface. The approach to the problem outlined in this thesis involves a method whereby an interference fringe pattern is analysed to ascertain its quality before measurement of an object takes place. The information gained from the analysis is then fed back to the machine controlling the fringe pattern to automatically adjust that pattern until it is seen as suitable for the measurement required. The problems of phase unwrapping and fringe optimisation are discussed in detail in chapters 4 and 5 respectively.

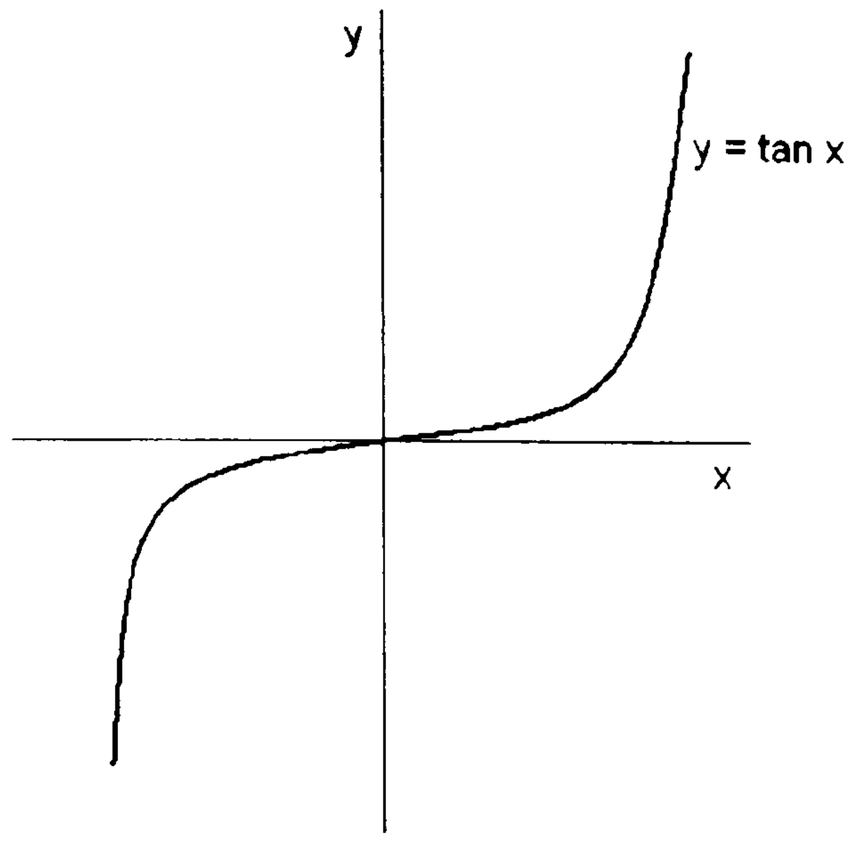


Figure 2.6: The tangent function.

References.

1. P.Hariharan; "Basics of Interferometry", Academic Press (1992)
2. M.Takeda, H.Ina and K.Kobayashi; "Fourier transform method of fringe pattern analysis for computer based topography and interferometry." J. Opt. Soc. Am. 72(1982)1
3. D.R.Burton and M.J.Lalor; "Multichannel Fourier fringe analysis for 3-D surface measurement" Proc. FRINGE '93, Akedemie Verlag (1993)37-44

Chapter3: Neural networks

Chapter 3: Neural Networks.

3.1 Introduction

Since the construction of the first digital computer shortly after the Second World War, the field of computing has developed at a phenomenal rate. It is said that the average Personal Computer of 1999 has more computing power than was available in the entire world in 1946. Today, computers are an integral part of everyday life. From microwave ovens to power stations, the list of computer controlled systems appears endless. This apparent level of power seems to have given the modern digital computer some kind of mystique. While there are some latter-day luddites who view the "computer age" with degrees of scepticism, many see the digital computer as some kind of panacea that is capable of solving mankind's problems with ease. The reality is, however, somewhat different. While it is true that modern machines are indeed powerful, they are far from the hyper-intelligent creatures that science fiction writers would have one believe. It is fair to say that the digital computer is exceptionally good at simple mathematics. Its strength lies in the fact that the average processor can perform millions of calculations per second. In comparison, the human brain is extremely slow. The switching speed of a neuron is approximately one million times slower than that of a computer gate [1]. However, with their capacity for parallel processing, humans are far more efficient at more complex tasks such as pattern recognition and speech understanding. Hence the neural network was conceived in attempt to create machines which mimic the massively parallel action of the computer known as the human brain.

The concept of the neural network originated in the early 1940s, when McCulloch and Pitts first attempted to model the operation of the biological neuron [2]. Figure 3.1 shows a schematic diagram of a biological neuron. Signals from other neurons are delivered to it by a collection of axons. Unless the collective influence of all its inputs reaches a threshold level, the neuron remains dormant. When this threshold level is reached, the neuron

delivers an output in the form of an electrochemical pulse. The pulse proceeds from the cell body, down the axon and into its branches, where the dendrites of the following neuron are influenced over narrow gaps known as synapses. When this process occurs, the neuron is said to *fire*. Simulated neurons model this behaviour by acting as thresholding devices.

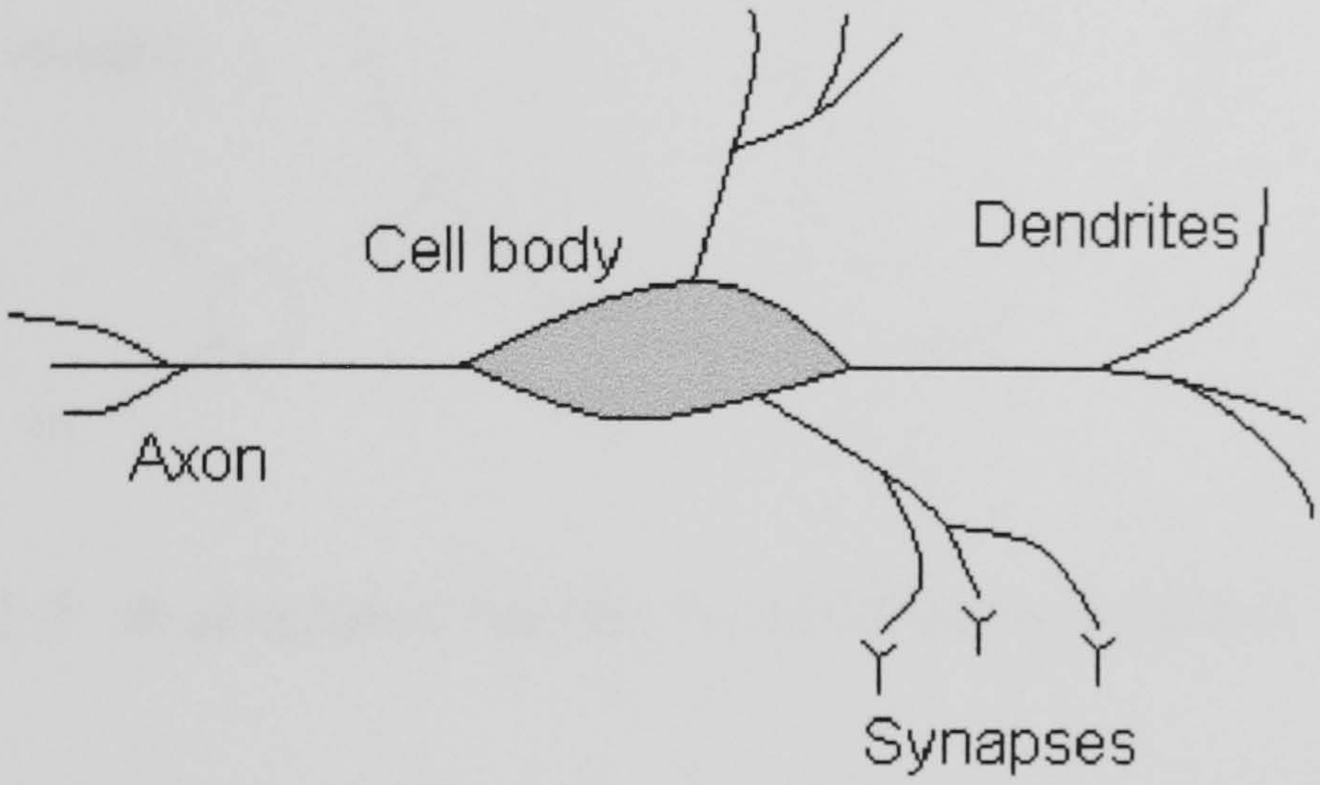


Figure 3.1: Schematic diagram of a biological neuron

Figure 3.2 shows a schematic diagram of a simulated neuron or *processing element*. Each processing element is essentially a node connected to other nodes via links resembling axon-synapse-dendrite connections. Each link is associated with a *weight*. This weight acts like a synapse to determine the strength of one node's influence on another. This influence is a product of the influencing neuron's output value and the connecting link's weight. Many input paths are combined into an overall influence by an activation function, usually a simple summation. The combined input is then modified by a transfer function and this output is passed to an output path which is connected to further input paths of other nodes in the network. It is this series of interconnected nodes which is given the term *neural network*.

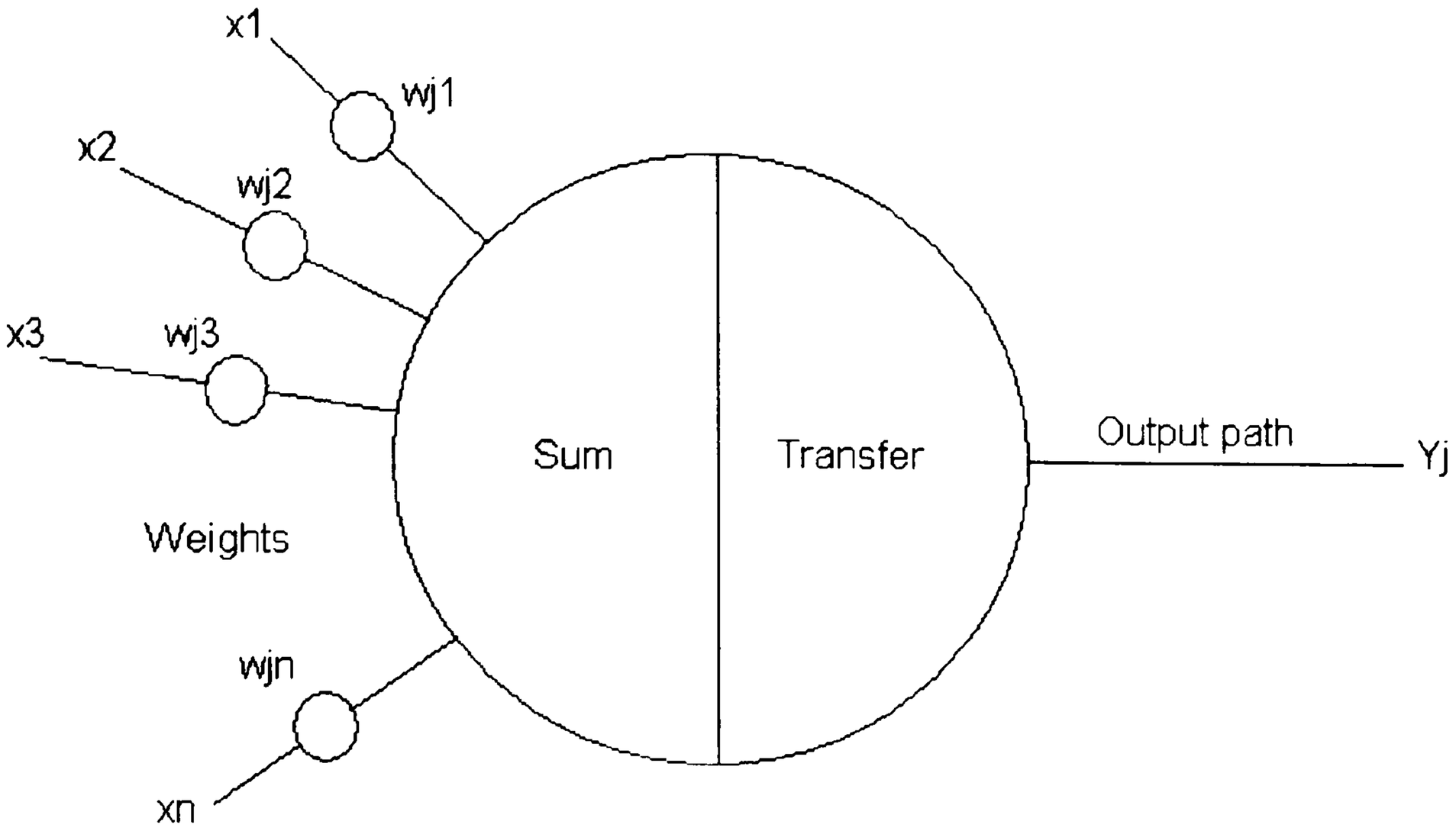


Figure 3.2: A simulated neuron, or processing element

It is tempting at this point to use the common term *Artificial Neural Network*. However, to use this terminology would be somewhat misleading as much of a real biological neuron's character is not actually modelled - the simulated neuron simply adds the weighted sum of its inputs. Also, use of the term *artificial* in this context implies that an attempt is being made to reproduce a system which behaves in an identical way to a human brain. The methodology in this work is to use a computing paradigm which developed from methods to mathematically model the brain's mode of operation, rather than to mimic its behaviour accurately. An average adult brain contains something of the order of 10^{14} neurons. A neural network contains a fraction of this number. Even large networks rarely employ more than a few thousand neurons. Also, the author does not wish to reinforce popular misconceptions about this particular science. Popular works of fiction often portray robots with human characteristics operated by neural network. This may make exciting science fiction, but the reality is somewhat different. Today's neural networks are far from sentient beings; they are simply alternative computing paradigms with no more power than the average desktop PC. Where this science may take us in the distant future remains to be seen, but in 1999, the neural network is simply an alternative

computing tool. During the years of research into the subject, a number of alternative terms for the paradigm have been coined, some of which have attempted to dispel the myth of replicating a human brain. These include

- Neurocomputing
- Network computation
- Connectionism
- Layered adaptive systems
- Self-organising networks
- Neuromorphic systems
- Parallel distributed processing

It cannot be over-emphasised that the "simulated neuron" is designed to be similar to a biological neuron only in its function as a thresholding device which, when networked, is able to perform parallel tasks. The modern neural network is not an attempt to reproduce an accurate copy of a fully functional human brain.

3.2 A history of neural computing

The McCulloch-Pitts model is an accurate mathematical definition of the neuron, but uses several simplifications. Only binary states are allowed; it operates under a discrete time assumption and synchronous operation of all neurons in a larger network is assumed. Figure 3.3 shows a schematic diagram of the model. The inputs x_i for i=1,2,...,n are 0 or 1 depending on the presence or absence of an input impulse at time t. If the neuron's output is denoted by y, then the firing rule for the model is given by:

$$y^{k+1} = \begin{cases} 1 & \text{if } \Sigma w_i x_i^t >= T \\ 0 & \text{if } \Sigma w_i x_i^t < T \end{cases}$$

where t is an integer and denotes the discrete time instant and w_i is the multiplicative weight connecting the ith input with the neuron. This assumes

a unity delay between t and t+1 and that w_i=+1 for excitatory connections and w_i=-1 for inhibitory connections. T is the neuron's threshold value, which needs to be exceeded by the weighted sum of the signals to fire. Although this model is extremely simplistic, it has the potential to perform basic logic operations. At the time of its conception, however, its actual implementation was not technologically feasible. Before the arrival of solid state electronics, the neural model was difficult to construct with the vacuum valves which were the only components available to scientists. Although not widely used on its own technical merit, it was from this model that the science of the neural network evolved.

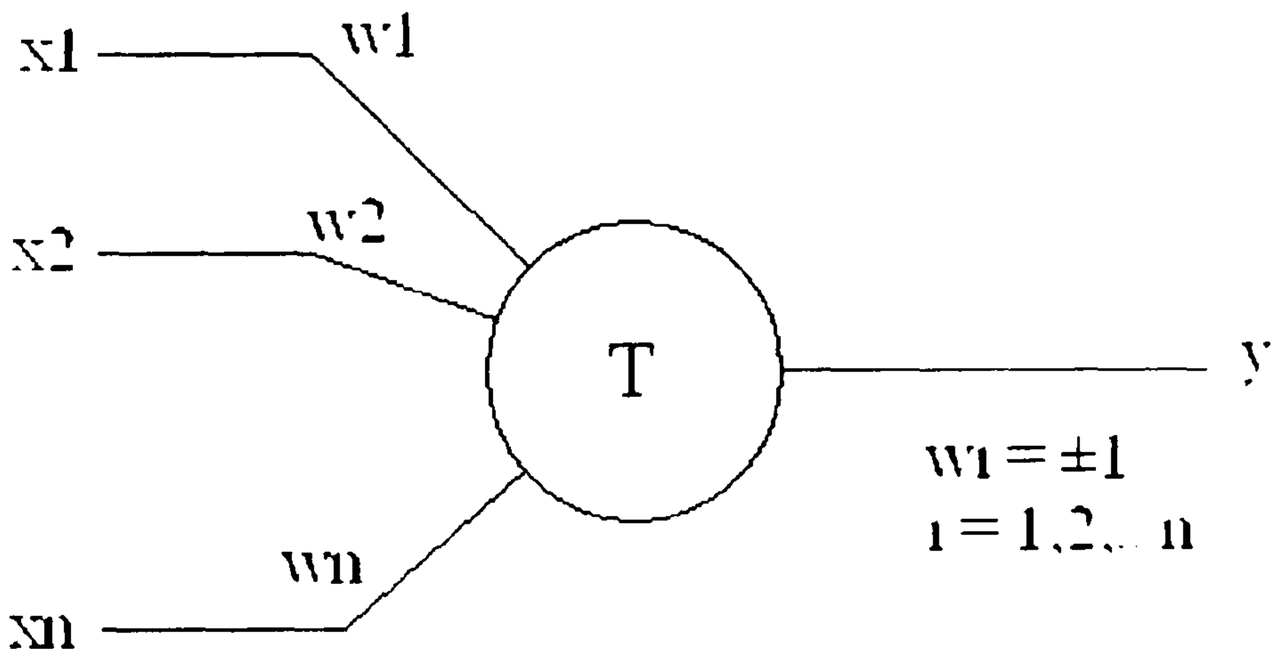


Figure 3.3: The McCulloch-Pitts model

A learning scheme for updating neural connections was first proposed by Donald Hebb[3] in 1949, which is now commonly referred to as the *Hebbian learning rule*. His idea stated that information could be stored in connections between neurons. The learning rule states that a connection weight on an input path to a processing element is incremented if both the input is high and the desired output is high. The biological equivalent of this rule says that a neural pathway is strengthened when activation on each side of the synapse is correlated.

The first recognised "learning machine" and precursor to many modern neural network models was Rosenblatt's *Perceptron*[4]. The perceptron consisted of neuron-like elements with trainable multiplicative weights, an adder and a threshold function. Figure 3.4 shows a schematic diagram of the perceptron used as a pattern classification system. The system was able to identify both abstract and geometric patterns. The perceptron was highly flexible as its performance was only degraded after damage to some of its component parts. It was also able to successfully classify patterns when noise was present in the input. The original aim of the perceptron was pattern recognition. Its primary optical stimuli were provided by an array of 400 photocells, corresponding to the light-sensitive cells of the retina. The photocells were randomly connected to associator units which received the electrical impulses from the photocells. If the input from the photocells exceeded a certain threshold value, the associator units produced an output. At this stage, extremely early in the history of neural computing, the perceptron was somewhat crude and had many limitations. The main drawback of the perceptron was its inability to solve problems which are not linearly separable at the output layer. It is this linearity that caused the perceptron to be incapable of performing the basic logical function of exclusive-OR (XOR). The 1960s saw much research into machine learning. Another important early device was ADALINE (ADAptive LINEar combiner), developed by Widrow and Hoff[5]. The Widrow-Hoff learning rule minimised the summed square error between desired and actual output during training. The ADALINE was the first neural computing system to be applied to a real-world problem. Widrow used the adaptive linear element algorithm to develop adaptive filters to eliminate echoes on telephone lines. ADALINE and its extension MADALINE (Multiple ADALINEs), which was essentially a two-layer adaline, had numerous practical applications, including pattern recognition, weather forecasting and adaptive control. This era was one of great optimism in the field of machine learning. However, lack of sufficient computing power led to a gradual slowing down of research in this area. Another blow was dealt with the publishing of Minsky and Papert's book *Perceptrons* in 1969[6]. The work cast severe doubts on the suitability of

Perceptron-type networks to solve significant problems. The book provided an involved mathematical analysis of an abstract version of the perceptron and consequently highlighted its shortcomings.

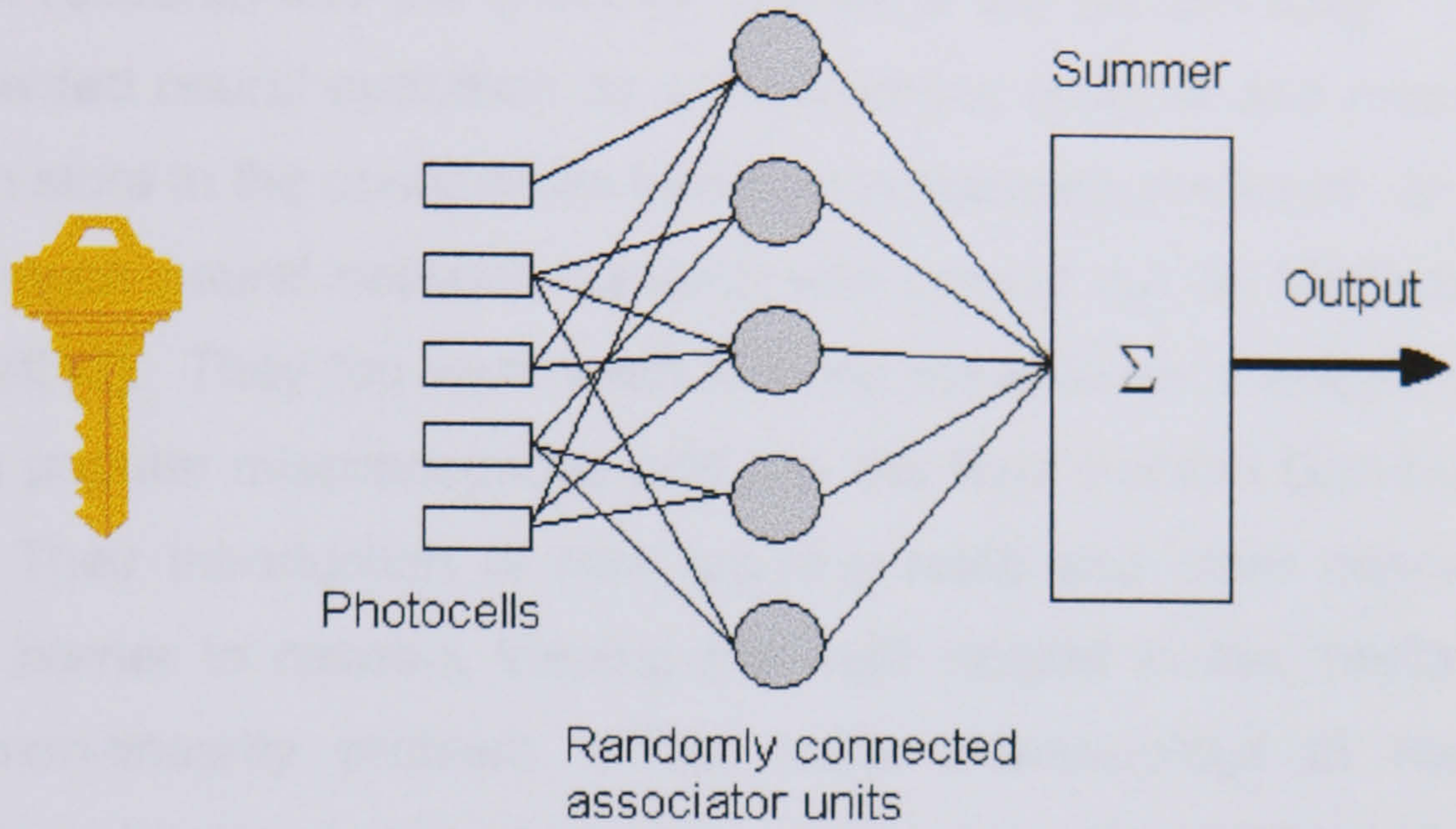


Figure 3.4: The perceptron as a pattern recognition tool

Among the conclusions that Minsky and Papert drew was that the perceptron was not capable of processing inputs that were visually non-local. It was this which conveyed to much of the scientific community that the neural network was far too limited in its scope to be of any real use as a computing paradigm and therefore was not worthy of extensive research programmes. This, coupled with the death of Frank Rosenblatt in 1971, saw a dramatic reduction in the amount of research in the field.

From the late 1960s to the early 1980s, some research was carried out into neural computing by a small number of researchers, namely Fukushima[7], Kohonen[8], Anderson[9], Carpenter[10] and Grossberg[11]. With the arrival of the 1980s came a renewed interest in the field, possibly due to the more powerful computers which were becoming available. The new enthusiasm for neural networks was headed by Hopfield and his seminal works on recurrent neural network architecture for associative memories. In his 1982

work[12], the first paper in this field to be presented to the American National Academy of Sciences since the 1960s, Hopfield described a new neural computing system which he called the "Hopfield model", or "cross-bar associative network". The model was a neurocomputing system which consisted of interconnected elements that sought an energy minimum and was based on research into the olfactory system of the garden slug! The model represented neural operation as a thresholding process and memory by information store in the connections between processing elements. In the early 1980s, much neural network research was carried out by McClelland and Rumelhart[13]. They too were keen to keep the science a respectable distance from popular misconceptions and use the term Parallel Distributed Processing. Their introduction of new learning rules and other concepts removed the barrier to network training that had existed in the 1960s by solving the non-linearity problem. The major shortcomings of neural computing that so hindered progress twenty years previously had finally been addressed, making the neural network a viable computing tool once more.

It was this work which fired the new enthusiasm for neural computing. Since the early 1980s, interest and funding in the field has increased dramatically, making it possible for a new generation of researchers to continue the work started with McCulloch and Pitts' simple neural model more than half a century ago.

3.3 Neural network operation

The difference between neural computing and conventional computing has already been stated. It is their ability to "learn through experience" that sets neural networks apart from traditional computing methods. Whereas an expert system relies on a series of rules to assimilate its knowledge, a neural network will generate its own rules by learning from a set of examples shown to it. This is achieved by use of a learning rule which adapts connections between processing elements in response to the example inputs and

(depending on the type of network used) desired outputs. There are generally three types of learning associated with neural networks:

- Supervised learning: For each input stimulus, a corresponding desired response is presented to the network, which configures its internal connections to achieve the correct input/output mapping.

- Unsupervised learning: Only input stimuli are presented to the network, which organises its internal connections in a way that hidden processing elements respond strongly to closely related groups of input stimuli.

- Reinforcement learning: This falls between the two above types. An input stimulus is presented to the network, but the network is only told whether its response is "good" or "bad".

There are two distinct phases in neural network operation: learning and recall. During the learning process, the network weights are modified in response to the applied training data. Training is similar for networks employing both supervised and unsupervised learning. When supervised learning is used, the network must be shown a series of inputs and corresponding outputs. The training sets for this method must therefore contain input stimuli and desired responses. If the desired outputs are different from the input stimuli, the network is hetero-associative, whereas if the each desired output is equal to its input stimulus for all the training vectors, the network is referred to as auto-associative. When unsupervised learning takes place, the training vectors contain only input stimuli. During learning, the most important feature of a neural network is its learning rule. This rule specifies how the connection weights are adapted in response to learning examples. In order to complete effective training, a number of different training examples are usually presented to the network several thousand times.

The recall process is the way in which the network responds to an input stimulus after training. Generally, during recall, only input stimuli are presented to the network.

3.3.1 The activation function

Each neuron consists of a processing element with input connections and a single output. A schematic diagram of a neuron has already been shown in figure 3.2. The output from any one neuron is generally given by the equation

$$o = f (\mathbf{w}^t \mathbf{x})$$

or

$$o = f (\Sigma \mathbf{w}_i \mathbf{x}_i)$$

where **w** is the weight vector and is defined as

$$\mathbf{w} = [w_1 \; w_2 \; . \; . \; . \; w_n]^t$$

and **x** is the input vector and can be defined by the equation

$$\mathbf{x} = [x_1 \; x_2 \; . \; . \; . \; x_n]^t$$

The vectors described here are column vectors and the superscript *t* indicates a transposition. The function

$$f (\mathbf{w}^t \mathbf{x})$$

is referred to as the activation function and its domain is the set of activation values *net*, which is defined as the scalar product of the weight and input vectors

$$net = \mathbf{w}^t \mathbf{x}$$

The activation function is generally referred to as

f (*net*)

3.3.2 The backpropagation paradigm

The backpropagation algorithm largely overcomes the problems associated with perceptron type networks. If an erroneous response is provided by the network, the error is propagated backwards to the previous layer of processing elements. The process continues until the error reaches the input layer. A typical backpropagation network consists of one input layer, one output layer and at least one hidden layer. Theoretically, there is no upper limit on the number of layers used, but most classification problems can be adequately solved using three or fewer hidden layers. Each layer of processing elements is fully connected to the succeeding layer. For a given input stimulus **i** and a desired output response **d**, the algorithm operates thus:

- The input **i** is presented to the input layer.
- The input is propagated through the network to obtain an output response **o** at the output layer.
- As the information is propagated through the network, all the summed inputs and output states are set.
- The scaled local error for each processing element in the output layer and delta weights is calculated.
- All the weights in the network are updated by adding the delta weights to the corresponding previous weights.

The network also employs a "bias" neuron. This is connected to all neurons in the hidden and output layers. The bias neuron provides a constant input of +1 to the entire network.

3.3.3 General Regression Networks

Another network employed in this experimentation is the General Regression Network (GRN). The GRN is a general purpose paradigm developed by Specht [19]. It uses a standard statistical formula for calculating the mean, Y, of a scalar random variable y given a measurement X of a vector random variable x. The variable x corresponds to the array of network inputs and y to the array of network outputs. When more than one input neuron is present, the formula is applied to each neuron. The calculation of the mean value requires knowledge of the probability density functions (pdf) of x and y. Which are approximated from the training vectors. The advantages of GRN are

- They learn quickly
- They converge to an optimum regression surface, as the number of samples becomes large
- They can be used effectively with sparse data
- They can handle non-stationary data, that is, data whose first derivative is not zero.

3.4 Neural networks for experimentation

3.4.1 Neural network configuration

The neural networks considered in this thesis were configured, trained and implemented using the software package NeuralWorks Professional II Plus. produced by NeuralWare, Inc. [14]. The package is commercially widely available and runs on a conventional computer. The machine used to carry out the experimentation described here was an IBM compatible 486 DX2/66 stand-alone PC. NeuralWorks Professional II Plus is a comprehensive multi-paradigm neural network development system. The package enables the user to design, build, train, test and implement a variety of types of neural

network. According to NeuralWare, Inc., the developers of the package, typical uses of NeuralWorks include

- Financial analysis
- Signal processing
- Automation and robotics
- Marketing analysis
- Medical diagnostics
- Classification
- Pattern recognition
- Process control
- Optimisation

Network performance can be monitored by a number of instruments and networks can be optimised using a large number of mathematical functions and learning rules.

3.4.2 Training the networks

As described in section 3.2, in order to function correctly, a neural network must be trained. During the learning process, a training set is applied to the network. A training set contains a number of training *vectors*. For supervised learning, each vector consists of both input and output values, therefore, for a network with **n** input neurons and **m** output neurons, each training vector will comprise **n** + **m** data. The position of each value in the vector corresponds to the position of the neuron in the network associated with that value. Consider the network shown in figure 3.5. This is a backpropagation neural network trained to classify float glass. The network consists of nine inputs, each corresponding to a separate input parameter. The input parameters are, in order, refractive index of the glass and the percentage content of each of the following chemicals: sodium, magnesium, aluminium, silicon, potassium, calcium, barium and iron. The network has

only one output neuron. When the glass is classified as float, the output neuron *fires*, or produces a "high" output. If the opposite is the case, the neuron remains dormant, giving a "low" value. The training set for the network is shown in figure 3.6. The set comprises 102 training vectors, each of which contains ten values. The first nine values are the parameters described above and the tenth value indicates whether the sample is float glass. A 1.0 corresponds to "yes" and a value of 0.0 corresponds to "no". As training progresses, each training vector is presented to the network. Although training sets typically consist of one to two hundred vectors, training can continue for several thousand presentations as the training vectors are presented more than once. How the data is presented to the network is dependent on which learning rule is used. This is an essential characteristic of the network, whichever type of learning is used. As learning progresses, the parameters which govern the learning rule may change. The long-term control of these learning parameters is referred to as the *learning schedule*.

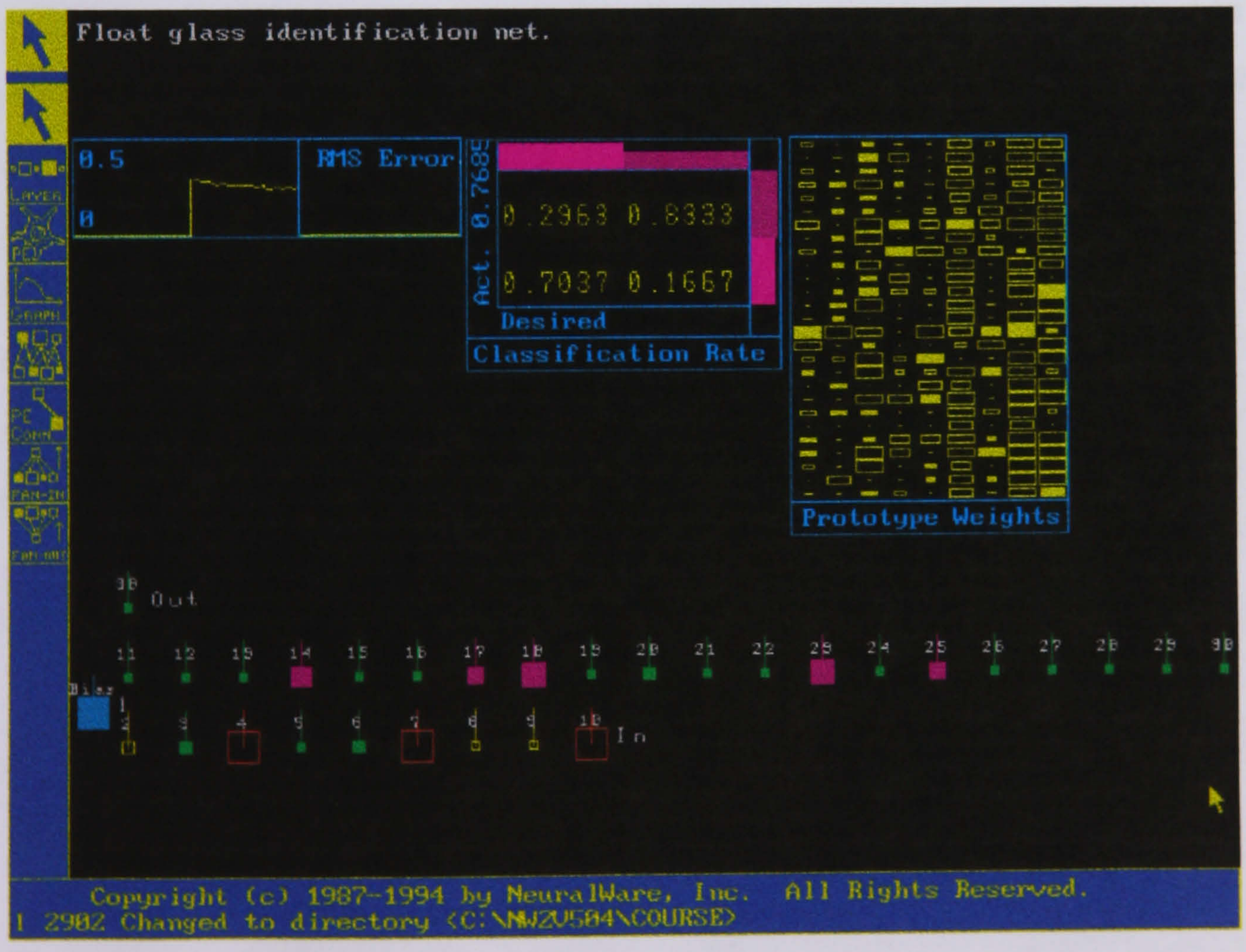


Figure 3.5: Float glass classification network.

3.4.3 The delta learning rule and its variations

One of the most widely used learning rules in the science of neural computing is the delta learning rule. The simplest form of the delta rule is based on reducing the error between the actual and desired outputs of the network by modifying incoming connection weights. The rule is valid only for networks employing supervised learning and having continuous activation functions. The learning signal for the rule, referred to as the delta, is defined by the equation

$$r = [\, d_i - f(\mathbf{w}_i^t \mathbf{x}) \,]\, f'(\mathbf{w}_i^t \mathbf{x})$$

The term $f'(\mathbf{w}_i^t \mathbf{x})$ is the derivative of the activation function $f(net)$ calculated for $net = \mathbf{w}_i^t \mathbf{x}$.

If a network's training set is ordered, problems may arise which cannot be solved by use of the delta rule in its simplest form. If data are applied to the network in an ordered manner, there is a significant risk that the RMS error will show oscillatory behaviour and fail to converge. Best results are achieved if the training vectors are presented to the network randomly. To ensure that this occurs, variations on the theme of the delta rule have been devised. The cumulative-delta rule was an attempt to alleviate the problem of structured data presentation by accumulating weight changes over several presentations and applying the changes all at once. The normalised-cumulative-delta rule is an extension of the cumulative-delta rule in that the value of the accumulation is linked to the size of the training set. This method assures that data are presented to the network randomly. The problem of poor training due to structured data presentation is therefore considerably reduced.

The use of neural networks to solve pattern recognition problems has been extensively researched during the past fifteen years[15]. Since the resurgence of interest in the subject, its applications appear to have gone from strength to strength. During this same period, fringe analysis has

become well established as a science. It is noteworthy that very little research has been done to combine the two fields. With the number of problems in fringe analysis which can be thought of as a type of pattern recognition, the use of conventional algorithmic approaches seems to be preferred. Some research, however, has been furthered. Mills et. al. Describe the use of backpropagation networks to address a number of problems in the fringe analysis process[16], while both Takeda[17] and the author[18] have attempted to specifically address the problem of phase unwrapping. The latter two approaches are described fully in chapter 4. It is the purpose of the work contained within this thesis to use various neural network architectures to investigate two aspects of the process of Fourier fringe analysis. Chapter 4 describes the use of backpropagation networks to detect phase discontinuities to assist the phase unwrapping process. Chapter 5 addresses the problem of fringe optimisation. This is a process which is still in its infancy and, to date, has not been extensively researched. The reason for this lack of research is mainly due to the fact that, until recent years, it has not been possible to have complete control over the quality of the fringe pattern during the measurement process. With the arrival of adaptive interferometry it has become possible to have much greater control over a fringe pattern, but the quality of such patterns has still largely been based on the opinion of a human operator. It has been postulated, therefore, that if this stage of the analysis were to be automated, its solution would be an ideal problem for some form of neurocomputing system.

Figure 3.6: Training set for float glass classification example.

!RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Float	
1.52101	13.64	4.49	1.1	71.78	0.06	8.75	0	0	1	0
1.51761	13.89	3.6	1.36	72.73	0.48	7.83	0	0	1	0
1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0	1	0
1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1	0
1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0	0	1	0
1.51571	12.72	3.46	1.56	73.2	0.67	8.09	0	0.24	1	0
1.51763	12.8	3.66	1.27	73.01	0.6	8.56	0	0	1	0
1.51763	12.61	3.59	1.31	73.29	0.58	8.5	0	0	1	0
1.51761	12.81	3.54	1.23	73.24	0.58	8.39	0	0	1	0
1.51911	13.9	3.73	1.18	72.12	0.06	8.89	0	0	1	0
1.5175	12.82	3.55	1.49	72.75	0.54	8.52	0	0.19	1	0
1.51966	14.77	3.75	0.29	72.02	0.03	9	0	0	1	0
1.5172	13.38	3.5	1.15	72.85	0.5	8.43	0	0	1	0
1.51764	12.98	3.54	1.21	73	0.65	8.53	0	0	1	0
1.51768	12.56	3.52	1.43	73.15	0.57	8.54	0	0	1	0
1.51768	12.65	3.56	1.3	73.08	0.61	8.69	0	0.14	1	0
1.51753	12.57	3.47	1.38	73.39	0.6	8.55	0	0.06	1	0
1.51783	12.69	3.54	1.34	72.95	0.57	8.75	0	0	1	0
1.51909	13.89	3.53	1.32	71.81	0.51	8.78	0.11	0	1	0
1.52213	14.21	3.82	0.47	71.77	0.11	9.57	0	0	1	0
1.51793	12.79	3.5	1.12	73.03	0.64	8.77	0	0	1	0
1.5221	13.73	3.84	0.72	71.76	0.17	9.74	0	0	1	0
1.51786	12.73	3.43	1.19	72.95	0.62	8.76	0	0.3	1	0
1.52667	13.99	3.7	0.71	71.57	0.02	9.82	0	0.1	1	0
1.52223	13.21	3.77	0.79	71.99	0.13	10.02	0	0	1	0
1.5232	13.72	3.72	0.51	71.75	0.09	10.06	0	0.16	1	0
1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0	0	1	0
1.51778	13.21	2.81	1.29	72.98	0.51	9.02	0	0.09	1	0
1.51824	12.87	3.48	1.29	72.95	0.6	8.43	0	0	1	0
1.51754	13.48	3.74	1.17	72.99	0.59	8.03	0	0	1	0
1.51977	13.81	3.58	1.32	71.72	0.12	8.67	0.69	0	1	0
1.52227	14.17	3.81	0.78	71.35	0	9.69	0	0	1	0
1.52172	13.48	3.74	0.9	72.01	0.18	9.61	0	0.07	1	0
1.52152	13.05	3.65	0.87	72.32	0.19	9.85	0	0.17	1	0
1.523	13.31	3.58	0.82	71.99	0.12	10.17	0	0.03	1	0
1.51574	14.86	3.67	1.74	71.87	0.16	7.36	0	0.12	0	1
1.51631	13.34	3.57	1.57	72.87	0.61	7.89	0	0	0	1
1.5159	13.02	3.58	1.51	73.12	0.69	7.96	0	0	0	1
1.51627	13	3.58	1.54	72.83	0.61	8.04	0	0	0	1
1.5159	12.82	3.52	1.9	72.86	0.69	7.97	0	0	0	1
1.51592	12.86	3.52	2.12	72.66	0.69	7.97	0	0	0	1
1.51594	13.09	3.52	1.55	72.87	0.68	8.05	0	0.09	0	1
1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0	0	0	1
1.51645	13.4	3.49	1.52	72.65	0.67	8.08	0	0.1	0	1
1.5164	12.55	3.48	1.87	73.23	0.63	8.08	0	0.09	0	1
1.51588	13.12	3.41	1.58	73.26	0.07	8.39	0	0.19	0	1
1.5159	13.24	3.34	1.47	73.1	0.39	8.22	0	0	0	1
1.5186	13.36	3.43	1.43	72.26	0.51	8.6	0	0	0	1
1.51689	12.67	2.88	1.71	73.21	0.73	8.54	0	0	0	1
1.51811	12.96	2.96	1.43	72.92	0.6	8.79	0.14	0	0	1
1.5182	12.62	2.76	0.83	73.81	0.35	9.42	0	0.2	0	1
1.52725	13.8	3.15	0.66	70.57	0.08	11.64	0	0	0	1
1.53125	10.73	0	2.1	69.81	0.58	13.3	3.15	0.28	0	1
1.52222	14.43	0	1	72.67	0.1	11.52	0	0.08	0	1
1.51818	13.72	0	0.56	74.45	0	10.99	0	0	0	1
1.52777	12.64	0	0.67	72.02	0.06	14.4	0	0	0	1

1.51892	13.46	3.83	1.26	72.55	0.57	8.21	0	0.14	0	1
1.51829	13.24	3.9	1.41	72.33	0.55	8.31	0	0.1	0	1
1.51673	13.3	3.64	1.53	72.53	0.65	8.03	0	0.29	0	1
1.51844	13.25	3.76	1.32	72.4	0.58	8.42	0	0	0	1
1.51687	13.23	3.54	1.48	72.84	0.56	8.1	0	0	0	1
1.51707	13.48	3.48	1.71	72.52	0.62	7.99	0	0	0	1
1.51667	12.94	3.61	1.26	72.75	0.56	8.6	0	0	0	1
1.52068	13.55	2.09	1.67	72.18	0.53	9.57	0.27	0.17	0	1
1.52614	13.7	0	1.36	71.24	0.19	13.44	0	0.1	0	1
1.51813	13.43	3.98	1.18	72.49	0.58	8.15	0	0	0	1
1.51789	13.19	3.9	1.3	72.33	0.55	8.44	0	0.28	0	1
1.51806	13	3.8	1.08	73.07	0.56	8.38	0	0.12	0	1
1.51674	12.79	3.52	1.54	73.36	0.66	7.9	0	0	0	1
1.51851	13.2	3.63	1.07	72.83	0.57	8.41	0.09	0.17	0	1
1.51662	12.85	3.51	1.44	73.01	0.68	8.23	0.06	0.25	0	1
1.51839	12.85	3.67	1.24	72.57	0.62	8.68	0	0.35	0	1
1.5161	13.33	3.53	1.34	72.67	0.56	8.33	0	0	1	0
1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0	0	1	0
1.51779	13.64	3.65	0.65	73	0.06	8.93	0	0	1	0
1.5161	13.42	3.4	1.22	72.69	0.59	8.32	0	0	1	0
1.51646	13.04	3.4	1.26	73.01	0.52	8.58	0	0	1	0
1.52121	14.03	3.76	0.58	71.79	0.11	9.65	0	0	1	0
1.51796	13.5	3.36	1.63	71.94	0.57	8.81	0	0.09	1	0
1.52211	14.19	3.78	0.91	71.36	0.23	9.14	0	0.37	1	0
1.51514	14.01	2.68	3.5	69.89	1.68	5.87	2.2	0	0	1
1.52151	11.03	1.71	1.56	73.44	0.58	11.62	0	0	0	1
1.51666	12.86	0	1.83	73.88	0.97	10.17	0	0	0	1
1.51316	13.02	0	3.04	70.48	6.21	6.96	0	0	0	1
1.51321	13	0	3.02	70.7	6.21	6.93	0	0	0	1
1.52043	13.38	0	1.4	72.25	0.33	12.5	0	0	0	1
1.51905	14	2.39	1.56	72.37	0	9.57	0	0	0	1
1.51829	14.46	2.24	1.62	72.38	0	9.26	0	0	0	1

References

1. J.M.Zurada. Introduction to Artificial Neural Systems. West, 1992.
2. W.S.McCulloch & W.Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5(1943)115
3. D.O.Hebb. The Organization of Behavior. Wiley, New York, 1949.
4. F.Rosenblatt. Principles of Neurodynamics, Perceptrons and the Theory of Brain Mechanisms. Spartan Books, New York, 1962.
5. B.Widrow & M.Hoff. "Adaptive switching circuits", 1960 IRE WESCON Convention Record, 4(1960) 96-104.
6. M.Minsky & S.Papert. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969.
7. K.Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks 1(1988)119-130.
8. T.Kohonen. Self Organisation and Associative Memory. Springer Verlag, Heidelberg, 1977
9. J.A.Anderson, J.W.Silverstein, S.A.Ritz & R.S.Jones. Distinctive features, categorical perception and probability learning: some applications of a neural model. Psychological Review, 84(1977)413-51
10. G.A.Carpenter & S.Grossberg. A massively parallel architecture for a self-organising neural pattern recognition machine. Computer Vision, Graphics and Image Processing, 37(1987)54-115
11. S.Grossberg. Adaptive pattern classification and universal recoding, Biological Cybernetics 23(1976)121-143, 187-202.
12. J.J.Hopfield. Neural networks and physical systems with emergent collective properties. Proc. Nat. Acad. Sci. USA 79(1982)2554-8.
13. J.L.McClelland & D.E.Rumelhart. Distributed memory and the representation of general and specific information. Journal of Experimental Psychology 114(1985)159-188
14. NeuralWorks Professional II Plus Reference Guide. NeuralWare Inc. 1993.
15. J.Loncelle, N.Derycke & F.F.Soulié. Co-operation of GBP and LVQ networks for optical character recognition. Int. Joint Conf. on Neural Networks 3(1992)694-99
16. H.Mills, D.R.Burton & M.J.Lalor. Applying backpropagation neural networks to fringe analysis. Optics and Lasers in Engineering 23(1995), 331-341.
17. M.Takeda. Phase unwrapping by neural network. Proc. FRINGE '93.
18. D.J.Tipper, D.R.Burton & M.J.Lalor. A neural network approach to the phase unwrapping process in fringe analysis, Nondestructive Testing and Evaluation 12(1996)391-400.
19. D.F.Specht. A general regression neural network. IEEE Transactions on Neural Networks, 2(1991)6, 568-76

Chapter 4: Phase unwrapping

Chapter 4: Phase Unwrapping.

4.1 Introduction

When constructing a map of surface contours using an interferometric fringe pattern, it is desirable to know not only the amplitude of the waveform, but also its phase value. The height information in the fringe pattern is related to the phase values of the reflected light. It is the mathematical calculation of the phase values of an image, which leads to the process known as *phase unwrapping*. In chapter 2, the concept of a cosinusoidal fringe pattern was introduced whose intensity at any given point can be described by the equation

$$I(x,y) = a(x,y) + b(x,y)\cos[\phi(x,y)]$$

where

$a(x,y)$ = additive noise - offset term

$b(x,y)$ = multiplicative noise - amplitude term

$\phi(x,y) = \phi_c + \phi_m$

ϕ_c = carrier phase

ϕ_m = modulation phase

As described in chapter 2, if a fringe pattern is subjected to either phase stepping or FFT techniques, the final phase values will be calculated as an arctangent function:

Using the phase stepping technique, phase is calculated by

$$\phi(x,y) = \arctan\left(\frac{I(\pi/2) - I(3\pi/2)}{I(0) - I(\pi)}\right)$$

whereas the FFT will give phase as

$$\phi(x,y) = \arctan\left(\frac{\mathrm{Im}[c(x,y)]}{\mathrm{Re}[c(x,y)]}\right)$$

where Im = imaginary component of c(x,y)

Re = real component of c(x,y)

Due to the arctangent function used in this calculation, phase values will invariably be returned which are "wrapped" modulo 2π. It is, therefore, necessary to "unwrap" the phase: that is to re-create a continuous phase distribution. Figure 4.1 shows a continuous one-dimensional phase distribution and how that same distribution appears when it is wrapped modulo 2π.

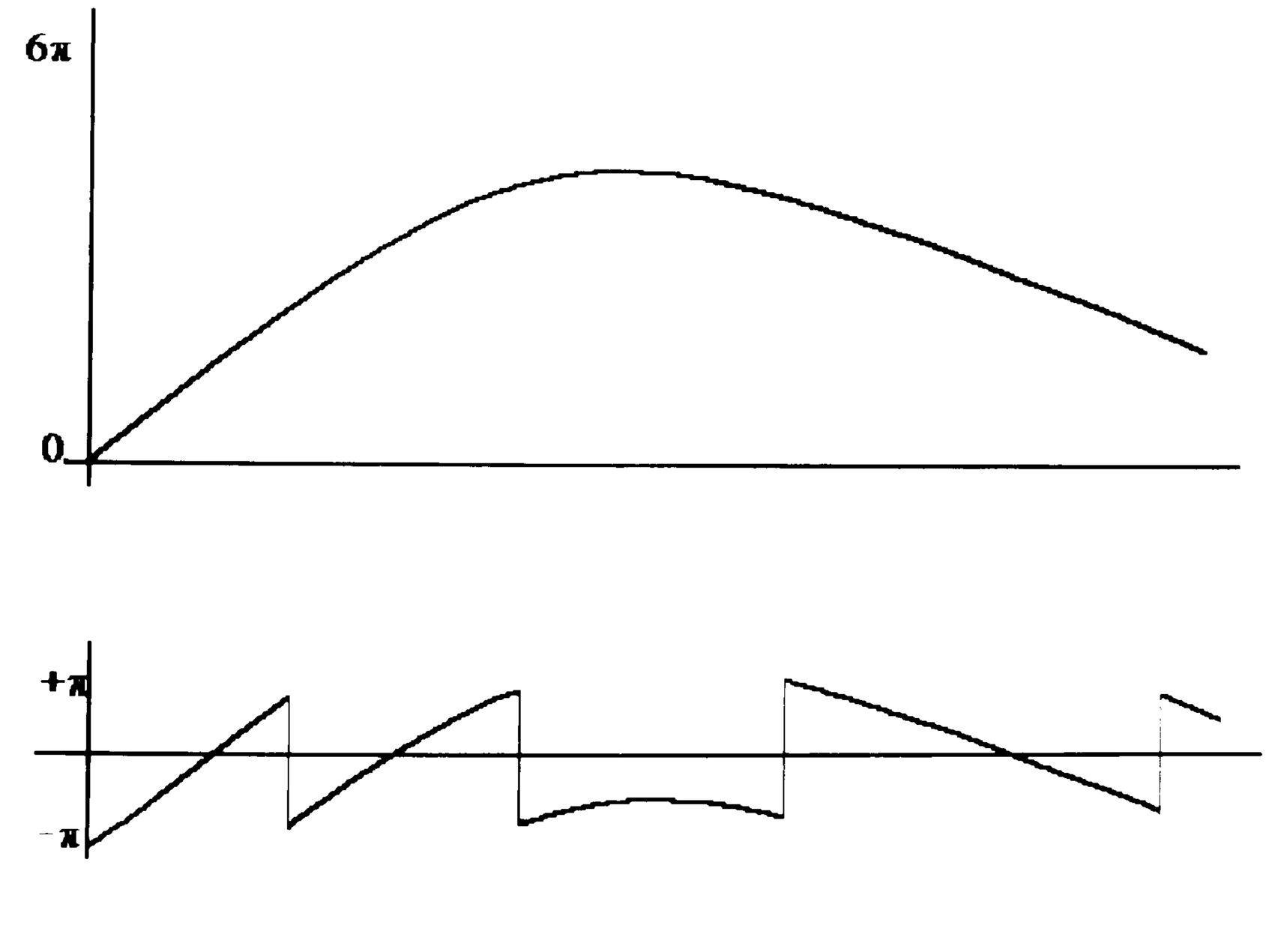


Figure 4.1: A 1-D phase distribution and how it appears when wrapped modulo 2π.

In a two-dimensional image, conventional computer mapping of wrapped phase data causes bands of contour lines where 2π phase jumps occur. An example of this is shown in figure 4.2. It can be argued that the most reliable

method of reconstructing the phase data is to unwrap solely "by hand", as the human brain can recognise what the unwrapped phase distribution *should* look like. However, due to the mathematical complexity of the problem, the time taken would be extremely impractical.

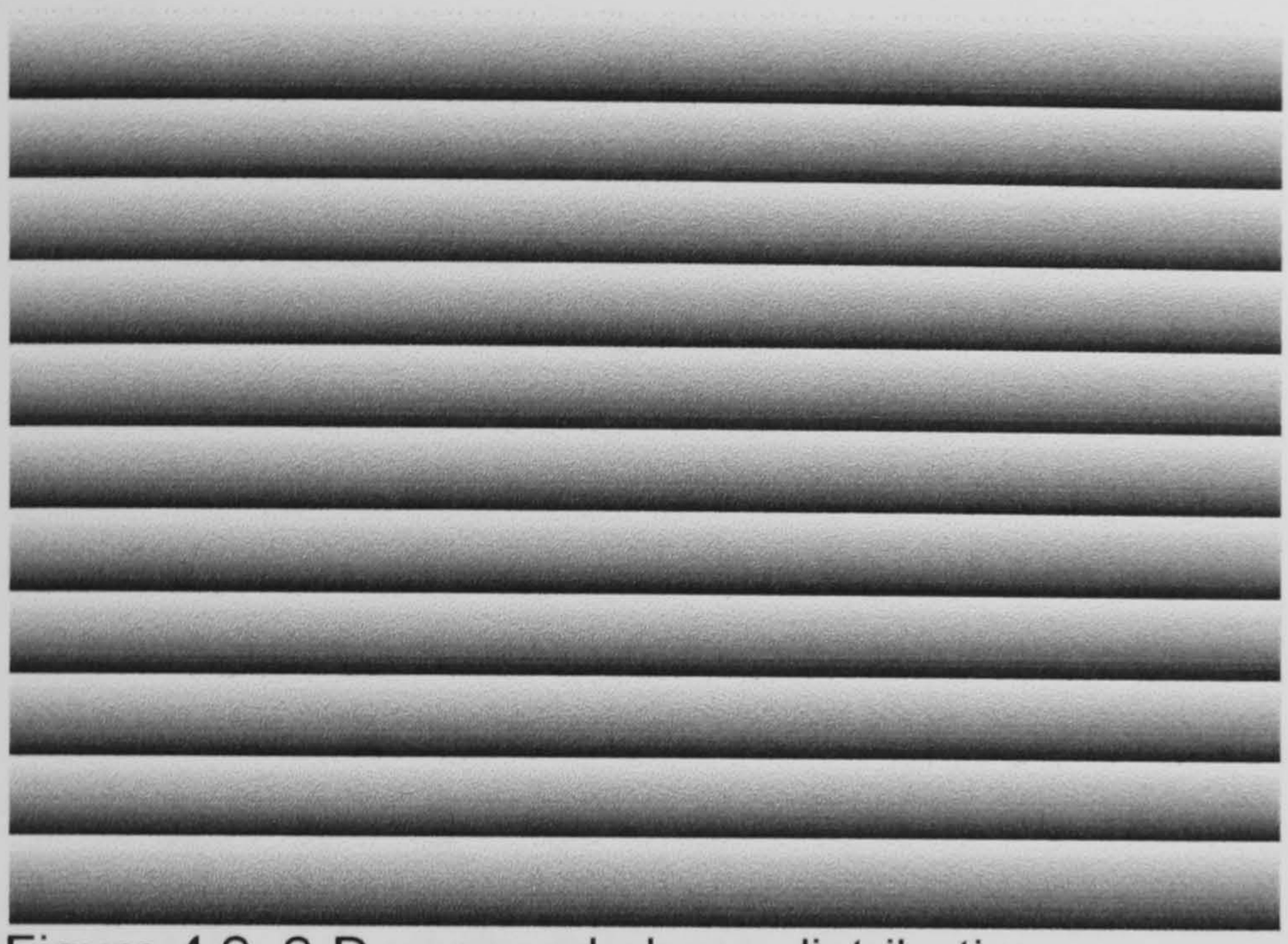
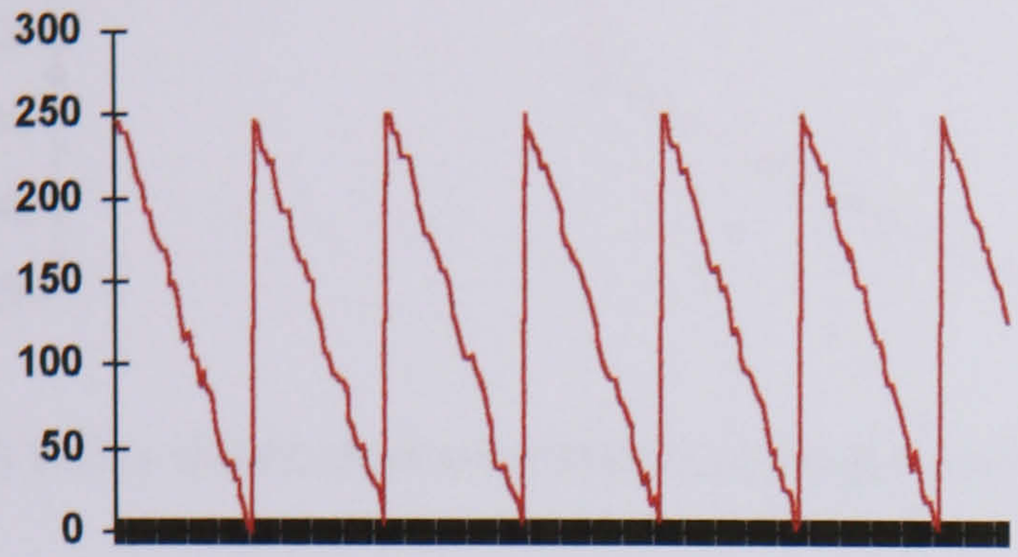


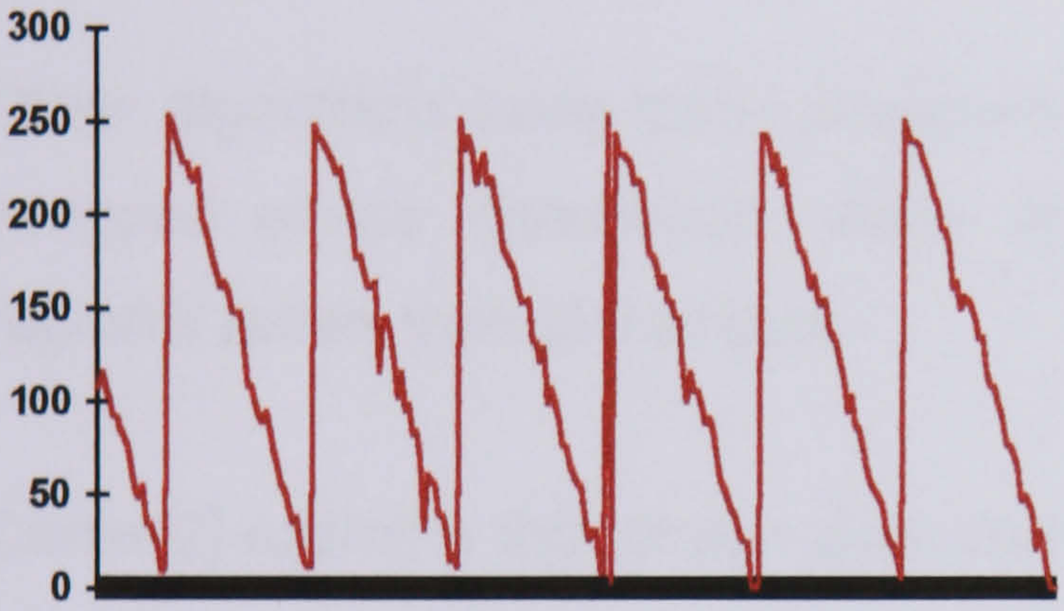
Figure 4.2: 2-D wrapped phase distribution.

Many algorithms have been proposed to deal with the phase unwrapping problem. The earliest of these is now known an "Schafer's algorithm"[1] and involves a pixel-by-pixel approach of comparing adjacent phase values. Researchers in this field now regard this as the “classical” point-to-point phase unwrapping algorithm. The theory of Schafer's algorithm can be explained thus: The first phase value in a row (or column) is recorded. The phase value of it's immediate neighbour is then recorded and compared with the previous value. If the difference between the two values is in the region of 2π, the value is updated by adding or subtracting 2π, depending on the sign of the difference. This is the simplest form of phase unwrapping and is adequate if the phase distribution contains very little noise. If significant noise is present in the distribution, any errors are easily propagated through the final unwrapped phase map. Figure 4.3 shows the effect of a noise spike on Schafer's algorithm. Figure 4.3(a) shows a noise-free distribution and 4.3(c) how it appears when unwrapped by this method. Figure 4.3(b) shows the same distribution, but with the introduction of a "spot noise" spike. The algorithm will treat the noise as a phase wrap and update the remaining

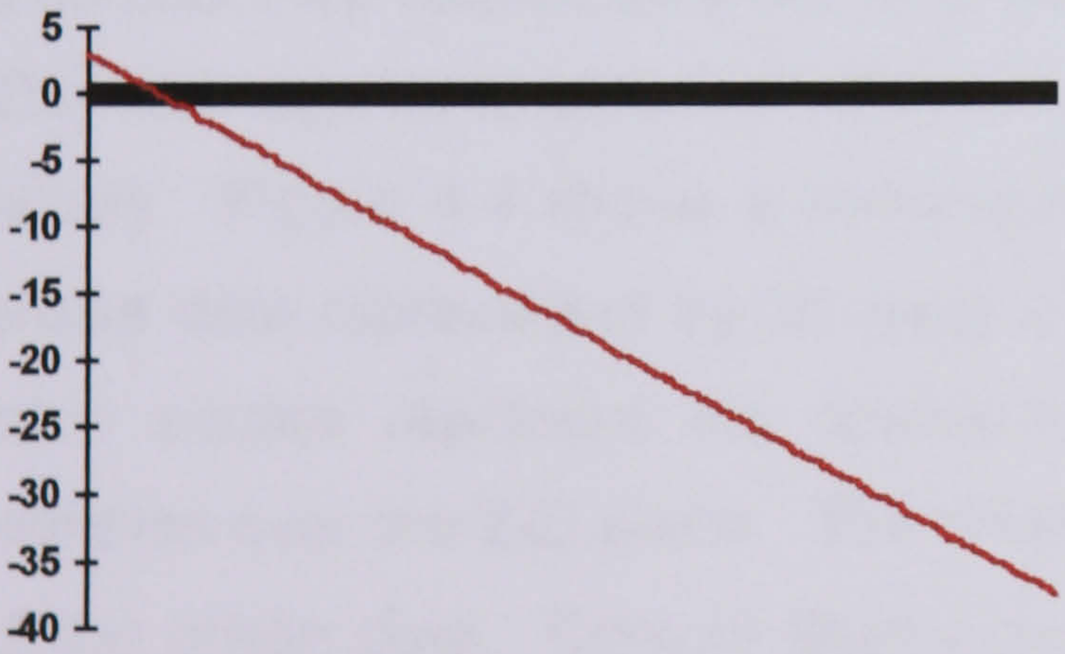
phase values accordingly. The error is propagated through the entire unwrapped distribution and the final result will be inaccurate.



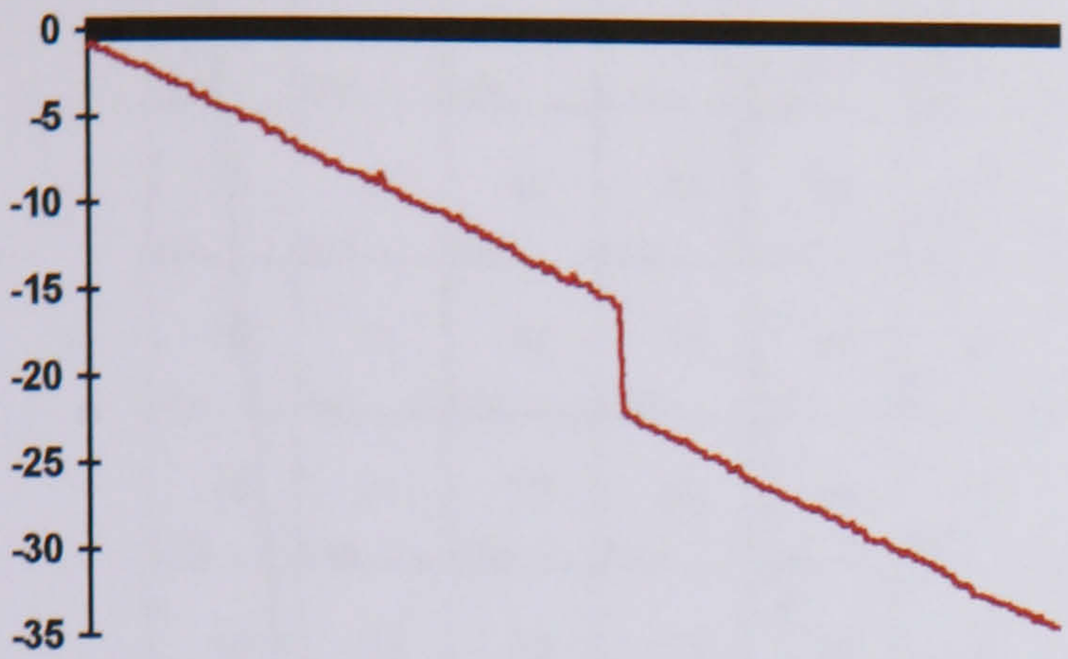
(a) Noise free 1-D wrapped phase distribution.



(b)Wrapped phase distribution with single noise spike.



(c) Noise-free distribution unwrapped using Schafer's algorithm



(d) Noisy distribution unwrapped using Schafer's algorithm

Figure 4.3: Phase distributions unwrapped by Schafer's algorithm

Other algorithms have been proposed to deal with the problem of noise in wrapped phase distribution, many of which employ analyses based on regional rather than global data.

Carter[2] explains that phase data can't be correctly unwrapped over a two-dimensional domain which includes zeroes in the field amplitude as the phase is singular at these points. Unwrapping in one dimension and plotting a contour map causes dark bands to trail out behind each zero along the row. This also applies to columns. The only way to avoid this result is to unwrap locally. Figure 4.4 shows a rectangular array of sampled two-dimensional phase data represented by an array of numbered squares. The corners of each square represent the spatial location of nearest neighbour phase samples over the 2-D plane. The small numbers indicate the original values of the phase data. Contour lines occur at 5.94, 6.28 and 0.34. These are shown interpolated through the grid. Unwrapping begins at square zero, whose value is 5.84. The points at the other corners of the square are tested to see by how much they differ from 5.84. If the difference is greater than π, then 2π phase jumps are likely to exist between some of the values along the path bounding the square. To remove these phase jumps, 2π must be added to or subtracted from the points until the values are within π of the local phase reference.

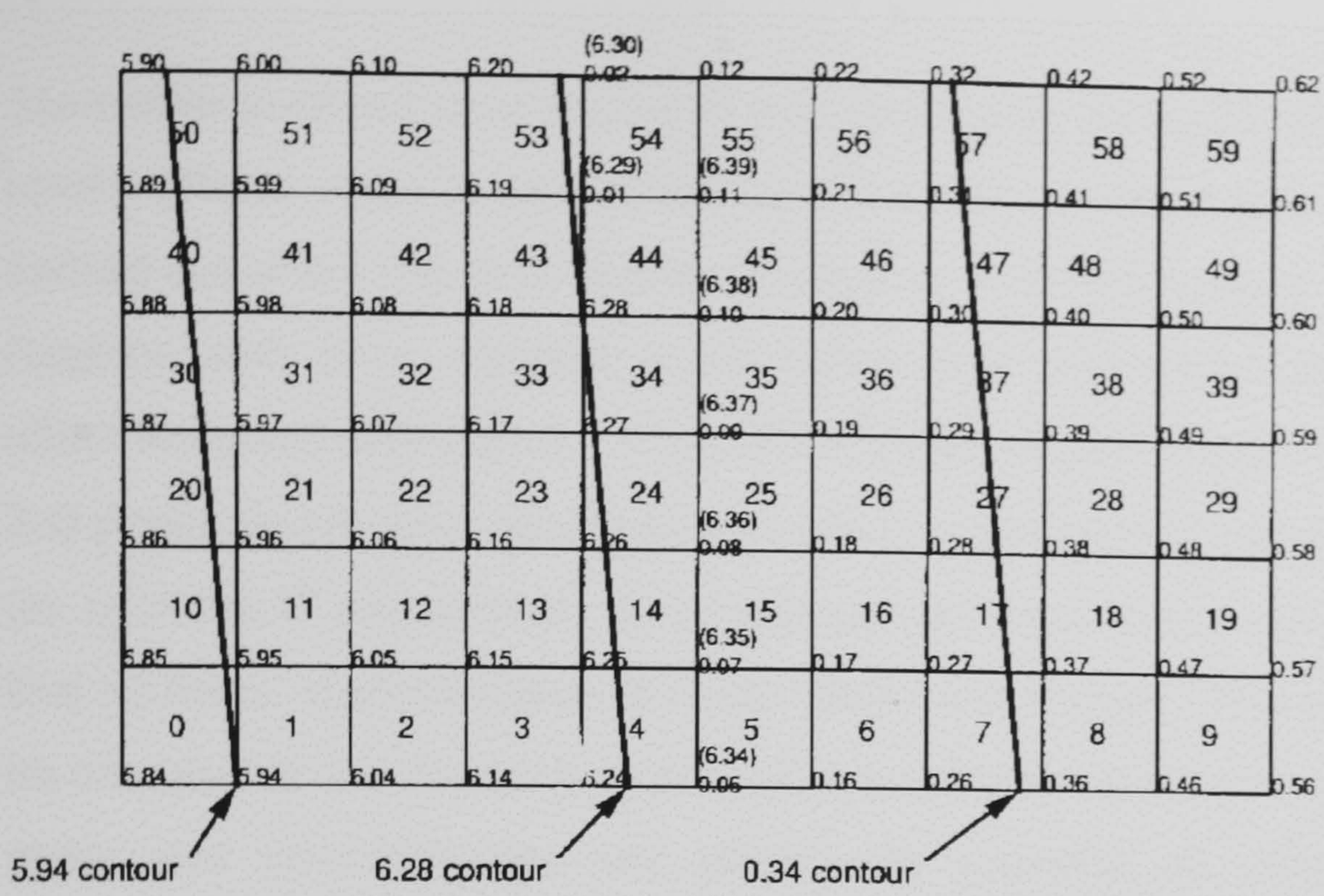


Figure 4.4: 2-dimensional phase data

To improve on the accuracy of phase unwrapping, Huntley[3] describes an algorithm, which is relatively noise-immune. The essence of the approach is to ensure that the final unwrapped phase distribution is completely independent of the path by which the process is carried out. This is achieved by placing "cut lines" in the phase map, which act as barriers to unwrapping. Generally, the source of a discontinuity in the phase map will be at one end of a cut line, while the other end of the cut is attached to a discontinuity source of an opposite sign, or to the edge of the phase map. Discontinuity sources tend to occur naturally in pairs of opposite sign, but some isolated sources can occur near the phase map boundary. Cuts are represented by two arrays of flags, one each for a vertical and horizontal cut, and once the cut lines have been constructed, unwrapping can be carried out in any order. Computation time varies with both signal to noise ratio and on the length of cuts and is apparently comparable to the time taken with a "conventional" algorithm

Huntley and Saldner [4] proposed a "temporal" phase unwrapping algorithm. Most algorithms search the 2-D spatial domain for 2π discontinuities; only one map is required, but errors can propagate outward from high noise regions, significantly corrupting the unwrapped image. The proposed

alternative involved unwrapping in one dimension along the time axis. Each pixel's phase value is measured as a function of time and unwrapping is carried out along the time axis for each pixel independently of all the others. Regions with poor signal-to-noise ratios and boundaries don't adversely affect the good data points. A set of 2-D phase maps is assembled to form a 3-D phase distribution, where $\phi(m, n, t)$ is used to denote the phase at a pixel (m, n) in the $\mathbf{t}^{th}$ phase map. Unwrapping can occur along any path, provided that no noise, discontinuities or major faults are present. The phase needs to be sampled at a sufficiently high spatial and temporal frequency. In practice, noise and discontinuities will always be present, so phase errors will invariably occur. In a 3-D map, these regions are orientated along the t axis and, provided that the boundaries don't change with time, they can be avoided by unwrapping in a direction parallel to the time axis. Although many intermediate phase maps are required for this method, the approach is simple and robust. Phase errors remain in regions of low signal-to-noise ratio. The method is particularly suitable to applications where the phase map builds up slowly and where phase changes rather than absolute phase values are important.

Ghiglia *et al.* [5] used a method employing cellular automata. Cellular automata can best be described as a series of discrete, simple mathematical processes, whose results, when combined result in a more complex whole. A cellular automaton is based on a discrete lattice of identical sites. Each site can be in any one of k states and evolves according to a simple function of its neighbouring sites. Experiments suggest that the patterns generated in the evolution of cellular automata from initially random states fall into four general classes:

1. Evolution to a homogeneous state,
2. Evolution to simple, separated periodic structures,
3. Evolution to chaotic, aperiodic patterns and
4. Evolution to complex patterns of localised structures.

This method "...promise[s] that this simple computation can be done in an unbiased, non-directional manner". The algorithm is based on a "strength-of-vote" rule. A point in the phase map is identified and its phase value is compared with its immediate neighbours. The strength of each neighbouring point's vote is defined as equal to the integral number of 2π rad necessary to wrap the respective phase differences. The integer strengths of vote are accumulated and the point changed by 2π in a direction appropriate to the accumulated vote strength.

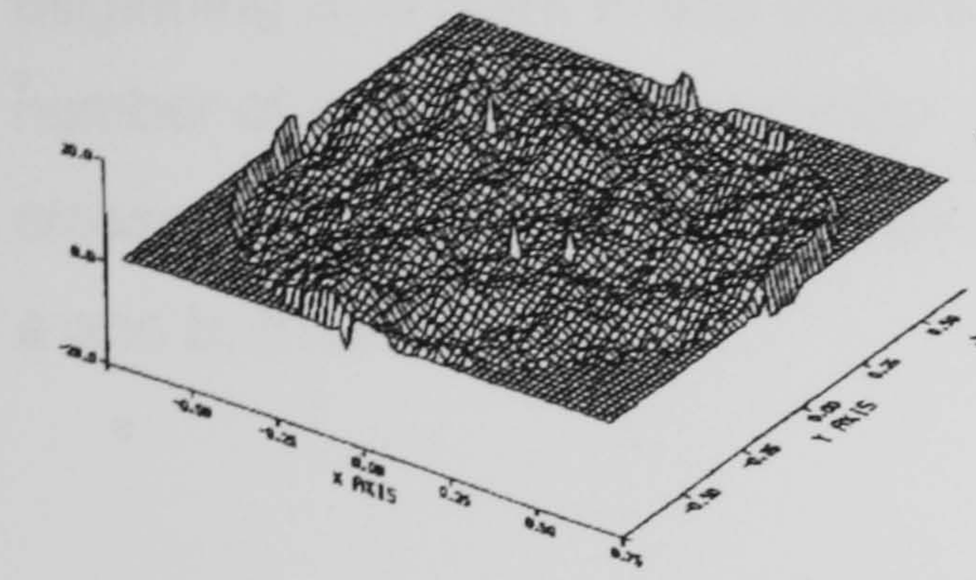
As with most algorithmic approaches, problems arise when path inconsistencies occur, whether these are noise induced, natural or aliasing induced. Generally, when a path hits an inconsistent point, it carries this inconsistency to neighbouring phase values and generates discontinuities. These normally have to be dealt with by post-filtering operations, which may corrupt some of the phase data. If a natural dislocation is present, it cannot simply be removed by additions of 2π. Also, removal of natural dislocations may have an adverse effect on the phase value. If an aliasing induced dislocation occurs, it is impossible to distinguish from a natural dislocation without *a priori* knowledge. Under these conditions, correct phase unwrapping is not possible. The cellular automata method makes allowances for this and "...offer[s] promise of other means of powerful and parallel computations"[5].

A novel approach to the unwrapping problem was proposed by Gierloff[6]. His method differs from the classical point-to-point algorithm by defining regions which are free from discontinuities. The method is much less susceptible to noise and does not propagate errors through the analysis. The philosophy behind this approach is to make a small number of decisions based on a much larger amount of analysis. The algorithm categorises points into regions by determining their relationships to neighbouring points, which have already been classified. A point is classified as belonging to a particular region if it is within certain tolerances. For simple images, regions can be classified by a simple raster-type scan, but for more complex images,

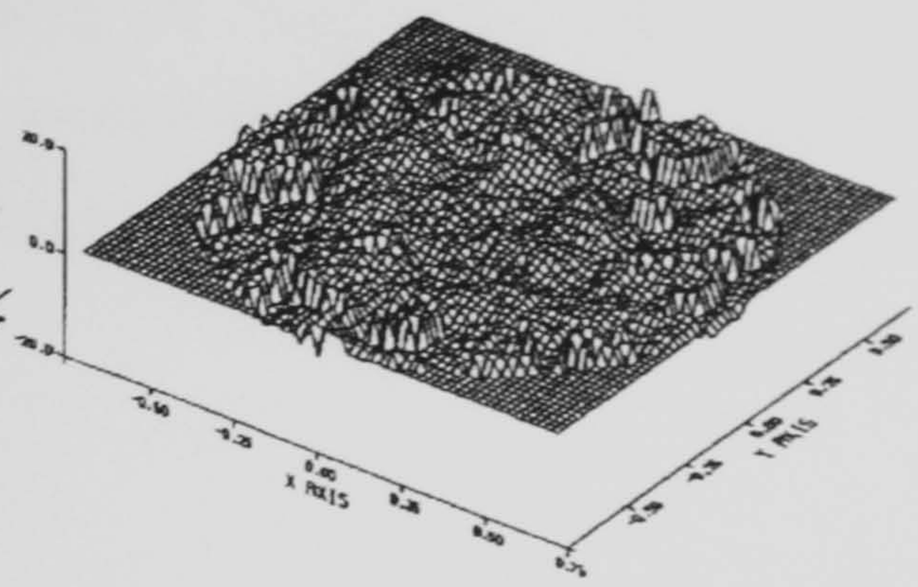
more intricate computation is required. Scanning of simple patterns can lead to "dead" spots at the edges of the image which cause calculation to cease before the region is completely defined. This can cause a larger number of regions than necessary to be defined. The method is somewhat computationally expensive, however, even for relatively consistent data.

Once all the regions have been defined, each is compared with its immediate neighbours to ascertain whether a phase wrap exists between them. The phase wraps are identified, relationships between regions are defined and ones with no phase ambiguities are combined to form larger regions. The new regions are further compared to determine any necessary phase shifts. The edge between two adjacent regions is traced and the edge points are compared. Figure 4.5 shows a comparison of unwrapping techniques. Figure 4.5(a) shows an original phase profile and (b) shows the corresponding wrapped phase profile. Figure 4.5(c) shows the phase distribution as unwrapped using Gierloff's regional method, and (d) shows the same phase distribution when unwrapped using Schafer's algorithm. The benefit of the regional analysis can clearly be seen. The regional analysis has contained any prominent errors where the point-to-point algorithm has propagated these errors through the analysis, resulting in a wildly inaccurate final phase distribution.

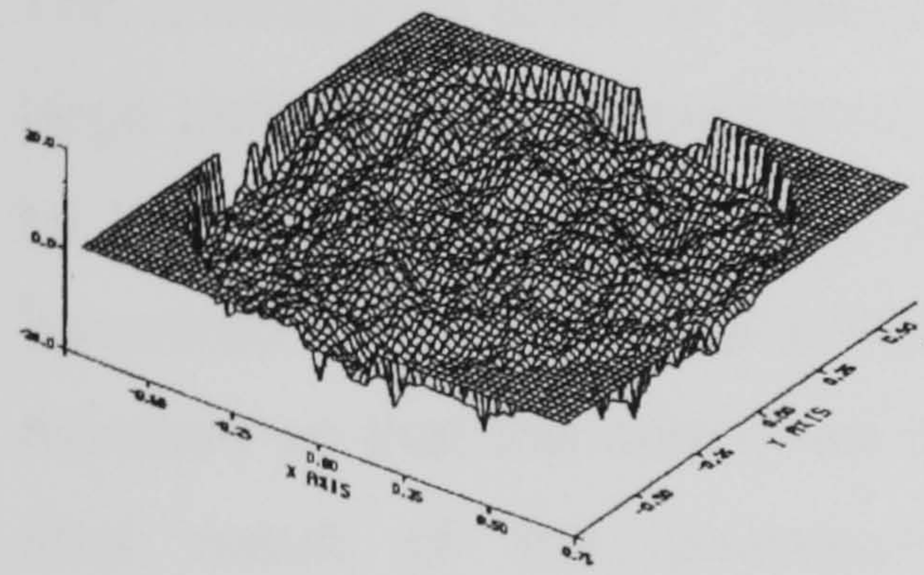
Bone[7] uses a slightly more complex method, which uses local phase information to mask out parts of the field, which cause inconsistencies in the unwrapping. This method uses what is described as *residue analysis of path dependence*. This is shown schematically in figure 4.6.



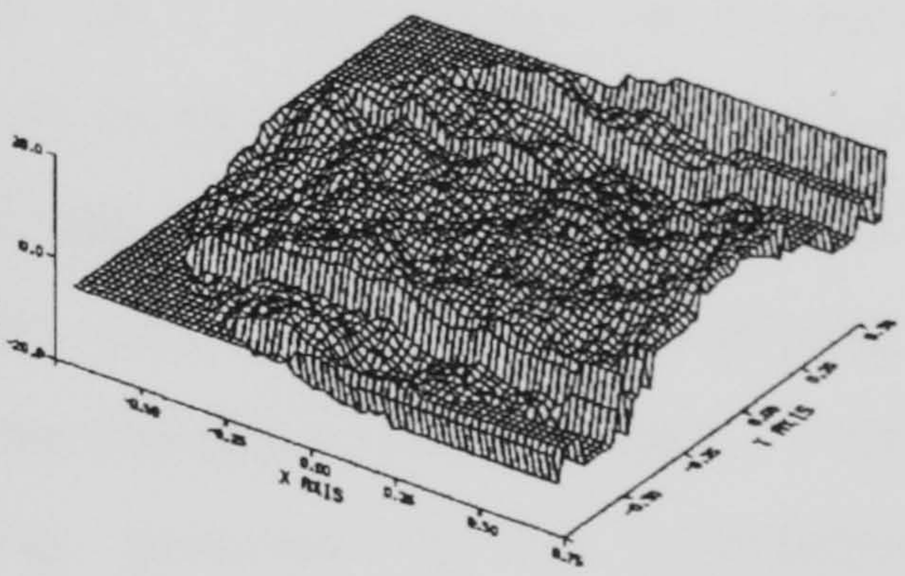
(a) Original phase profile



(b) Wrapped phase profile



(c) Unwrapped by regional analysis



(d) Unwrapped by Schafer's algorithm

Figure 4.5: Gierloff's regional phase unwrapping algorithm

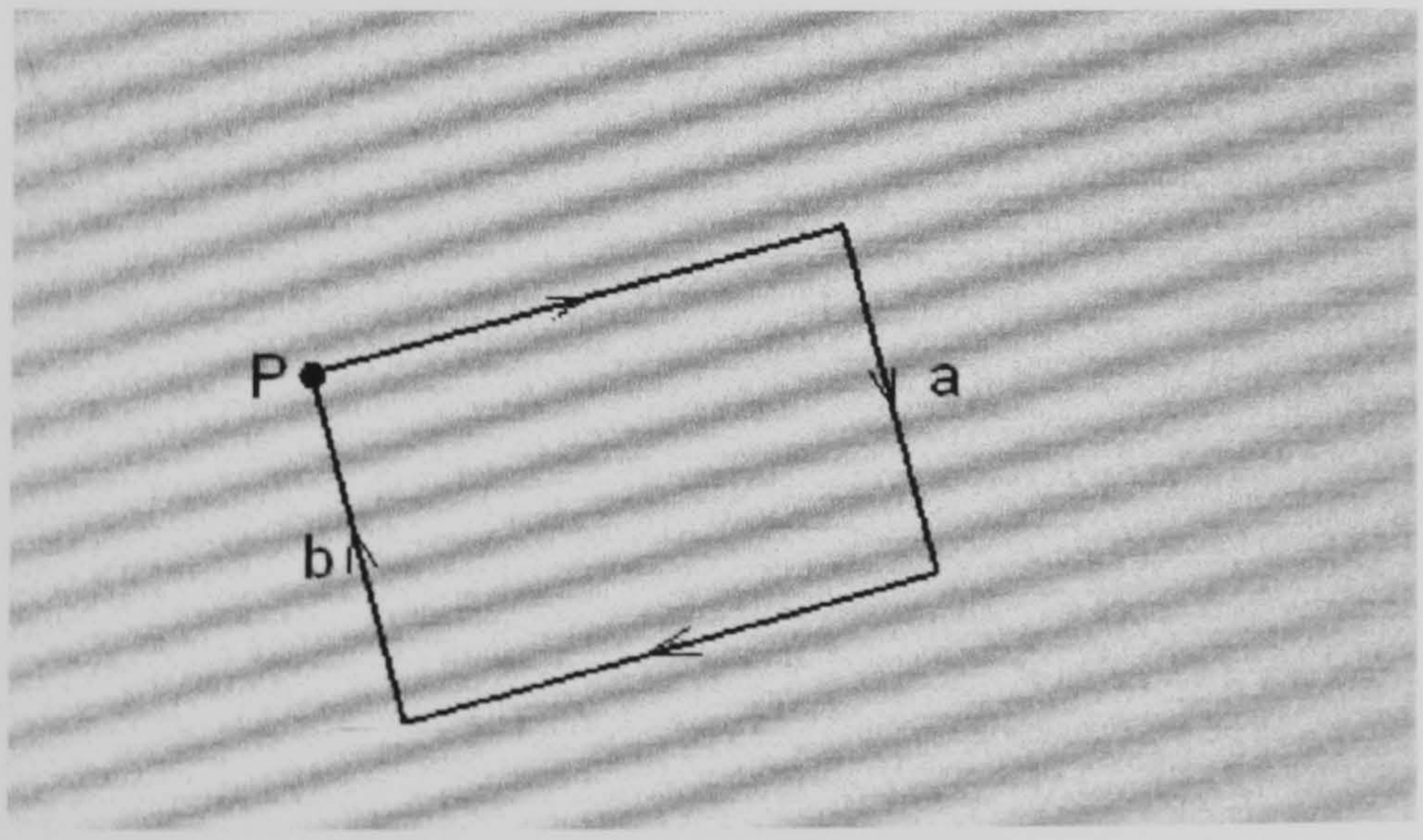


Figure 4.6: Residue analysis of path dependence.

Beginning at a point **P** and traversing the square in the direction shown, the number of discontinuities crossed along **a** should be the same as the number crossed along **b**. In mathematical terms, the unwrapping error is the sum of **a** and **b**, thus:

$$\Delta = \Sigma\left(\frac{\phi\big(x(k),y(k)\big)-\phi\big(x(k-1),y(k-1)\big)}{2\pi}\right)$$

The unwrapping error, or *residue*, for a small square will be zero. For any large path, the unwrapping error is simply the sum of the residues enclosed by the path. The next stage of the algorithm is similar to that previously described by Huntley, as branch cuts are constructed joining groups of residues so that the sum of all the residues joined is zero. This allows the final result of the unwrapping process to be completely free of inconsistencies providing that unwrapping never crosses a branch cut.

Bone also proposes an algorithm, which, rather than relying on branch cuts joining the residues, utilises a mask. A bit-map mask is constructed which is overlaid on the phase field to prevent the unwrapping algorithm from following any path which could lead to inconsistencies. This simplifies the process as only local information is used. It is then possible to calculate further differences from the locally unwrapped phase. The phase is unwrapped clockwise around a given point (i,j) in a clockwise manner, as shown in figure 4.7.

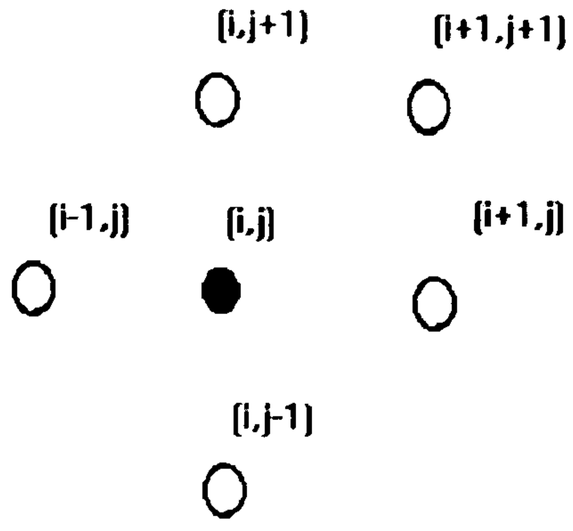


Figure 4.7: Unwrapping in a clockwise manner.

Once the mask has been defined, the phase in the regions defined by the mask has to be unwrapped. Unwrapping is carried out using a standard recursive flood fill algorithm.

Brown[8] describes a method for unwrapping phase distributions developed at the Ford Motor Company as part of a proprietary Computer Aided Holography (CAH) system. The philosophy behind the system was to make it easy for an operator to create binary masks to guide the unwrapping process, to automate the mask creation as much as possible, to use an algorithm which could process an arbitrarily complex structure, to achieve processing time of the order of minutes and to use 2-D interpolation to fill any small bad spot regions. The stages of the CAH process with which the work is concerned are:

- Calculation of the 2π phase change map,
- Creation of the binary bad spot mask,
- Setting of "seed points" in the unwrapped pixel mask,
- Unwrapping of the phase map using the mask, seed point and rectilinear path algorithm and
- Removal of bad spots.

Stephenson *et. al.* describe a technique which uses data validation routines to ascertain the faults present in a wrapped phase distribution[9]. The

image of interest is divided into tiles, each of which is subjected to the following tests:

- Perimeter test: Schafer's algorithm is applied around the perimeter of each region. If a continuous phase wrap is present in the region, there should be an equal number of wraps in each direction around the perimeter.

- Tile modulation test: This is carried out to check for low-signal modulation in the image, a frequent cause of apparent phase noise.

- Wrap continuity test: This checks for spot noise or wrap bifurcations. If a point is found to be a wrap, its neighbours are tested to ascertain whether they are also wraps.

- Template matching test: For a particular region, templates are constructed showing all likely positions of wraps, which are dependent on the position of the fringes in the original image. A good region will have a perfect match with one wrap position template.

When the validation tests have been carried out on the regions, each tile is then assigned to one of three categories:

1. Cleanly unwrapped tiles with no detected problems
2. Tiles with a minor problem that could easily be remedied
3. Tiles with one or more serious defects

This series of simple tests increased the robustness of the unwrapping algorithm.

There are obviously drawbacks associated with all of these algorithmic methods, the most prominent being those of time and computational complexity. It is also evident that there is no "generic" phase unwrapper.

There will always be complex wrapped phase distributions which an algorithm will fail to unwrap adequately. The modern digital computer is extremely adept at performing complex mathematical calculations very quickly, but is not as competent at tasks at which the human brain excels, such as pattern recognition. If it is presumed that the basic problem of phase unwrapping can be considered as a kind of pattern recognition problem, then it can be postulated that a parallel processing structure such as the brain is better suited to its solution. An experienced human operator will be able to decide which data are noise spikes or other discontinuities and which are phase wraps with far better accuracy than can be achieved by the point-by-point analysis by a computer. With a human operator, part of the analysis is likely to be based on intuition, rather than on mathematics alone. This gives rise to an interesting question: does a machine exist which can be "taught" what is a phase wrap and what is a noise spike?

The depth of research into Neural Networks (NNs) has already been discussed in chapter 3. It has been shown that this particular computing paradigm is, by its very nature, ideally suited to recognition problems. The idea of using NNs for pattern recognition would suggest that the potential exists for a suitably trained network to act as the basis for a robust phase unwrapping system.

Kendall and Hall[10] have carried out work which, although not directly related to the phase unwrapping problem, describes the use of NNs for various problems in image processing, including edge detection and texture classification. For edge detection, a multilayer perceptron with a 2x2-pixel window is scanned across the image. The NN is trained on an image of shaded circles with added Gaussian noise and well defined edges. Testing on a more "natural" image (i.e. one with less prominent edges) showed the technique to be faster and more accurate than a conventional Laplacian operator.

The remainder of this chapter deals with research carried out into the use of neural networks for phase unwrapping.

4.2 Phase unwrapping by neural network.

To date little research has been published which deals with this particular approach to the problem. The only work of which the author is aware is that of Takeda [11]. This approach uses a large network, which performs a single analysis of an entire image. Consider a wrapped phase distribution as shown in figure 4.1. At any pixel i, the phase value can be represented by ϕ_i. If the offset phase value which must be added at the ith pixel to unwrap ϕ_i is represented by f_i, the unwrapping problem can be mapped onto a Hopfield network by the function:

$$E = \Sigma \{[(\phi_{i-1} + 2\pi f_{i-1}) - (\phi_i + 2\pi f_i)]^2 + [(\phi_{i+1} + 2\pi f_{i+1}) - (\phi_i + 2\pi f_i)]^2\}$$

Each image is analysed with a network consisting of five neurons per pixel. This assumes that a maximum number of five wraps is going to be encountered in the image. This may well be the case for a fringe pattern subjected to the FFT method with frequency shifting, but if frequency shifting is not used or the fringe pattern is subjected to the Phase Stepping method, the total number of wraps is likely to be far in excess of five. With such a large network, both large training sets and long training times are likely to be required. Also, the volume of data involved is such calculations will be very large. If a CCD camera is used whose array utilises 512x512 pixels, the resulting image has 262,144 points of data. If five neurons per pixel are used, the network will have to contain in the region of 1.25 *million* neurons. Due to the large number of neurons and, therefore, the high processing times that would be required, it was thought that an approach may be possible using a NN to assist in the *regional* unwrapping of phase images.

The approach outlined in this thesis involves the use of small networks, which are used to analyse one small region of the wrapped phase distribution at a time and effectively build up an entire unwrapped phase map from a

large number of simpler calculations. It was felt that it would be beneficial to attempt to reduce the complexity of the approach to the problem using a NN. Previous algorithmic solutions have been most efficient when reliant on a regional approach, thus limiting error propagation through the entire image. If, however, a NN solution is investigated which continues to use a global method of analysis, some risk of error propagation is likely to be reintroduced. The previously described NN approach appears to be reasonably robust, however, the assumptions which are made will greatly reduce its adaptability. Although it would appear impossible to produce a "generic" phase unwrapper capable of reconstructing every phase distribution it encounters, it may well be possible to create an extremely robust unwrapper if certain points are borne in mind. These are:

1. Parallel processing of an image will produce better results than a point-to-point algorithm.
2. Regional analysis will cause less error propagation than global analysis.
3. Wraps are likely to occur in any direction.
4. The number of wraps will only be reduced if Fourier analysis with frequency shifting is used.
5. Generally, the number of wraps is limited by the number of fringes present in the original image and the resolution of the wrapped phase image, therefore an unwrapper should be able to analyse any given number of wraps.

The previously described NN unwrapper[11] took into account only one of the above points, therefore an approach was formulated which addressed all 5.

The use of a NN addresses point number 1, as by their nature NNs use a parallel architecture. A system was, therefore, required which not only employed a parallel processing paradigm, but also relied on local analyses of phase data, took into account wraps which occurred in every direction and was not limited to the number of wraps it was capable of detecting. The remit of this chapter is to describe a possible solution to the phase unwrapping problem which satisfies all of these criteria.

Initial experimentation

The first problem investigated was that of one-dimensional phase unwrapping. This is the simplest method encountered and involves unwrapping phase distributions in a single line of pixels. An example of this has been shown in figure 4.1. Although this was not entirely representative of the problem as a whole, it was thought that it was a good starting point from which to evaluate the validity of using a neural network to unwrap a phase distribution.

A standard image is 512 by 512 pixels. It was therefore decided to work with 512 data points or fractions of this. Experimentation began with a simple representation of the problem: a single line of 64 phase values containing 4 wraps. The first task was to train a neural network to recognise the positions of the four wraps in the distribution.

A neural network was constructed using the minimum configuration of one input neuron for each value and one out output neuron for each corresponding response. The resulting network consisted of 64 input neurons, 64 output neurons and a single hidden layer of 64 neurons. The network architecture was backpropagation with a delta learning rule and sigmoid transfer function. A complete explanation of this architecture is given in chapter 3. Training data consisted of 70 sets of 64 phase values, each containing 4 wraps at various positions. The first training sets were constructed from simulated data; that is data were not taken from real wrapped phase distributions, but created artificially to avoid the introduction of noise or other extraneous data at this early stage in the experimentation. A sample of the simulated data used for the initial training set is shown in appendix 1. The first training exercise consisted of randomly presenting the network with 60 training vectors. The behaviour of the network throughout the training period was observed and, when training was complete, the network was tested. Again, the testing procedure is discussed in detail in chapter 3. A test file, again using simulated data was presented to the

network. The desired and actual outputs from the network were automatically written to a results file which was then analysed to ascertain how the outputs varied.

During training, the RMS error of the network remained high. Inspection of the results file showed that the network was attempting to guess the position of the wraps, but appeared to be unsure as to their exact positions. The results suggested that either not enough training data were present in the training file, or problems were arising with the actual presentation of the data to the network. An attempt was made to rectify the problem by addition of further data sets to the training file, bringing the total number of training vectors up to 100. The network was initialised and re-trained using the extended training set. When training was complete, the network was tested. It appeared that the addition of further training data had failed to have any noticeable effect on the results. During training, the RMS error still failed to converge and the network still appeared to be unsure of the exact position of the wraps, as shown in figure 4.8.

As the number of training data was having no effect, it was thought that the problem may have been in the presentation of the existing data. As described in Chapter 3, when using the delta learning rule, it is important that the input data set is well randomised. The NeuralWorks Reference Guide [12] states that "Well ordered or structured presentation of the training set often leads to a failure to converge". The cumulative-delta learning rule goes some way to preventing this type of problem. This learning rule accumulates weight changes over several presentations and applies this cumulative result at once, giving a pseudo-random presentation of the data. Changing the learning rule to cumulative-delta appeared to have little beneficial effect on the results. The training period was extended to investigate whether the network was simply being under-trained.

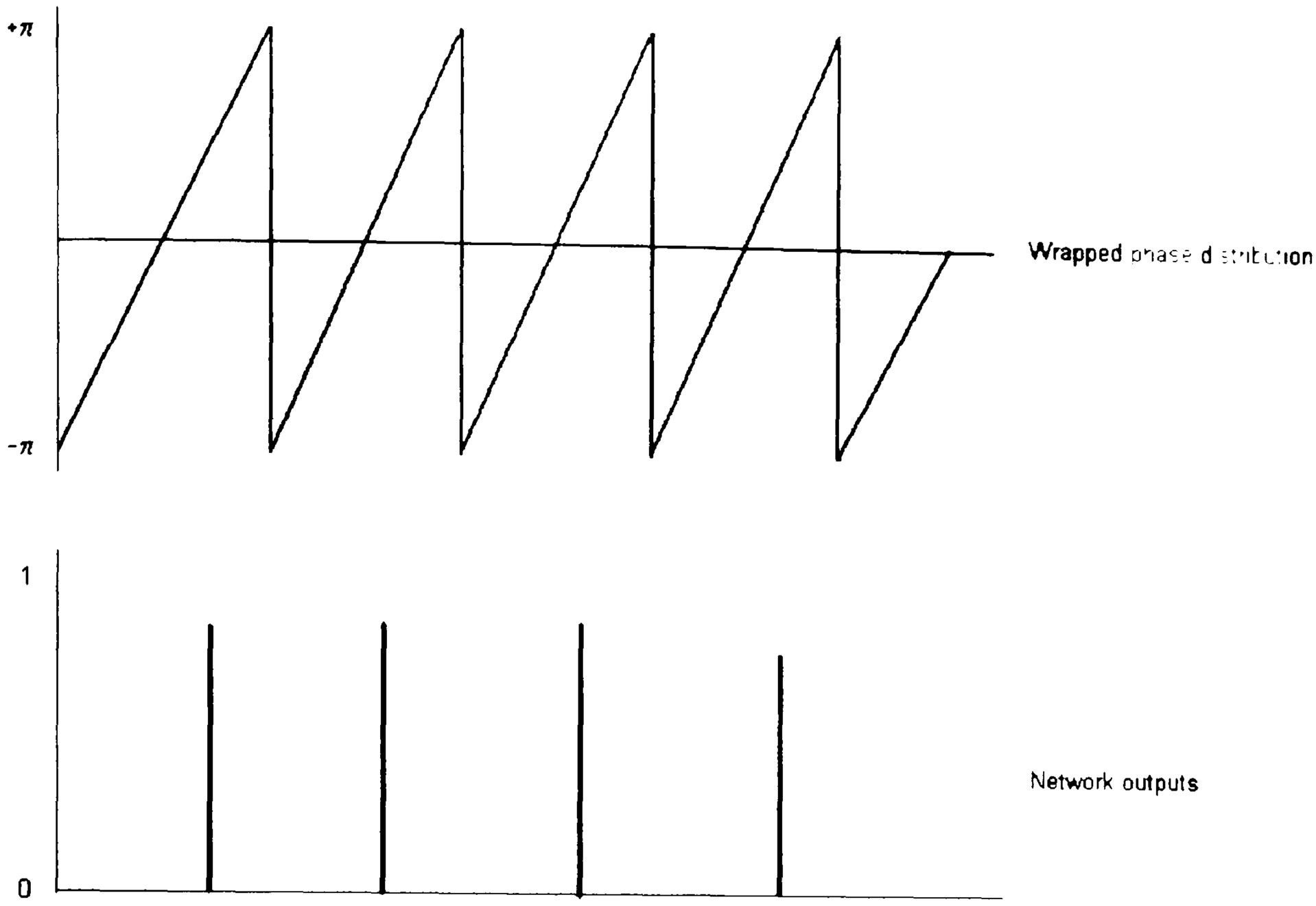


Figure 4.8: Wraps erroneously detected by initial network.

In order to ascertain whether the RMS error would converge given a longer training period, the total number of presentations of data before cessation of training was increased. Again, the behaviour of the network during the training period was closely observed and, when training was complete, the network was tested as previously described. Increasing the number of presentations to 10,000 had no effect on the RMS error of the network, which still failed to reach its convergence criterion. The results for the increased training were even more vague than those previously described, as the bands of indecision displayed in the results file were much wider, indicating that the opposite effect to that anticipated had been achieved and that the network may have been over-trained. The results are shown in figure 4.9.

Due to the limited success of the experimentation at this point, it was decided to concentrate on a much simpler approach to the problem. Instead of

beginning by attempting to unwrap entire rows or columns of phase data, a network was configured to analyse a much smaller region of phase data, dealing with only a single wrap. The philosophy behind the approach was to create a network that would have a small input "window" which could be convolved with an image to detect the presence of phase wraps. It was thought that if a relatively simple network was used, it would be possible to use more complex learning strategies which would make the approach much more computationally efficient. At this stage, experimentation was still being carried out to ascertain whether a NN approach was suitable for wrap detection and not to attempt to perform a complete and successful phase unwrapping process.

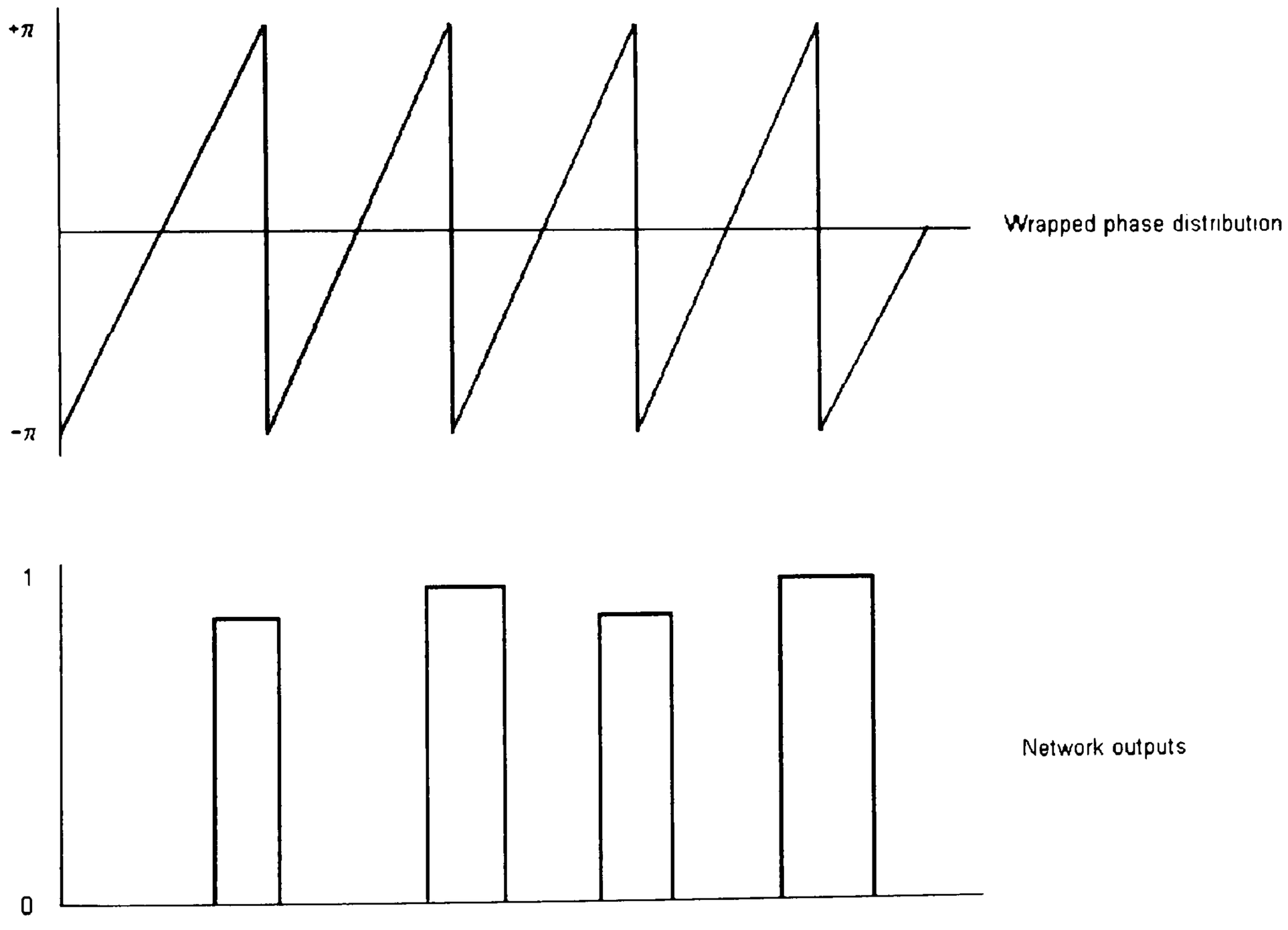


Figure 4.9: Network outputs showing bands of indecision.

The first network utilising the simpler approach consisted of a backpropagation architecture, employing the cumulative-delta learning rule and sigmoid transfer function. This configuration was employed as backpropagation networks have been proved to be a reliable method of recognition in previous experiments. The use of a cumulative-delta rule was

again used to ensure satisfactory random presentation of the training data. The physical configuration of the network relied on the minimum number of neurons per pixel to attempt to keep the initial computational complexity to a minimum. The number of pixels to be analysed was reduced to six. This was thought to be around the minimum number of pixels which it would be possible to analyse and detect the presence of a phase wrap without risking reducing the number to a point where the network was acting as a very computationally expensive point-to-point algorithm. The input layer consisted of six neurons, each corresponding to a single pixel, and both the hidden and output layers each contained six neurons. A schematic diagram of the network is shown in figure 4.10. The network was designed to recognise the position of a single wrap in a six-pixel region of a phase distribution and respond by firing a single neuron in the corresponding position in the output layer. The training set for the network was constructed using artificial data and consisted of 100 vectors, each of 6 phase values with no more than one phase wrap occurring at a point within that vector. A number of "null" examples were also included, that is, vectors containing *no* phase wraps.

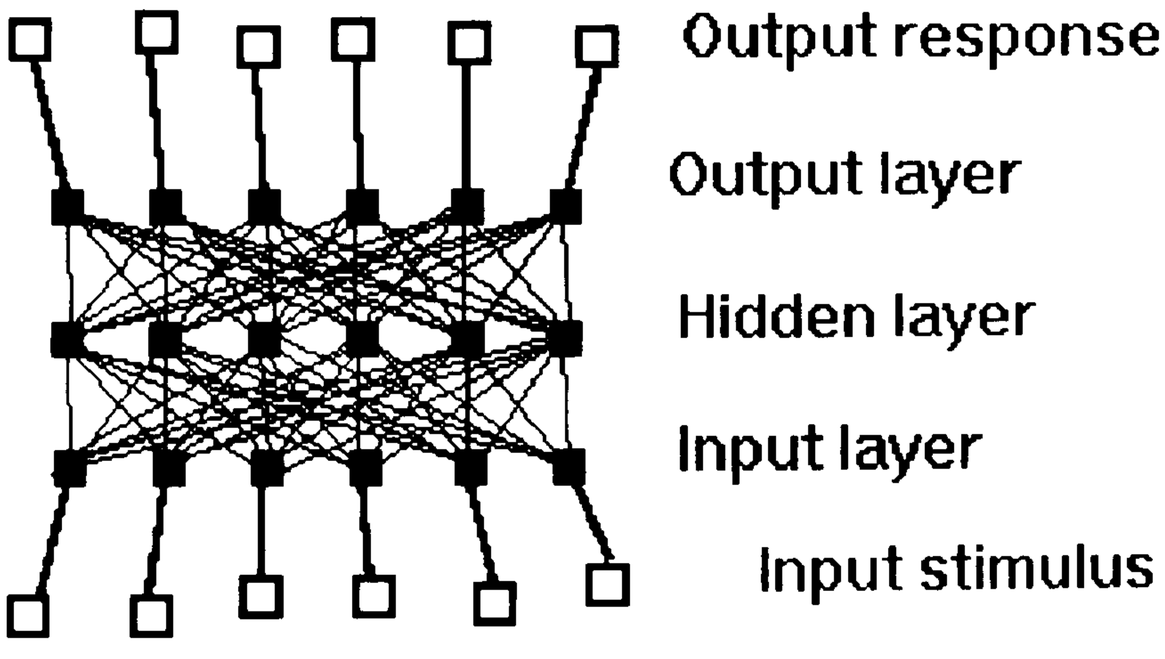


Figure 4.10: Schematic diagram of the 6-input network.

The data was simulated for the first training set in order to ensure no noise was present in the initial training set. Nominal phase values showing a step of plus or minus 2π were used and no noise or false wraps were present. The first training set was constructed as described in chapter 3. As the network contained six input and six output neurons, each training vector contained twelve values, the first six being phase values and the remaining six being either a zero or a one depending on the position of the wrap. An example of the training data for the six input network is shown in appendix 1.

The network was trained and tested as previously described, with its behaviour being closely monitored throughout the training process. Random presentation of the training data caused the network's RMS error to decrease almost to its convergence criterion and the weight histogram showed the spread of weights across the network to approximate a Gaussian distribution. After 10,000 presentations, although the set convergence criterion had not been reached, training was stopped and the network was tested. The test set was again constructed of artificial data and presented to the network only once. The results of the test showed that the network had accurately identified approximately 75% of the phase wraps. In the correct cases, when the desired output was one, the actual output value was in the region of 0.90 to 0.97 and when the desired output was zero, the actual output was approximately 0.02 to 0.05. These values were improved by further training. It was found that by 20,000 presentations of training data, the RMS error value had converged, but to a point slightly in excess of the set convergence criterion. To allow for this, the threshold was raised to 0.005. This was to ensure that training ceased at a point before over-training could occur.

When unwrapping is carried out on a real wrapped phase distribution, it is highly unlikely that the wrap density will be uniform. An ideal and an actual wrapped phase distribution are shown in one dimension in figure 4.11. To allow for this variation, a training set containing data with non-uniform wrap densities was constructed to train the network further. Again, all training data were simulated and the same network architecture and experimental

procedures were adhered to. The results shown were comparable to those of the previous experiment, with a success rate of approximately 75% being achieved.

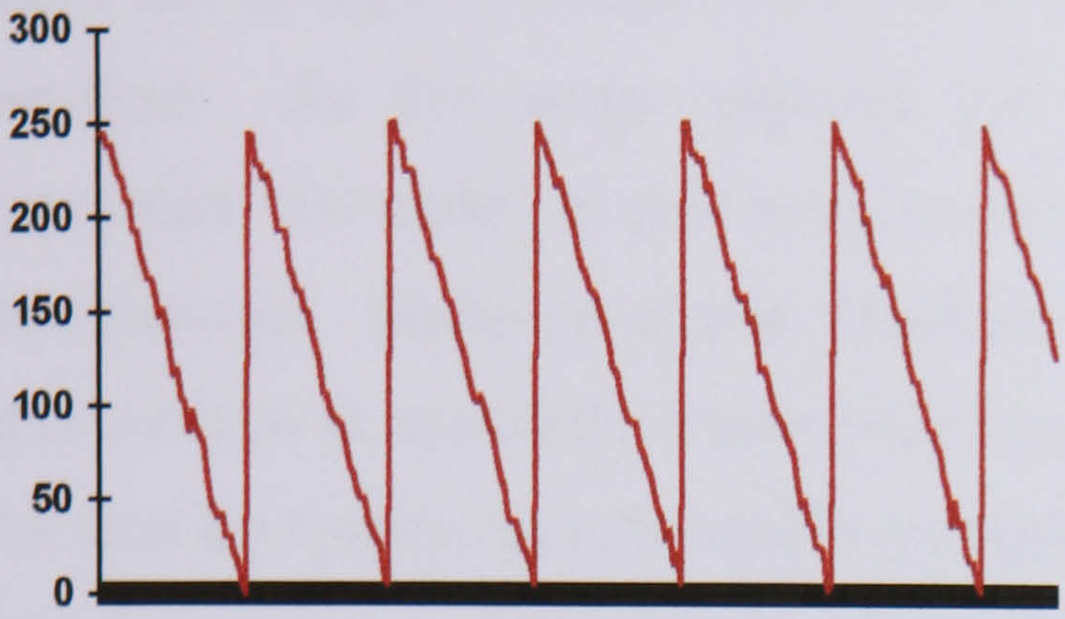


Figure 4.11(a): An ideal 1-D wrapped phase distribution.

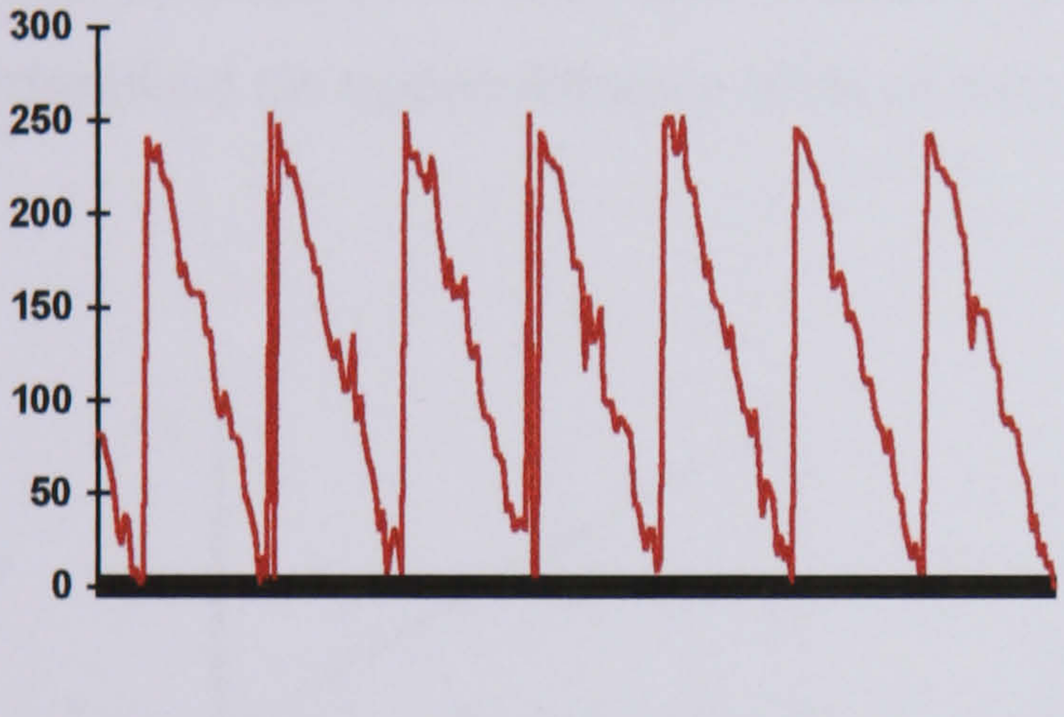


Figure 4.11(b): A wrapped phase distribution containing spot noise.

In both cases, when identification of a phase wrap was unsuccessful, one of three things had occurred:

1. An output neuron directly adjacent to the wrap had fired.
2. An output neuron fired when no wraps were present.
3. An output neuron had failed to fire when the test set showed a wrap to be present at the very end of the pixel "window",

The first two of these phenomena can easily be explained in terms of bad data or incomplete training. However, the third occurrence poses an altogether different problem. It appeared that the training data was trying to teach the network to respond to a discontinuity which was not actually

present. As wrap detection necessarily involves comparison of two adjacent phase values, it is, therefore, impossible to detect a wrap occurring directly on the edge of a pixel window. Consider the wrapped phase distribution shown in figure 4.12. As the six pixel window is moved along the distribution from left to right, it encounters the wrap, which moves left in relation to the window. As the wrap reaches the furthest left pixel of the window, it becomes "invisible" to the window, as there is no further pixel to its left for comparison. Because of this "disappearance" of the wrap when the edge of the window is reached, it was necessary to adjust the training data to allow for this by removing references to phase wraps which occurred under these circumstances. Using the same network architecture and methods as previously described, the network was re-trained and tested. Analysis of the results showed a dramatic improvement, with phase wraps being correctly identified on approximately 95% of occasions.

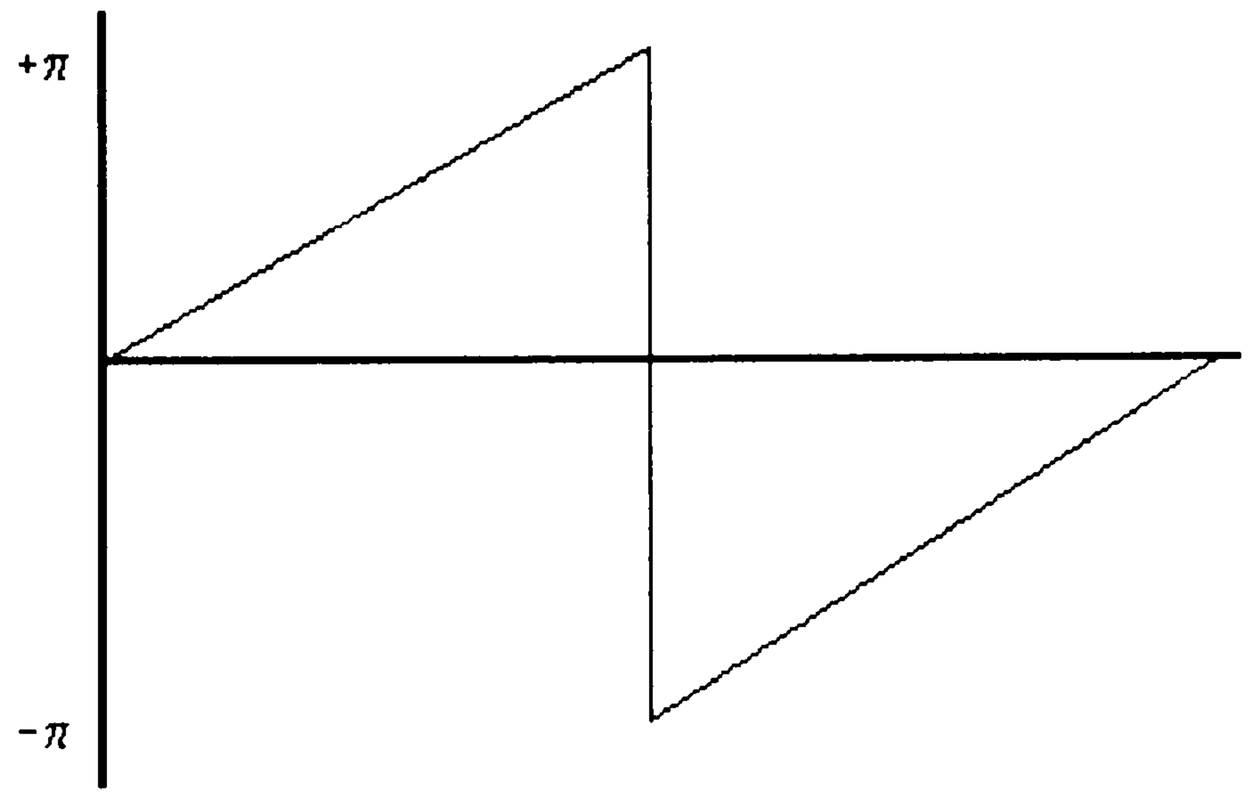


Figure 4.12: A single phase wrap.

Comparison of training methods

The next experiment was to compare methods of training. So far, the networks had been trained using entirely artificial, noise-free data. In order to investigate how the network would behave when presented with more realistic data, a new training set was constructed.

The training set consisted of data in the same format as previously described, but this time taken from real wrapped phase distributions. The distributions were very low-noise and were of the type shown in figure 4.13. This sample phase distribution is the result of projecting a cosinusoidal fringe pattern onto a flat, matt white surface and subjecting that fringe pattern to Fourier analysis. In this image, all the phase wraps occurred in the same direction. A single line of phase values was taken at various points in the image and the data from these lines adapted to train the network. Each data set was examined and the positions of any phase wraps were identified. Desired output values were added to each training vector showing where in the data set the wraps occurred. An example of the training data is shown in appendix 1.

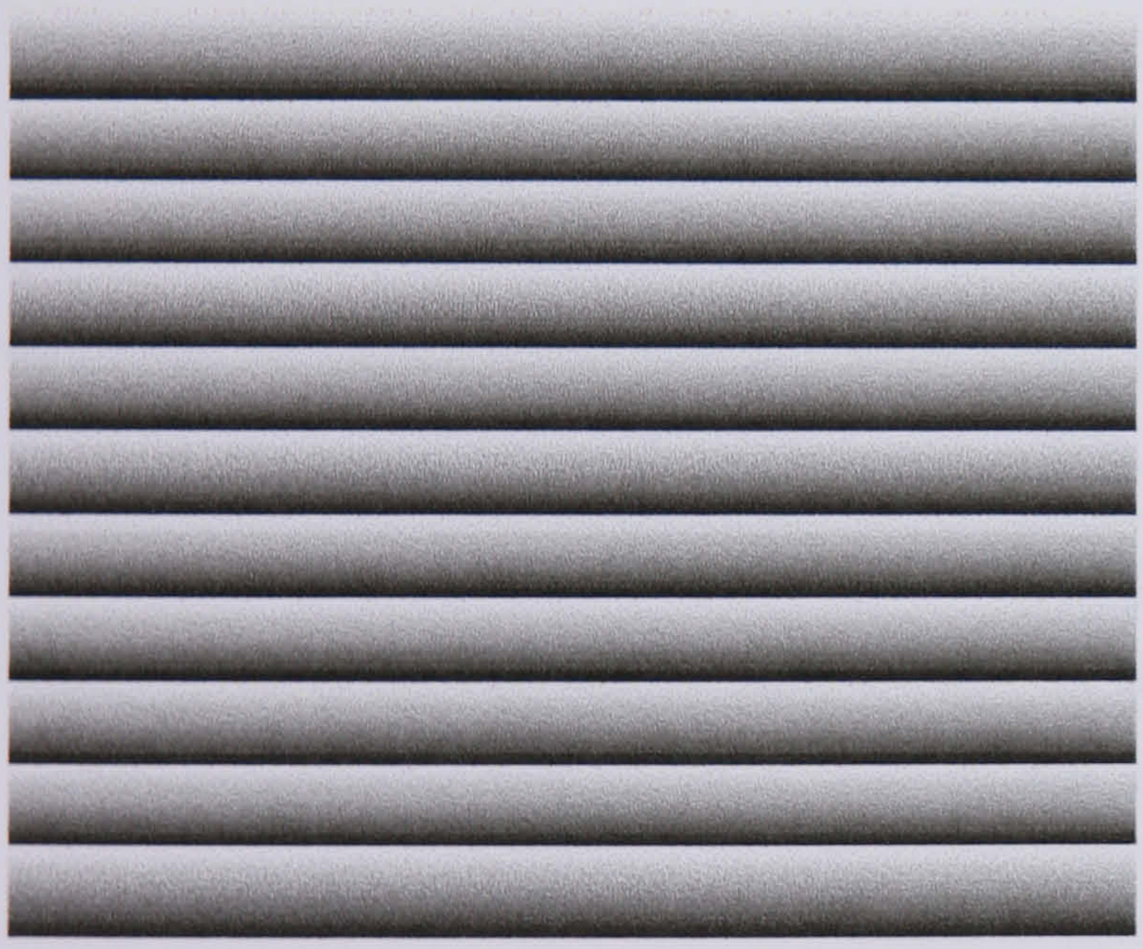


Figure 4.13: 2-D phase distribution used for training data.

4.2.1 Trimming the hidden layer.

So far, all experiments had been conducted using a network consisting of the same number of hidden neurons as input and output neurons. If a network is too large for the amount of data with which it is presented, it can become "confused" and give multiple answers for a single problem. With this phenomenon in mind, the number of hidden neurons was trimmed to leave a total of two and the network was retrained. A relatively short training session

was tried initially. After 1000 presentations of the training set, no effect on RMS error or weight distribution was noticed and the results file showed little more than random data. The network was retrained with 10,000 presentations, but there was no noticeable difference in the RMS error graph. The weight histogram showed more of a tendency towards a Gaussian distribution, but there seemed to be a large number of weights with a value of -8. When the training was complete, it appeared that the two hidden neurons were "locked on", i.e. their values were both at a maximum. Inspection of the training file showed a very poor response. Only around a tenth of the phase wraps had been correctly identified. It was noteworthy that the only wraps which had been correctly spotted were those which occurred at position number five. The number of hidden neurons was increased one by one and the network retrained for each additional neuron. As the number of hidden neurons increased, so did the network's ability to identify phase wraps until the accuracy previously described was repeated when six hidden neurons were used.

In order to investigate the possibility of improving accuracy by adding *more* hidden neurons, a new experiment was carried out using a single hidden layer of twelve units. The network was trained as previously described. The RMS error graph showed very erratic changes during the training period and on completion of the training, ten of the twelve hidden neurons were "locked on". Inspection of the results file showed that again, approximately one tenth of the phase wraps were being correctly identified. However, in this case, no noticeable pattern of which wraps were being identified emerged. Training was continued, with hidden neurons being disabled one by one, until the original configuration was arrived at. This layout of six input, six hidden and six output neurons showed the greatest ability to correctly identify the positions of wraps in a phase distribution.

Further experimentation was carried out with hidden layer configurations. A network with two separate hidden layers, each containing six neurons, was constructed. The network was trained in the same way as previously

described, its behaviour monitored and, after training, tested and the desired and actual outputs compared. During training, the RMS error remained as for the previous experiment, but the a slightly larger spread of connection weights was evident from the weight histogram. When training was complete and the network fully tested, the results file was analysed and showed that only approximately 50% of wraps were being correctly identified. The initial conclusion to be drawn from this use of a second hidden layer is that too many processing elements within the network can adversely effect its performance. The NeuralWorks Reference Guide[12] explains that if too many processing elements are employed in a relatively simple network, some of the information may be "lost" as values are spread throughout the network.

4.3 Experiments with unsupervised learning

Although the experiments with supervised learning as previously described proved successful, experiments were also carried out with unsupervised learning in order to compare the two techniques. As discussed in chapter 3, unsupervised learning uses only a set of input values and the network is left to calculate its own outputs. For this series of experiments, the same training data were used as for the previous experiments. Due to its success in supervised learning cases, phase data taken from real images were used.

4.3.1 Learning Vector Quantisation (LVQ)

An LVQ network was configured with six input neurons, six output neurons and a Kohonen layer containing six neurons. The existing training data was presented to the network randomly 10,000 times. Initially, the weight histogram showed weights present with only minimum and maximum values and the RMS error changed apparently randomly. However, after approximately 2000 presentations, the spread of weights began to show the

beginnings of a Gaussian distribution and the RMS error value settled and began to fall. At 8010 presentations, the convergence criterion was met and training ceased.

Initial analysis of the results file proved promising. Approximately 50% of the phase wraps were correctly identified. One major problem was noted with this configuration, however. The network had difficulty in identifying the situation when no phase wraps were present. In this case, whenever the network encountered a vector with no wraps, the fifth output neuron fired, apparently identifying a wrap which did not exist.

Experimentation was again carried out with the number of hidden neurons in an attempt to improve on the wrap detection rate. By the nature of the LVQ network it is not possible to have fewer hidden neurons than input neurons and the number of neurons in the Kohonen layer must be a multiple of the number in the input layer. With twelve neurons in the Kohonen layer, the convergence criterion was met after only *one* presentation of the training data.

Analysis of the results file showed the output data as apparently random, identification of wrap positions being by chance. The network was returned to its original state, initialised and retrained. The convergence criterion was met after only 20 presentations of data. The results file yielded the same data as for the previous experiment. Initialising and retraining the network on further occasions showed that convergence criterion was met after fewer than 20 presentations of data, and each time, the results appeared to be completely random.

4.3.2 Hopfield networks: the classical approach

Initial experimentation with a Hopfield network gave unsatisfactory results. A network was configured to work with the same data in an attempt to solve the same wrap detection problem consisting of six input, six hidden and six

output neurons. Figure 4.14 shows the Hopfield network described in this section. When training commenced, the RMS error immediately exceeded the upper limit of the graph and all the processing elements locked "on". The network remained in this state, so training was terminated and the network was tested. This caused all the processing elements to lock "off". The results file showed that when a wrap was present at any point, all the output neurons fired. However, when no wraps were present in the input data field, no output neurons fired. The network appeared to be able to tell that a wrap was present in the data field, but could not identify its position. The network was effectively operating as a logical "OR" gate.

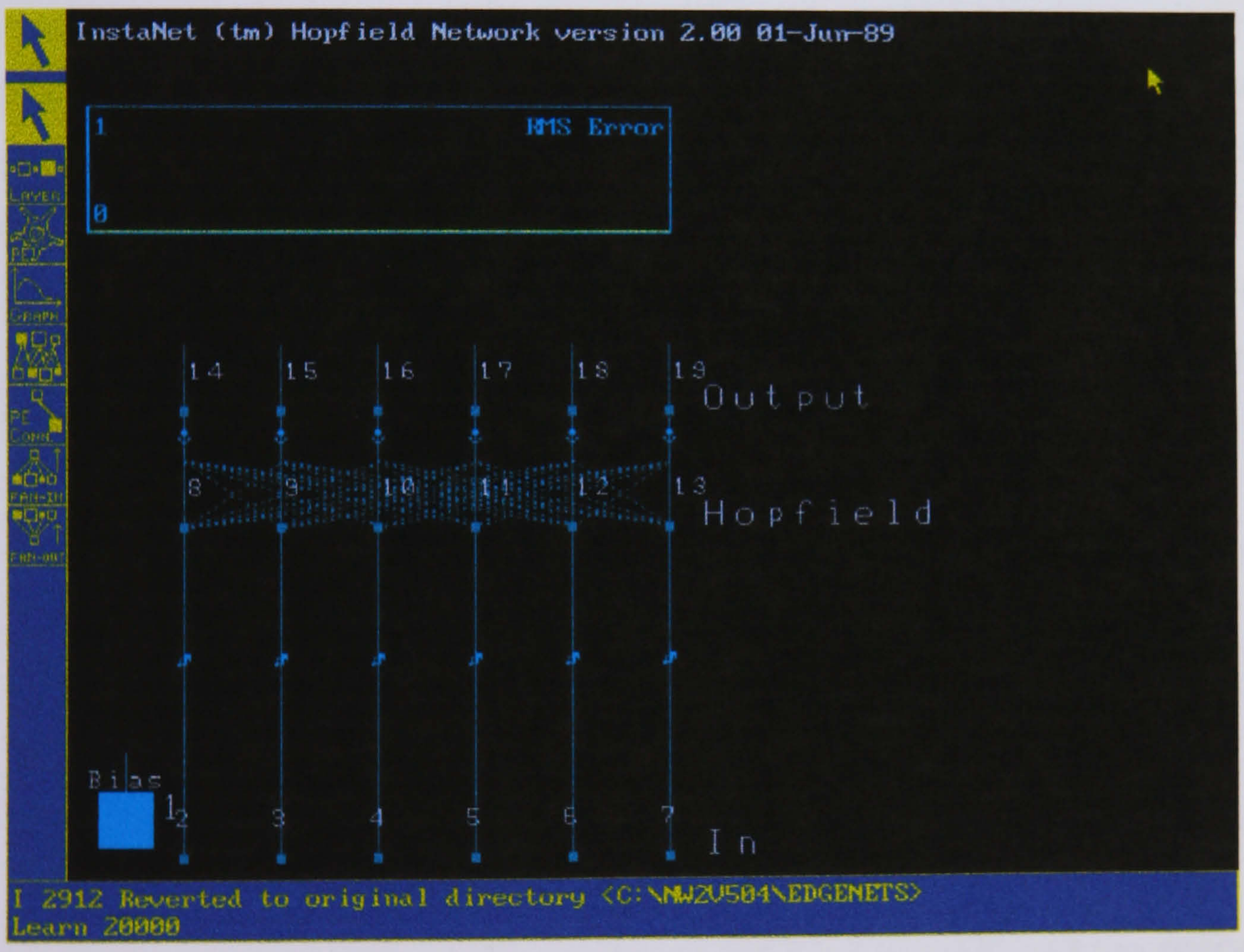


Figure 4.14: 6-input Hopfield network.

4.4 Comparison of techniques

The above description of experimentation using both supervised and unsupervised techniques shows that the two approaches yielded quite different results. Supervised learning appeared to show more potential for the detection of wraps in a phase distribution. The use of the

backpropagation paradigm gave the best results for the 1-D phase wrap detection problem.

Appraisal of one-dimensional phase unwrapping

So far, experimentation has shown that a small, backpropagation neural network is suited to the task of identifying the position of phase wraps in a one dimensional wrapped phase distribution. To facilitate a full unwrapped phase distribution, it was necessary to extend the experimental work to include the updating of phase values to achieve the desired end result. Although the network described performed satisfactorily when a simple binary output was required, it was thought that training the network to update the phase values when necessary would be unnecessarily complex. It was, therefore, decided to rely on a standard method of programming to perform the actual addition and subtraction of 2π required to produce the final unwrapped phase distribution. The fully trained and tested wrap detection network was FlashCoded into standard C and the following procedure implemented to unwrap a one-dimensional phase distribution.

1. The six-pixel window was aligned with the first six pixels of the phase distribution.
2. The phase values were presented to the network which reported the position of any phase wraps.
3. If a wrap was present within the window, the corresponding values of the original phase distribution were updated accordingly.
4. The wrap detection window was indexed by one pixel and the procedure repeated.

This was continued until the complete phase distribution had been analysed.

The code for this operation is shown in appendix 2.

Comparison with Schafer's Algorithm.

Two 1-D phase distributions were used for this experiment, one consisting of noise-free data and one containing several spot noise spikes. The two distributions were taken from real phase images which were the result of a fringe pattern which had been subjected to a phase-stepping algorithm. The two distributions are shown in figure 4.15.

Each of the two wrapped phase distributions was unwrapped using the two methods. Figure 4.15(b) shows the result of distribution 4.15(a) after unwrapping using Schafer's Algorithm and 4.15(c) the result of unwrapping by the backpropagation network. In this case, as the wrapped phase distribution is free of noise, the results are the same. The task is a straightforward one with no risk of error propagation in the final result.

The difference in the two methods is shown more clearly in figure 4.16. Here, the wrapped phase distribution contains several prominent spot-noise spikes which are likely to cause errors in the final result. Figure 4.16(b) shows the result of unwrapping using Schafer's algorithm. Here, the algorithm has treated the noise spikes as phase wraps and updated the phase values by 2π in each case. The result is that errors are propagated through the unwrapped phase distribution, meaning that instead of a smooth distribution like the previous examples, several steps are present. Figure 4.16(c), however, shows the same phase distribution when unwrapped using the backpropagation network. Here, the noise spikes have been effectively ignored and the correct unwrapped phase distribution had largely been retained. The noise spikes can then be filtered out by some post-processing operation.

Considering the relative success of the approaches described above, it was decided to further investigate the use of backpropagation neural networks to perform a complete, two-dimensional phase unwrapping operation.

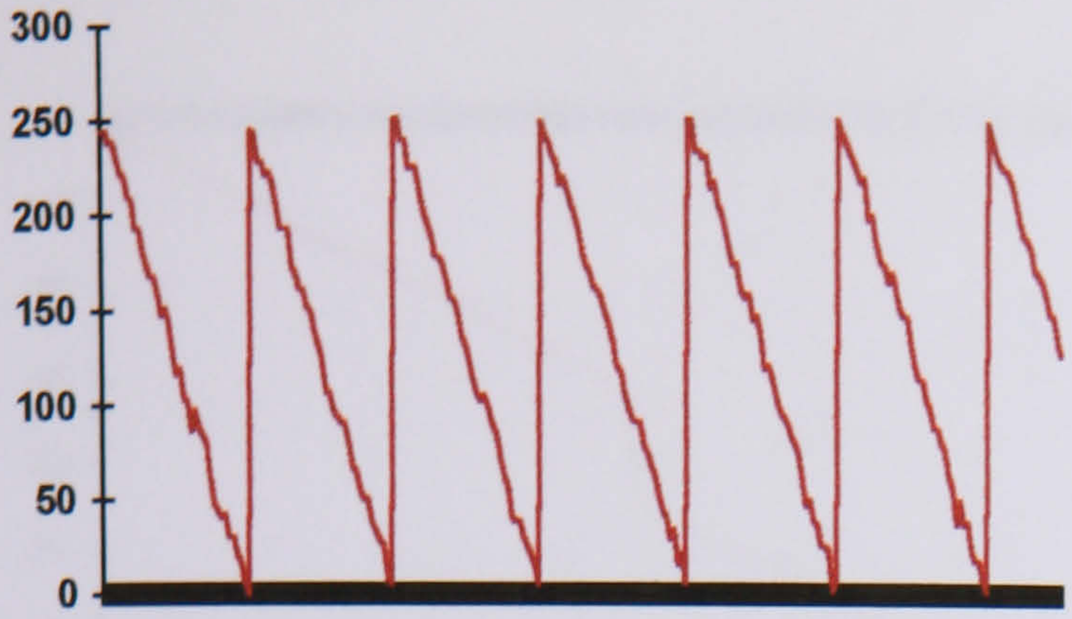


Figure 4.15(a) Wrapped phase distribution

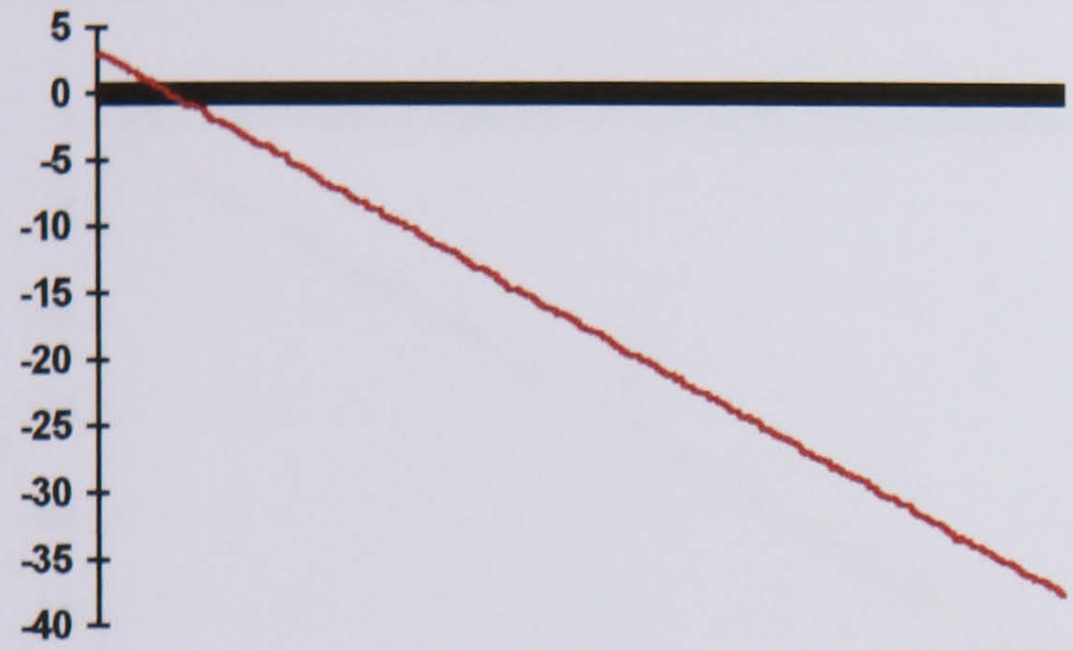


Figure 4.15(b) Distribution unwrapped using Schafer's algorithm

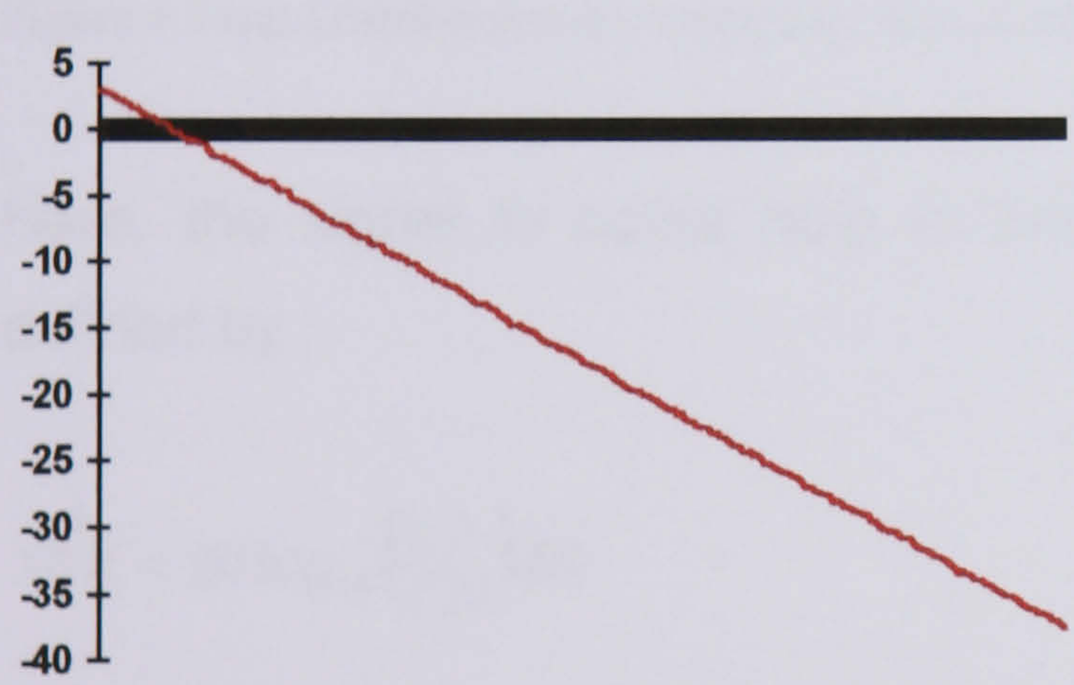


Figure 4.15(c) Distribution unwrapped by neural network

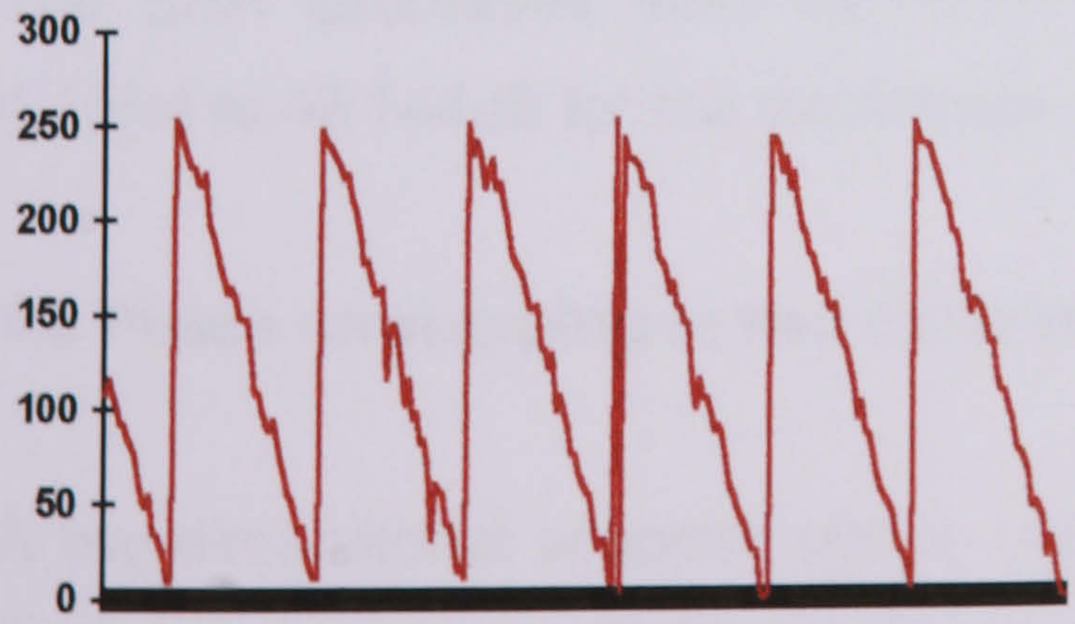


Figure 4.16(a) Wrapped phase distribution with spot noise

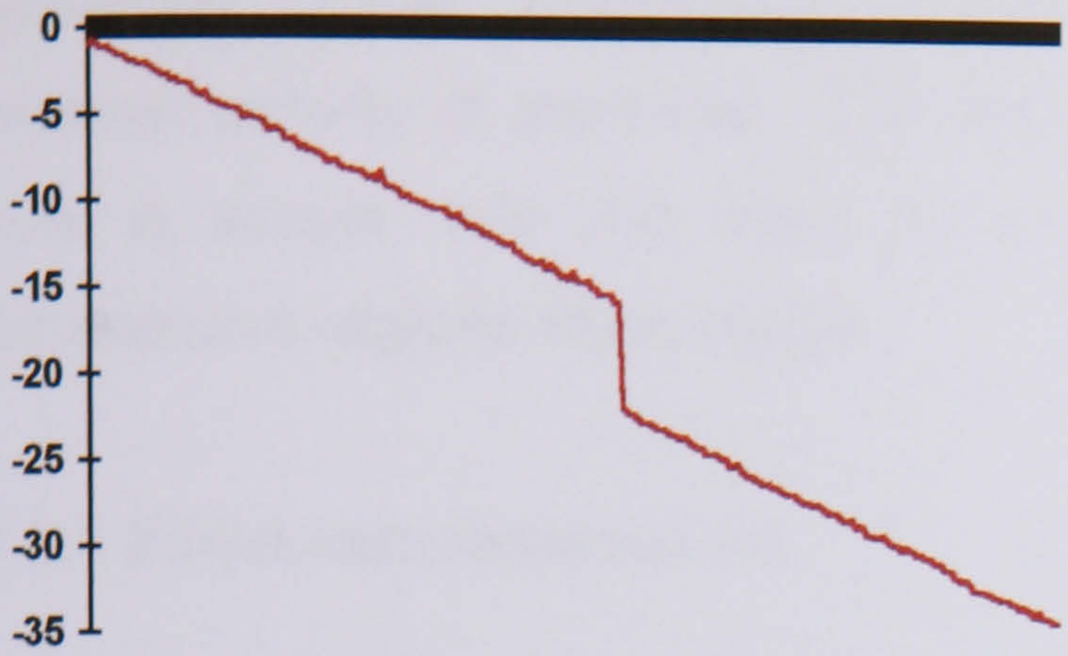


Figure 4.16(b) Distribution unwrapped using Schafer's algorithm.

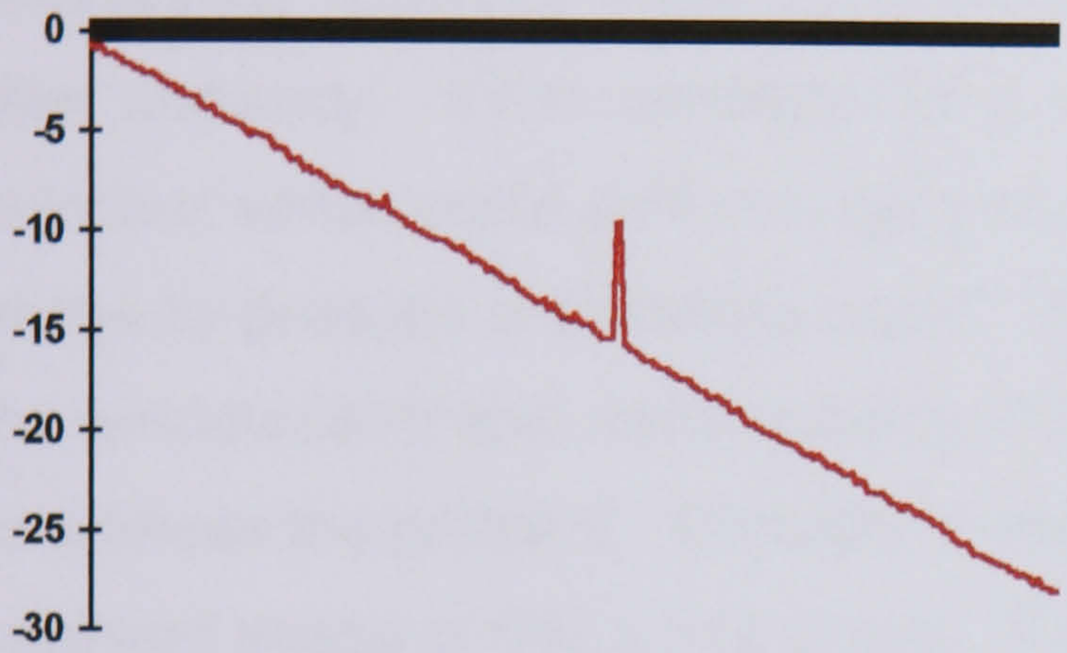


Figure 4.16(c) Distribution unwrapped by neural network.

Here, the signal to noise ratio in the wrapped phase distribution can be defined by

$$SNR = 20\log_{10}\left(S/N\right)dB$$

where S is signal and N is noise values of the distribution.

The SNR decreases from 48.165dB for the distribution shown in figure 4.15(a) to 42.144dB for the distribution shown in figure 4.16(a).

4.5 Phase unwrapping in two dimensions

A two-dimensional wrapped phase distribution must be thought of as more than just a series of lines. It could be tempting at this point in the experimentation to simply extend the previously described technique to

unwrap a series of rows. Although this may well work on relatively simple, noise-free phase distributions, it does not take into account fully the interconnectivity of the rows. The technique needs to be extended further, from a simple 1-D line input to a system capable of analysing two-dimensional regions of an image.

4.5.1 Initial experimentation

Experimentation was carried out into how the success of the earlier 1-D unwrapping networks could be transformed into 2-D systems and maintain their accuracy. Initial experiments concentrated on use of an **n**x**n** square "window" which could perform regional unwrapping and be convolved with an image to produce a complete result. The need was to optimise the size of the window used and, consequently, the size of the neural network employed to address the problem. Consider a wrapped phase distribution defined by a standard image of 512 x 512 pixels. There are two extremes associated with analysis of a distribution of this size. It would be possible to attempt to analyse an entire image with a single, large network. This problem has been addressed by Takeda[11] and is described earlier in this chapter. There are a number of drawbacks associated with this approach. A 512 x 512 pixel image contains 262,144 pixels. To analyse the entire image in one pass will require use of a network with at least 262,144 input neurons. With a network of this size comes the associated problems of training. The training set will require large amounts of data. If 100 vectors are used, a total of 26,214,400 input values alone will be required. Each image used in the training set will have to be individually analysed to provide desired output responses for the network. If it is assumed that one output response is required for each input stimulus, a total number of data in excess of 52 *million* will required for the training set. Coupled to the size of the set is training time. Not only will the analysis of the training images to provide the necessary output responses be time consuming, the amount of time required to perform the actual training of the network will be exceedingly high. At the opposite end of the scale to this is the use of a very small network to analyse small regions of the image. If, in order to reduce training set sizes and, consequently, training time a very

small "window" of pixels is used, the risk arises of returning to the original problem associated with phase unwrapping. If only a few pixels are analysed at any one time, the danger occurs of data being so localised that the network behaves in the same way as a highly computationally complex version of a simple point-to-point algorithm, with local errors being propagated through the final result. It was thought that an optimum solution to the phase unwrapping problem may lie between these two extreme cases. Investigation was carried out into neural network based regional phase map analysis.

4.5.2 Unwrapping by regions

For the purpose of this work, the images used were ITEX and Targa format, that is, images consisting of 262,144 pixels, arranged in a square of 512 x 512. This thesis concentrates on the unwrapping of phase images of this size and format.

As previously described, if a single "tile" of 512 x 512 phase values is used, the result will be one of extreme computational complexity. However, if a region-based unwrapper is configured where the region of interest is very small, then the original problem is encountered; a highly complex point-to-point unwrapper is achieved.

The smallest viable tile

If a tile of 4 pixels is used, that is, a 2 x 2 matrix, the point-to-point problem will undoubtedly occur. Simply comparing two adjacent values, even in two directions is no advance on the original point-to-point algorithm. The smallest tile that would be a viable basis for a neural network based unwrapper is, therefore a 9-pixel region, i.e., a 3 x 3 pixel matrix.

Network configuration

Following the previous successes of the backpropagation paradigm for phase unwrapping, the same network type was used. The network was configured to analyse a 3x3 pixel matrix, so consisted of 9 input neurons. Following previous work, the network would be designed to produce a "high" output at a wrap position, so the number of output neurons was the same as the number of input neurons. For initial experimentation, the same number of hidden neurons was again used. Learning rules and transfer functions were used following the successes of previous experiments.

The first network was, therefore, configured as follows:

- 9 input neurons
- 9 hidden neurons
- 9 output neurons
- Sigmoid transfer function
- Normalised-cumulative-delta learning rule

Figure 4.17 shows a schematic diagram of the network used for this experiment.

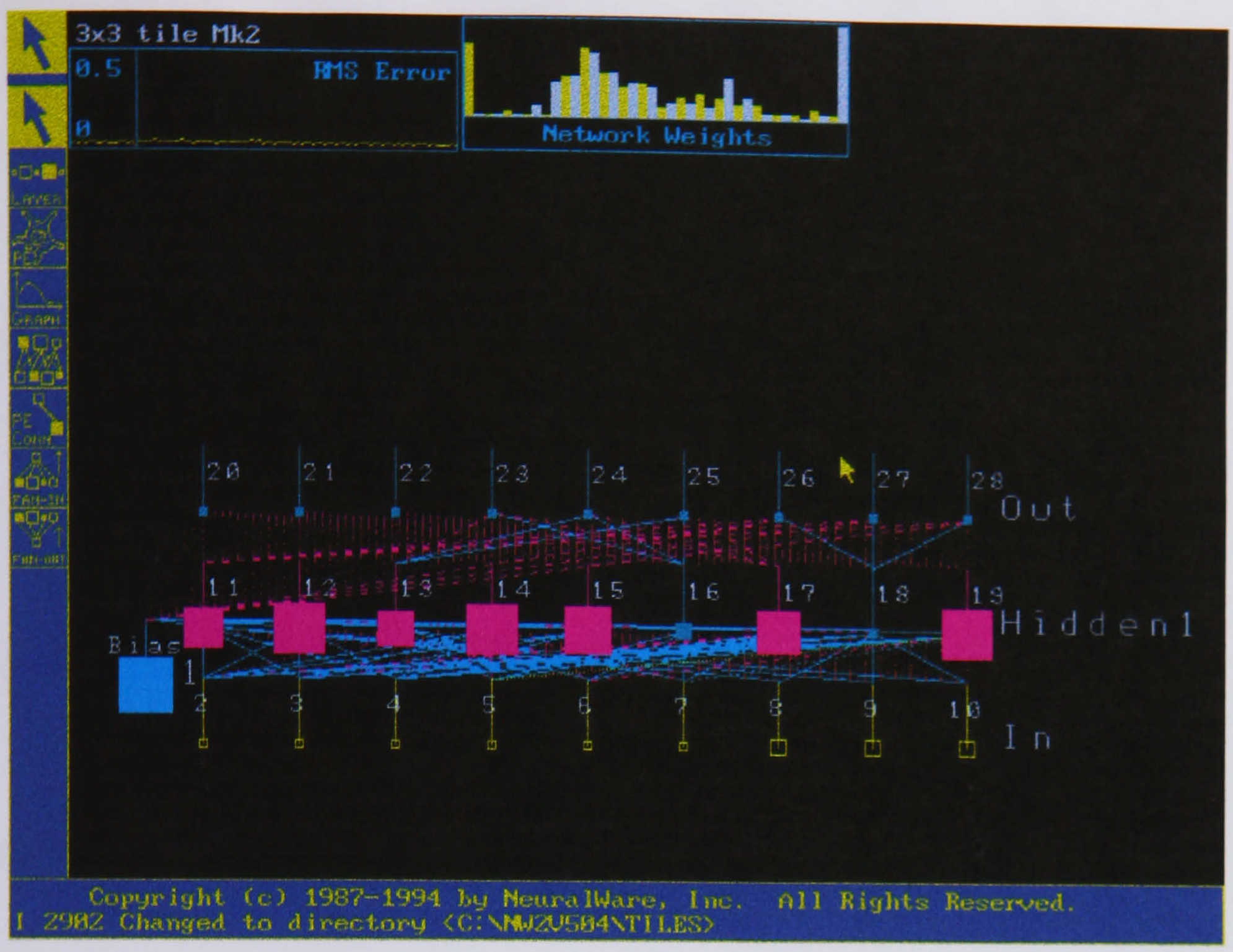


Figure 4.17: The 9 input tile unwrapping network

Training the network

Earlier experiments have also shown that networks achieve better results when trained with "real" data. Training data was taken from 3x3 pixel regions of real phase maps. The first phase maps had a high signal to noise ratio, as shown in figure 4.18. The phase map shown in this diagram is the result of performing an FFT analysis on straight fringes projected onto a flat, matt white surface. The data were chosen from random points within the image. Each set of data was analysed manually to ascertain where a wrap occurred and output values added to each set of data accordingly. Where no wrap occurred, the desired output value was left as zero, with a 1 being inserted where a phase wrap was present. Each training vector was thus formed. An example of selected training vectors is shown in appendix 1.

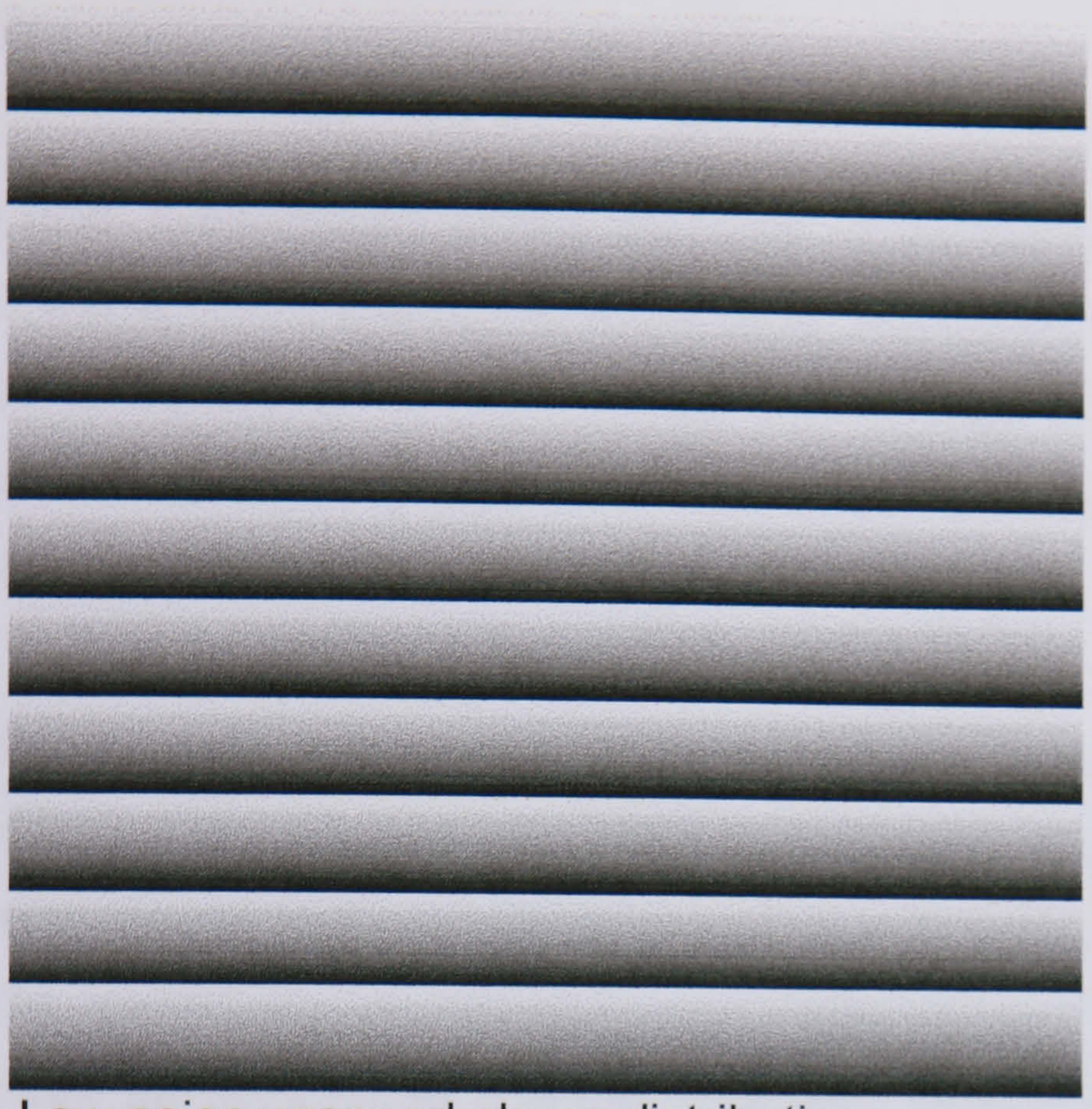


Figure 4.18: Low noise wrapped phase distribution

With the training data complete, it was possible to train the first network. The initial limit was set at 10,000 presentations and the network set to train. Training continued for the whole session, with the RMS error failing to reach the specified convergence criterion. The values did converge after approximately 8,000 presentations, but to a value slightly above the convergence criterion (CC). The distribution of weights throughout the network was approximately Gaussian, according to the weight histogram.

When the network had been fully trained, it was tested. The test data were taken from the same images as used for the training data, but from different areas of those images. This was to avoid using the same values for both training and testing the network. Again, if the network is tested using the training data, this simply proves that it has learned the patterns presented to it in that data and is not representative of how the network will perform.

4.5.3 Results

The results from presentation of the test data to the network are shown in appendix 1. For a clean, low-noise, real image, the network appeared to be producing an accuracy of 100%. Where the desired output expected a 1.0000, the network gave 0.959 and where a low output (0.000) was required, the highest value given by the network was 0.018.

While the above results appeared to show the network to be 100% accurate in its recognition of the position of various phase wraps, it must be borne in mind that these results had been achieved using only low-noise, high quality images. As previously described, most "real world" wrapped phase distributions are far from clean and will invariably contain significant noise. To test the network further, a second test set was created. This used the original data extraction program to take phase data from a noisier phase map. The phase map used for this experiment is shown in figure 4.19. Using the equation described in section 4.4, the signal to noise ratio of the image was calculated as 36.124dB. Here, the noise value was calculated using the Fourier transform of the original image. The central peak of the FFT was isolated and classified as signal, the rest of the data being taken as noise. This method of calculation can be somewhat arbitrary, as the size of the filter used to isolate the peak in the FFT is not always clearly defined. Several regions of spot noise can be seen in this image. Conventional point-to-point unwrapping algorithms can easily be confused by this kind of noise and, although achieving high accuracies on the initial test data, the tile unwrapper had not been trained to deal with noise of this kind. A test file of 25 vectors was constructed as shown in appendix 1.

Figure 4.19: A noisy wrapped phase distribution

When presented with the new test data, the network gave the results shown in appendix 1. Again, all desired high outputs were greater than 0.95 and low outputs less than 0.02. The network appeared to be "ignoring" the spot noise present in the test file and giving an accurate diagnosis of phase wrap position.

As the results given by the initial wrap detector appeared good, the network was Flash Coded to produce a fully functional unwrapper. As with previous work, the Flash Code function of the NeuralWorks package provided network simulation code to perform the wrap detection. Further code was required to turn this into a useable phase unwrapping system.

The initial unwrapping code was designed to test the viability of the Flash Coded network by unwrapping a single tile. The program read a series of phase values from an image, presented them to the network and wrote the network outputs to an array. The positions of the values in the array correspond to the positions of the phase wraps in the region of interest. The phase values were then updated by $\pm 2\pi$ at the points specified by the neural network.

The results of the initial experiment show that it is theoretically possible to use a "tile" system such as this as the basis for phase unwrapping. The unwrapping of an entire 512x512 pixel wrapped phase distribution is, however, much more complex. The results from a number of tiles must be combined to give a complete unwrapped phase distribution. The next problem to be addressed was how to combine the wrap detector tile with a complete wrapped phase distribution.

The size of the tile for the initial experimentation has been defined above as a 3x3 pixel square. This size was arrived at for reasons of computational complexity. If a large tile was used (the maximum tile size being equal to that of the image - 512x512 or 262,144 pixels), the network needed would be extremely complex. If, however, the smallest possible tile of 2x2 pixels was used, the result would be an extremely complex point-to-point unwrapping algorithm. In the latter case, the same problems as encountered with Schafer's algorithm would arise and the entire object of the research would have been somewhat defeated. The use of a tile unwrapper still poses a number of problems.

The connectivity problem

Consider a portion of an image as shown in figure 4.20. If a 3x3 pixel tile is positioned with tile pixel a overlapping image pixel x, it will correctly unwrap its allotted section. If the window is then repositioned at image pixel x+3, again, a satisfactory unwrap of the new tile is likely to occur. Although two successful tile unwrapping operations may well have been carried out, it is also likely that the problem of a "missed wrap" may occur. If a phase wrap is present vertically in the image between pixels x+2 and x+3, no calculation will have been made to detect its presence. The most sensible method of allowing for this would appear to be to convolve the tile and the image in a similar manner to the approach taken with the six pixel line unwrapper in the initial experimentation.

x,y	x+1,y	x+2,y	x+3,y
x,y+1	x+1,y+1	x+2,y+1	x+3,y+1
x,y+2	x+1,y+2	x+2,y+2	x+3,y+2
x,y+3	x+1,y+3	x+2,y+3	x+3,y+3

Figure 4.20: Tile convolution.

Tile convolution

Figure 4.21 shows how the input tile and the input layer of the network are related. Each tile element has a corresponding single input neuron

The tile was convolved with a 512x512 wrapped phase distribution, which yielded an intermediate image showing the position of the wraps. The first image used to test the system is shown in figure 4.22 and the output from the wrap detector in figure 4.23. Calculation of signal to noise ratio using the equation described in section 4.4 yielded a value of 38.622dB Conventional code was used to perform the 2π phase shifts required and the result of the unwrapping procedure is shown in figure 4.24. In order to produce the final code for the unwrapping system, the "C" code generated by the NeuralWorks package was translated into IDL (Interactive Data Language).

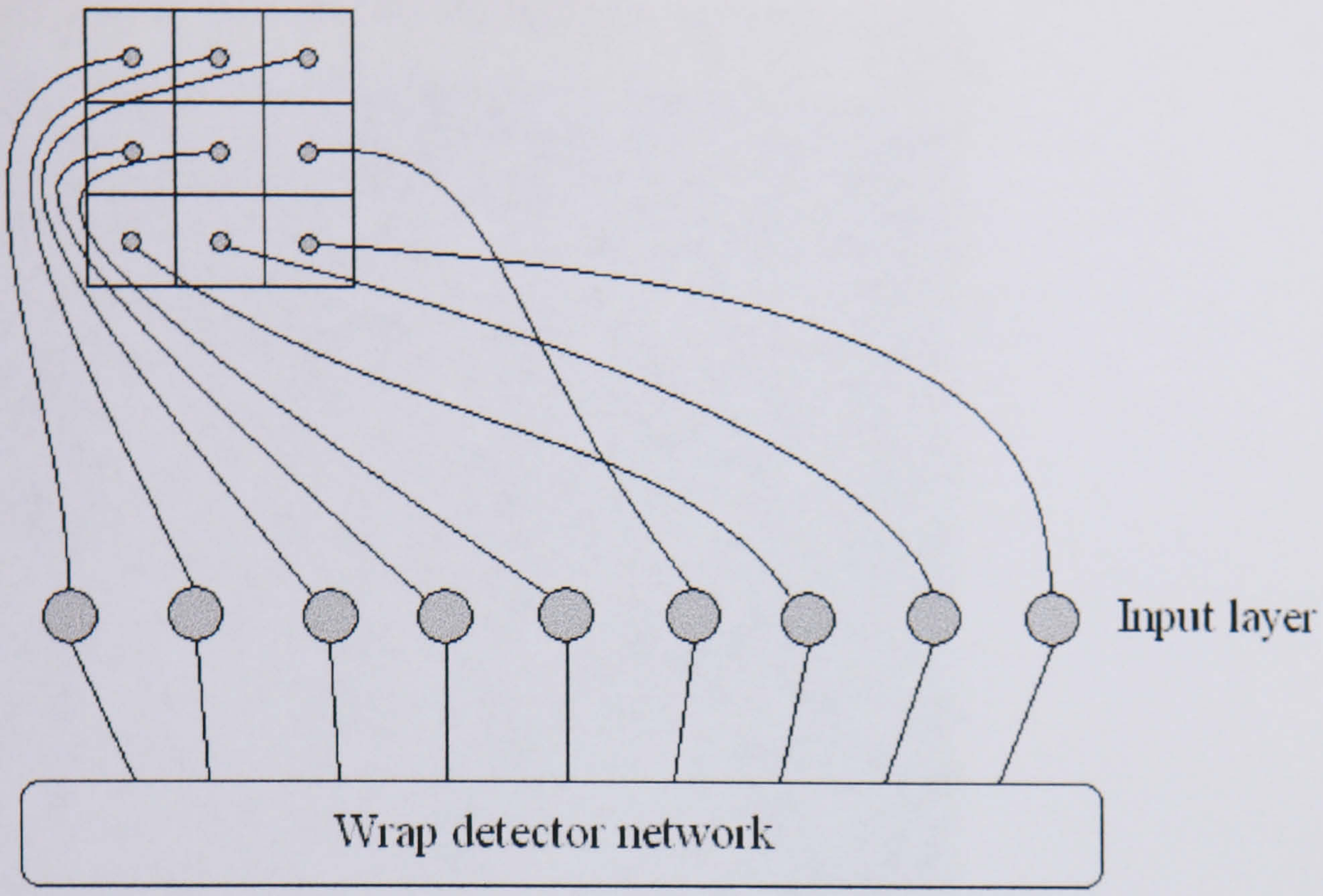


Figure 4.21: Tile network input configuration.

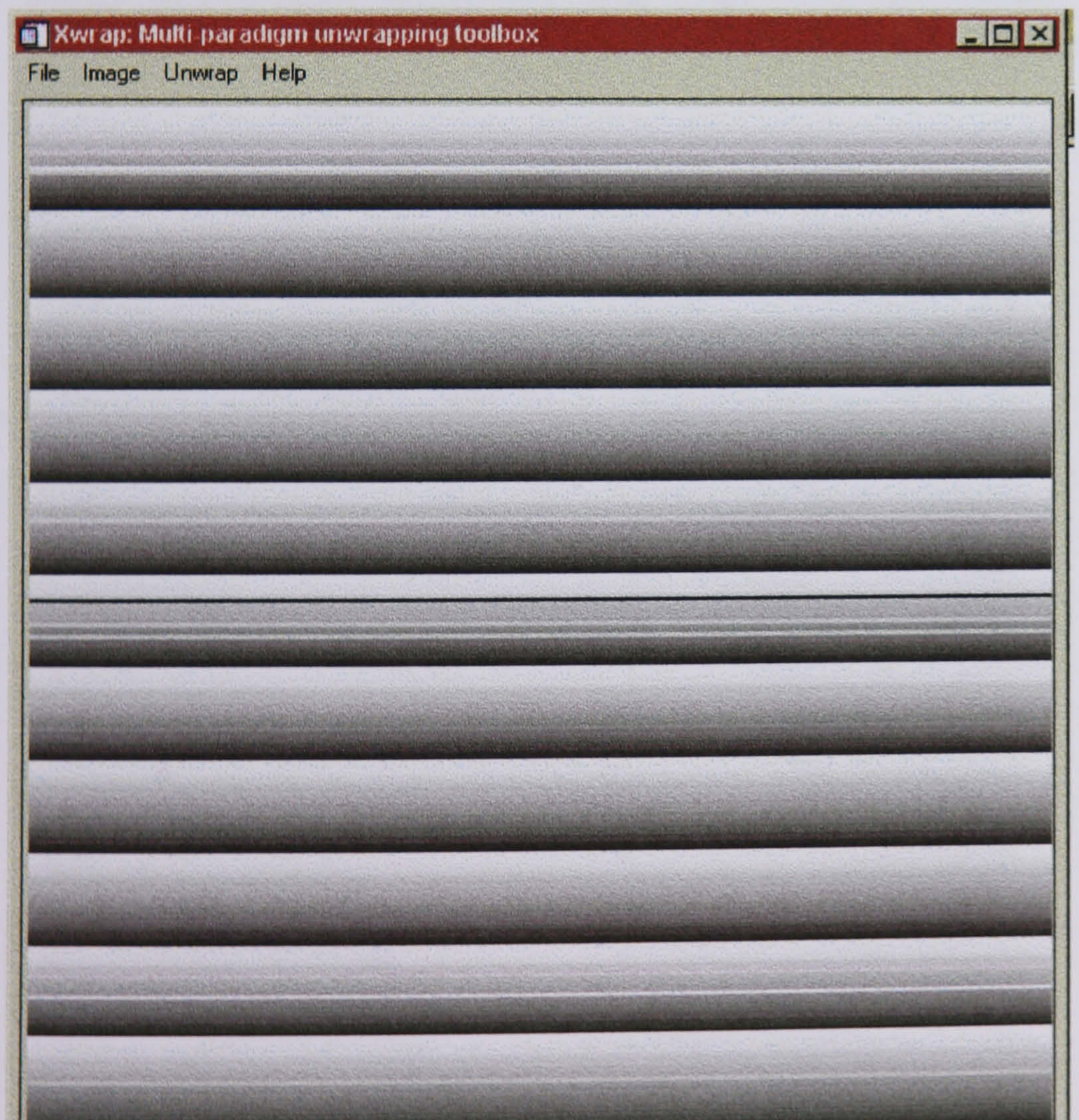


Figure 4.22: 2-D wrapped phase distribution with added noise

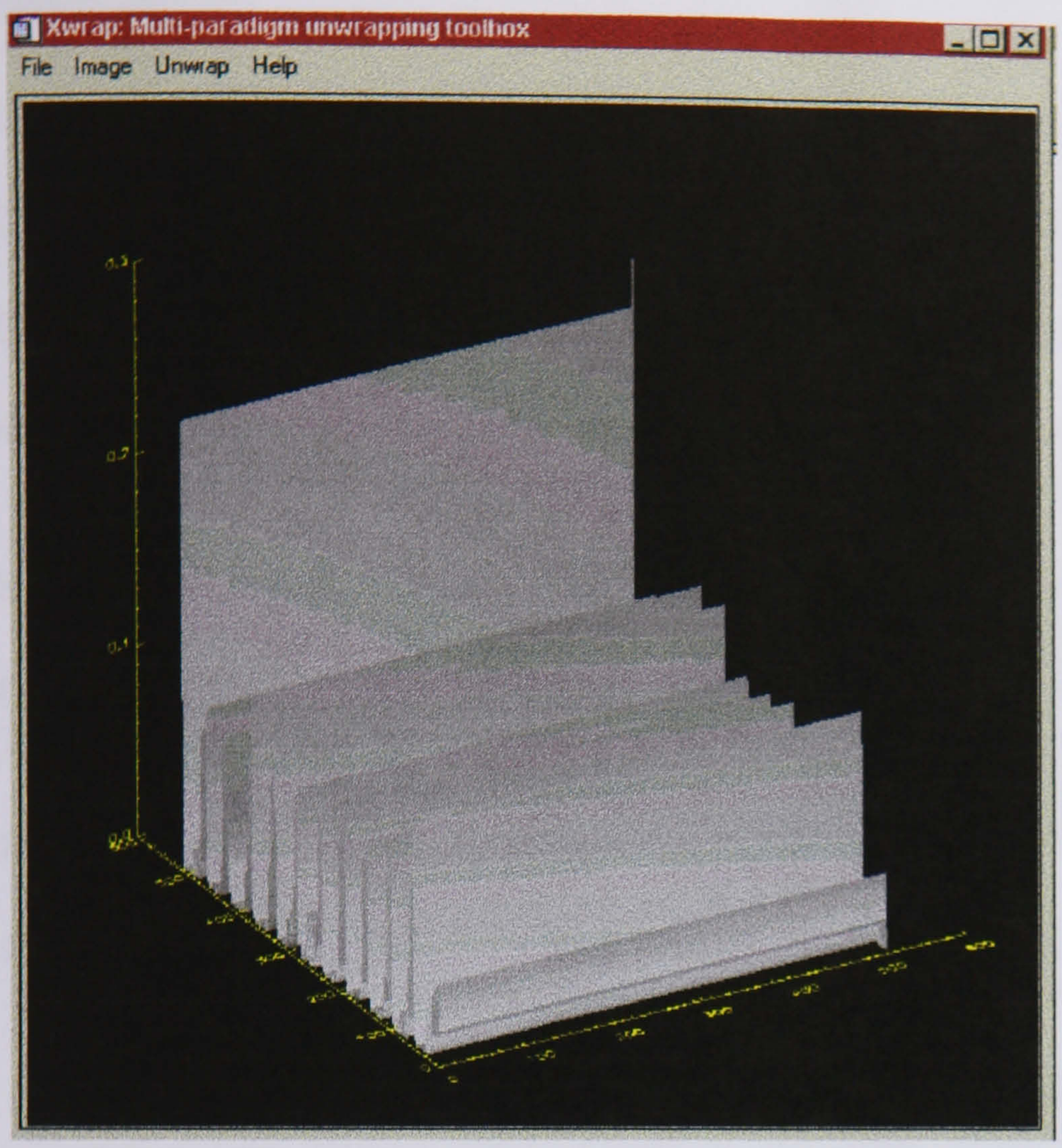


Figure 4.23: Output of tile wrap detector.

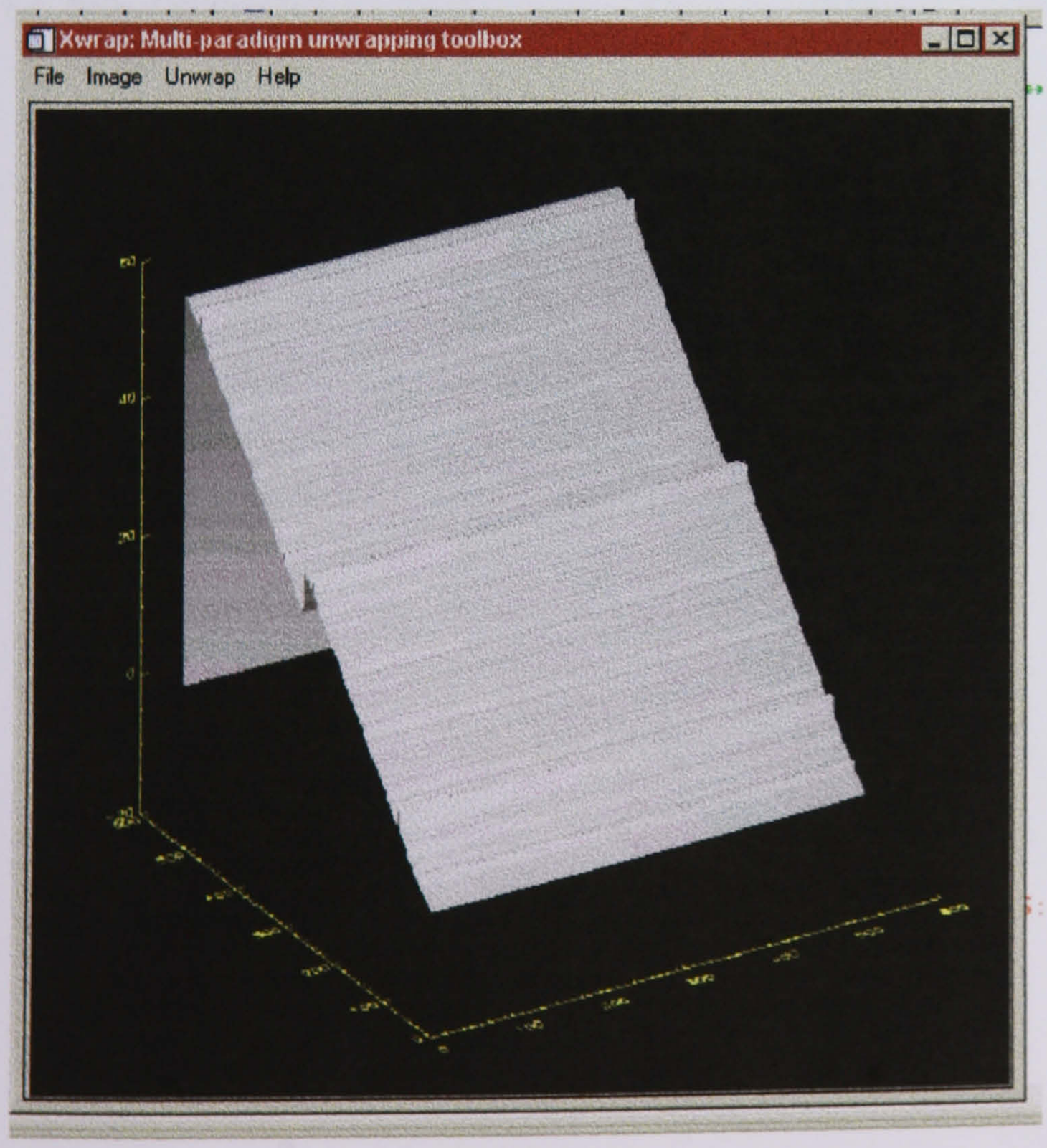


Figure 4.24: Unwrapped phase distribution.

References

1. Schafer and Oppenheim. "Digital Signal Processing", Prentice-Hall, 1975.
2. W.H.Carter. On unwrapping two-dimensional phase data in contour maps. Optics Communications 94(1992)1-7
3. J.M.Huntley. Noise-immune phase unwrapping algorithm. Applied Optics, 25,10(1986)1653-1660
4. J.M.Huntley & H.Saldner. Temporal phase unwrapping algorithm for automated interferogram analysis. Applied Optics, 32,17(1993)3047-3052
5. D.C.Ghiglia, G.A.Mastin & L.A.Romero. Cellular automata method for phase unwrapping. J. Opt. Soc. Am. 4,1(1987)267-280
6. J.J.Gierloff. Phase unwrapping by regions. Proc. SPIE 818(1987)2-9
7. D.J.Bone. Fourier fringe analysis: the two-dimensional phase unwrapping problem. Applied Optics 30,25(1991)3627-3632
8. G.M.Brown. Practical phase unwrapping. Proc. SPIE 1553(1991)204-212
9. P.Stephenson, D.R.Burton & M.J.Lalor. Data validation techniques in a tiled phase unwrapping algorithm. Optical Engineering 33(1994)11, 3703-3708.
10. G.D.Kendall & T.J.Hall. Performing fundamental image processing operations using quantised neural networks. Proc. FRINGE '93.
11. M.Takeda. Phase unwrapping by neural network. Proc. FRINGE '93.
12. NeuralWorks Professional II Plus Reference Guide. NeuralWare Inc. 1993

Chapter 5: Fringe optimisation

5 Fringe Optimisation

5.1 Introduction: What is fringe optimisation?

While the problem of classical optimisation is well documented, the need for the optimisation of projected fringe patterns has not been the subject of extensive research. Previous chapters have discussed the need for and implications of fringe analysis as a measurement tool. During a measurement operation a fringe pattern is projected onto an object's surface, the returned image is processed and a result is given. The result can generally be classified as "good" or "bad" - The analysis gives an acceptable result, or the analysis fails to work or it works badly. This leads to three possible questions:

- For any given surface, is there a fringe pattern that produces a good result?
- How can the quality of that result be defined?
- How can such a fringe pattern be found?

Until recently, this was a problem that was difficult to address as it was not possible to have accurate control over the characteristics of a fringe pattern on-line. With the development of the twin-fibre adaptive interferometer[1], however, dynamic fringe pattern optimisation is now feasible. Consider a conventional fringe analysis system as shown in figure 5.1. The system has three major components: a projection device, an image capture device and a digital computer. The projection device usually consists of an interferometer or grating to produce a fringe pattern that is projected onto the object's surface. The fringe pattern is viewed through a CCD camera, which is connected to a frame store, and all analysis is carried out by a digital computer. This type of system is adequate for analysis of fringe patterns

and has, for several years, been the accepted method. Its main drawback, however, is the inability of the user to have control over the pattern which is projected. All gratings provide a fixed fringe pattern and the adjustment of most interferometers is an extremely labour intensive task. If the system is required to analyse a number of objects whose shapes vary considerably, a fixed fringe pattern may not be suitable. It was with this in mind that the twin-fibre adaptive interferometer was developed[1]. Consider the example as shown in figure 5.2. Here a surface is shown onto which fringe patterns have been projected. Figure 5.2(a) shows a fringe pattern which is likely to produce a good final result. However, the fringe pattern shown in figure 5.2(b) is less satisfactory. The fringe spacing is very small, the fringe contrast is very low and the overall intensity of the image is very low. These criteria will mean that a successful measurement is less likely.

Figure 5.3 shows a schematic diagram of the interferometer. The light is supplied by a 15mW helium-neon gas laser, which is launched into an optical fibre. The optical fibre is split to provide two coherent light sources which, if closely spaced, behave in a similar manner to Young's experiment to provide an interference fringe pattern. The adaptive nature of the interferometer is reliant upon how the fibres are mounted. One fibre end is mounted in a fixed position inside the interferometer, while the other is mounted on a translating stage. Figure 5.4 shows the internal arrangement of the interferometer. If the position of the movable fibre is varied, it becomes possible to change both the spacing and the orientation of the fringes. The position of the translating stage is governed by a pair of MotionMaster positional controllers, manufactured by the Klinger Corporation. The controllers provide a means of accurately positioning the fibre in two dimensions. If the computer that performs the analysis of the fringe pattern controls the position of the fibres, it then becomes possible to achieve a "closed loop" system, where the fringe pattern is adjusted to suit the object to be measured before measurement actually takes place. Figure 5.5 shows a schematic diagram of the "closed loop" system. The fringe pattern is provided by the twin-fibre interferometer, with the fibres being

positioned by the computer. The fringe pattern is viewed by the CCD camera and the image relayed to the computer's frame store. Here it is possible to make a decision regarding the quality of that fringe pattern and, if it is not satisfactory, it can be adjusted before any analysis takes place. This leads on to the question of how it is possible to produce an optimum fringe pattern for a given surface.

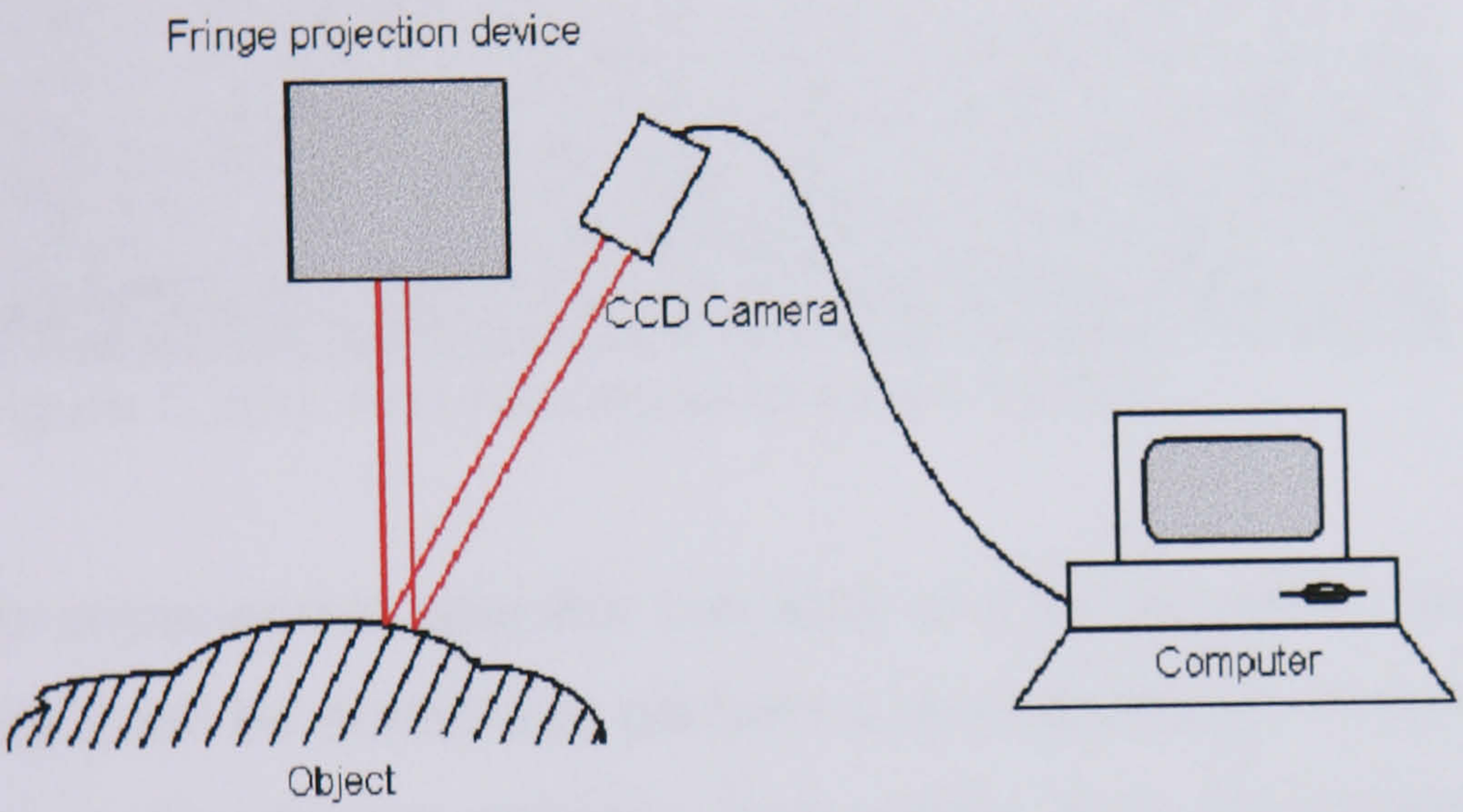


Figure 5.1: A conventional fringe analysis system

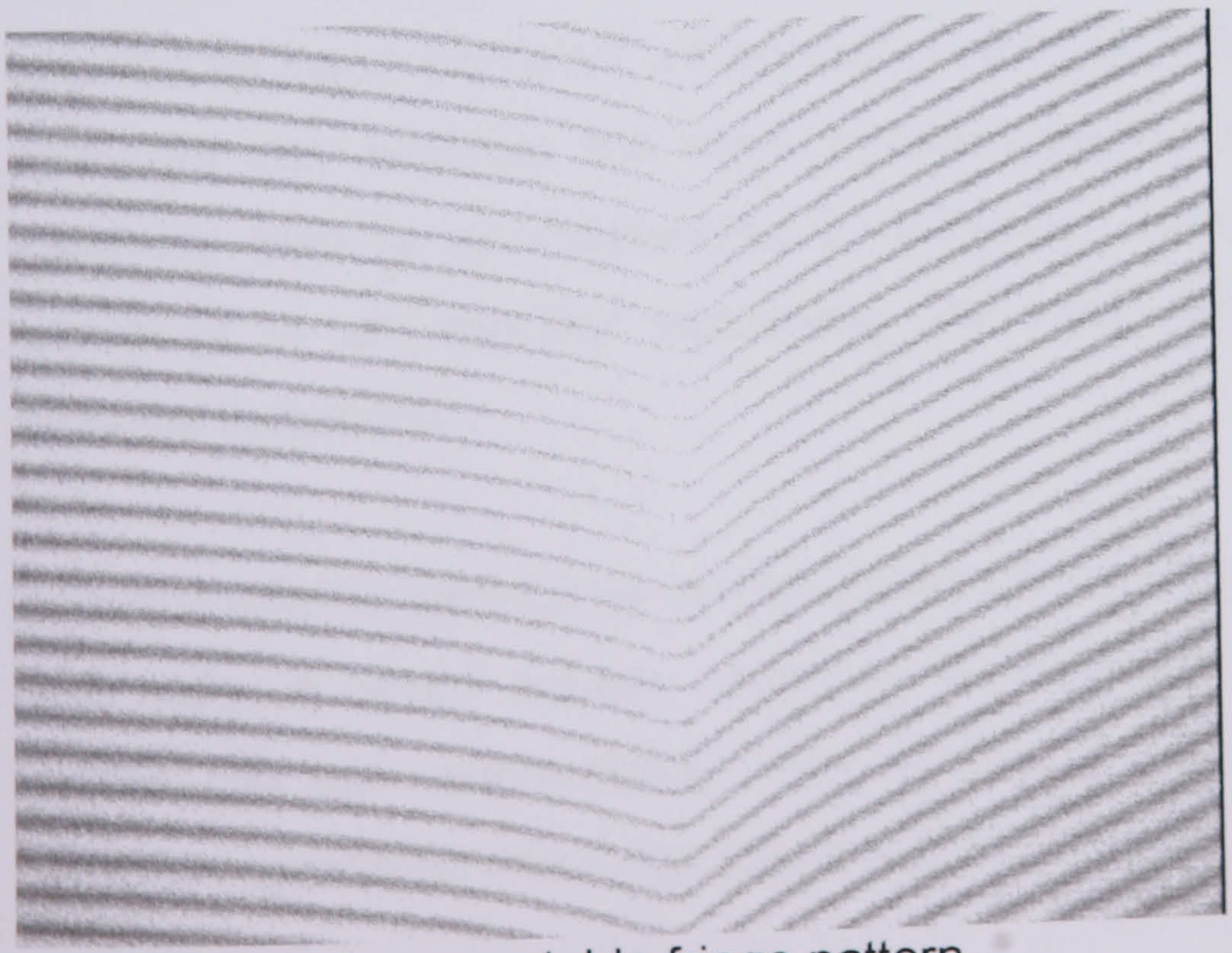


Figure 5.2(a): An acceptable fringe pattern

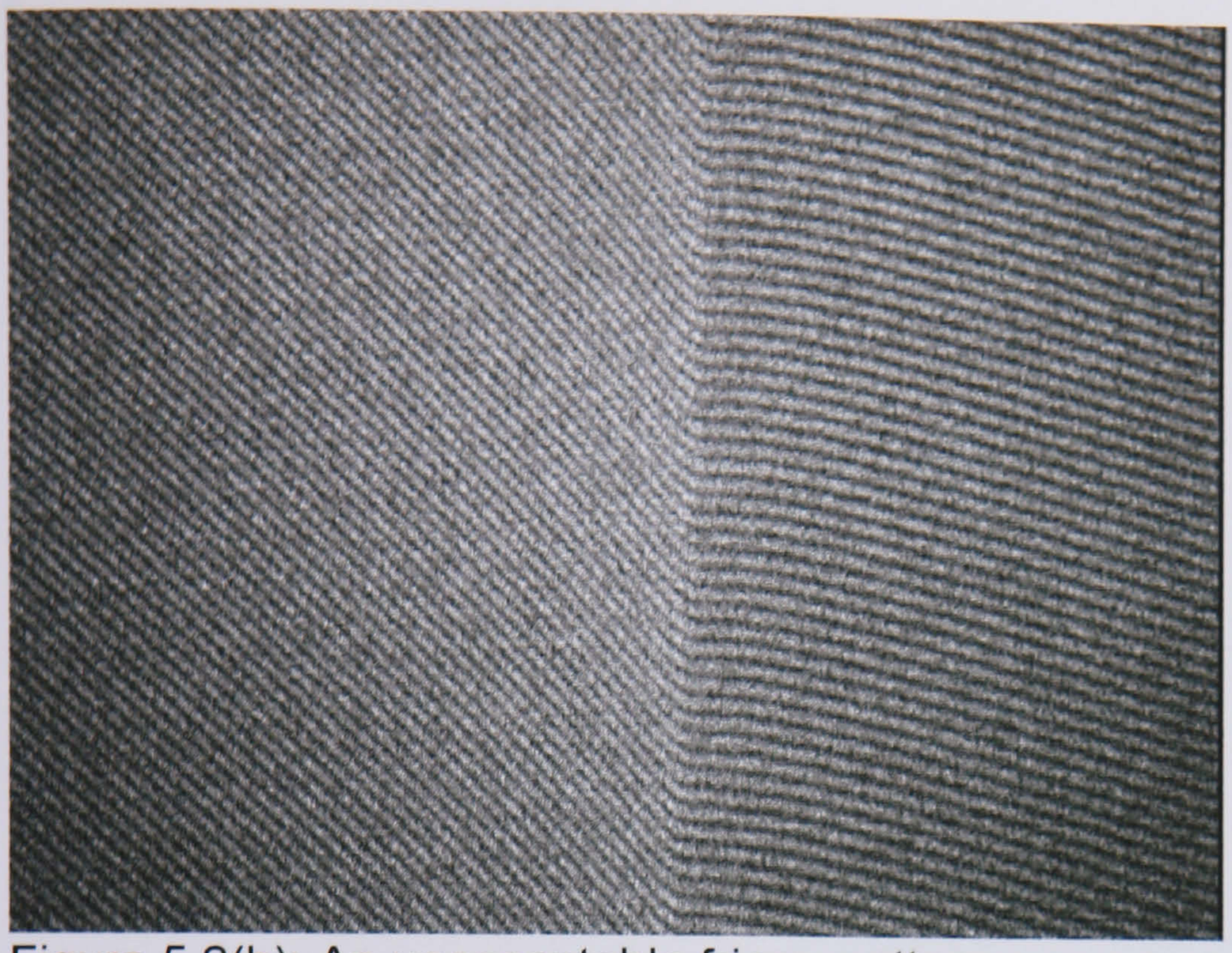


Figure 5.2(b): An unacceptable fringe pattern.

An experienced operator can look at a fringe pattern and decide whether or not it will be suitable to perform a measurement. A cursory analysis of how many fringes are present, how widely they are spaced or how bright the image is will give an indication of its suitability. A decision such as this is most often based on intuition and experience of the operator, not detailed mathematical analysis. As with many processes which rely on a human operator for input, the result will not necessarily be consistent. Two different operators may perceive the same situation slightly differently, which will inevitably lead to inconsistencies in analysis. Also, many factors may contribute to an operator perceiving the same situation differently on separate occasions.

Now that reliable technology exists to adjust the attributes of a fringe pattern accurately, it would appear desirable to construct a system that could perform this task automatically. It is on this premise that experimentation has been conducted into fringe optimisation.

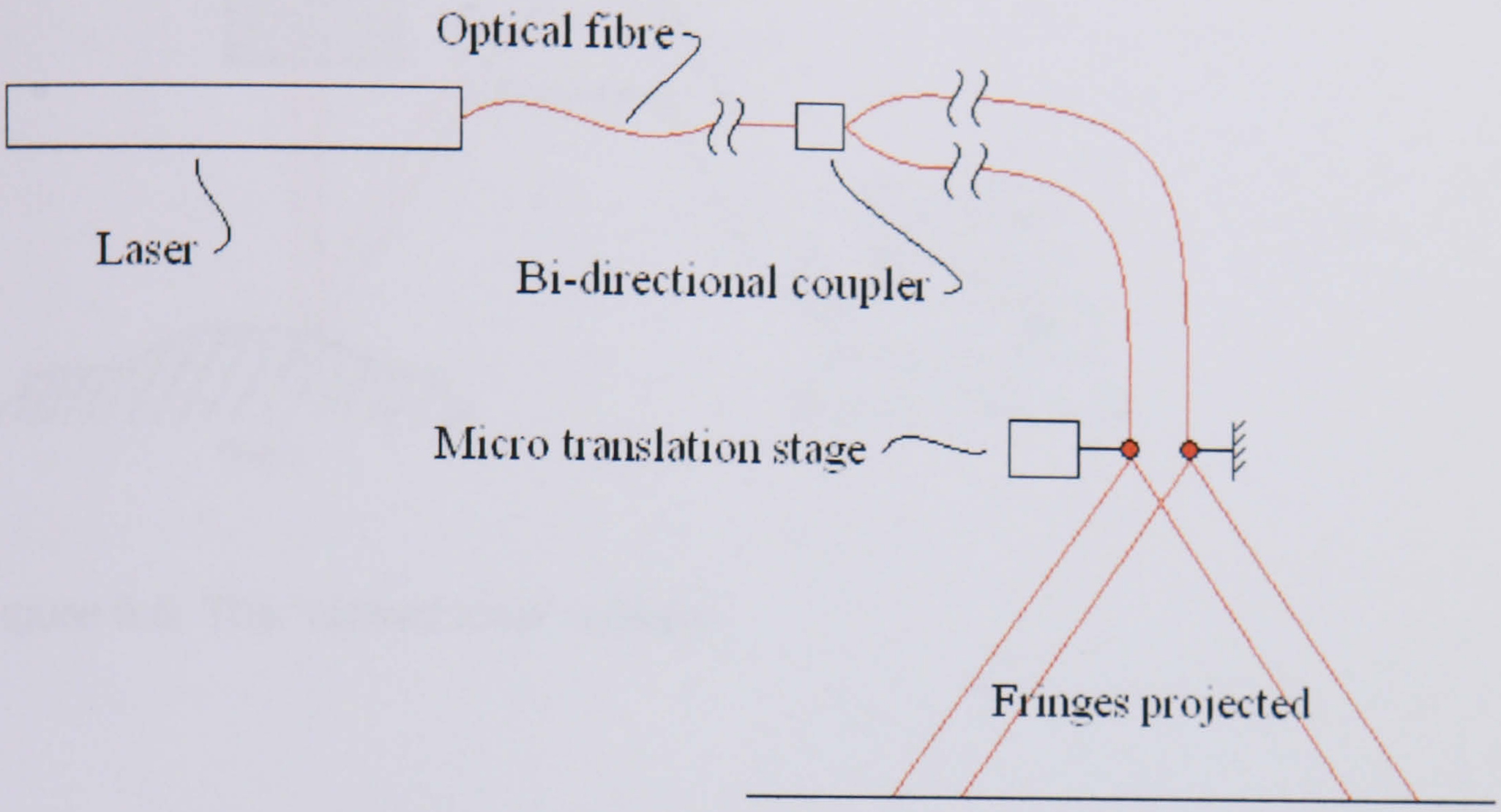


Figure 5.3: Schematic diagram of the adaptive interferometer

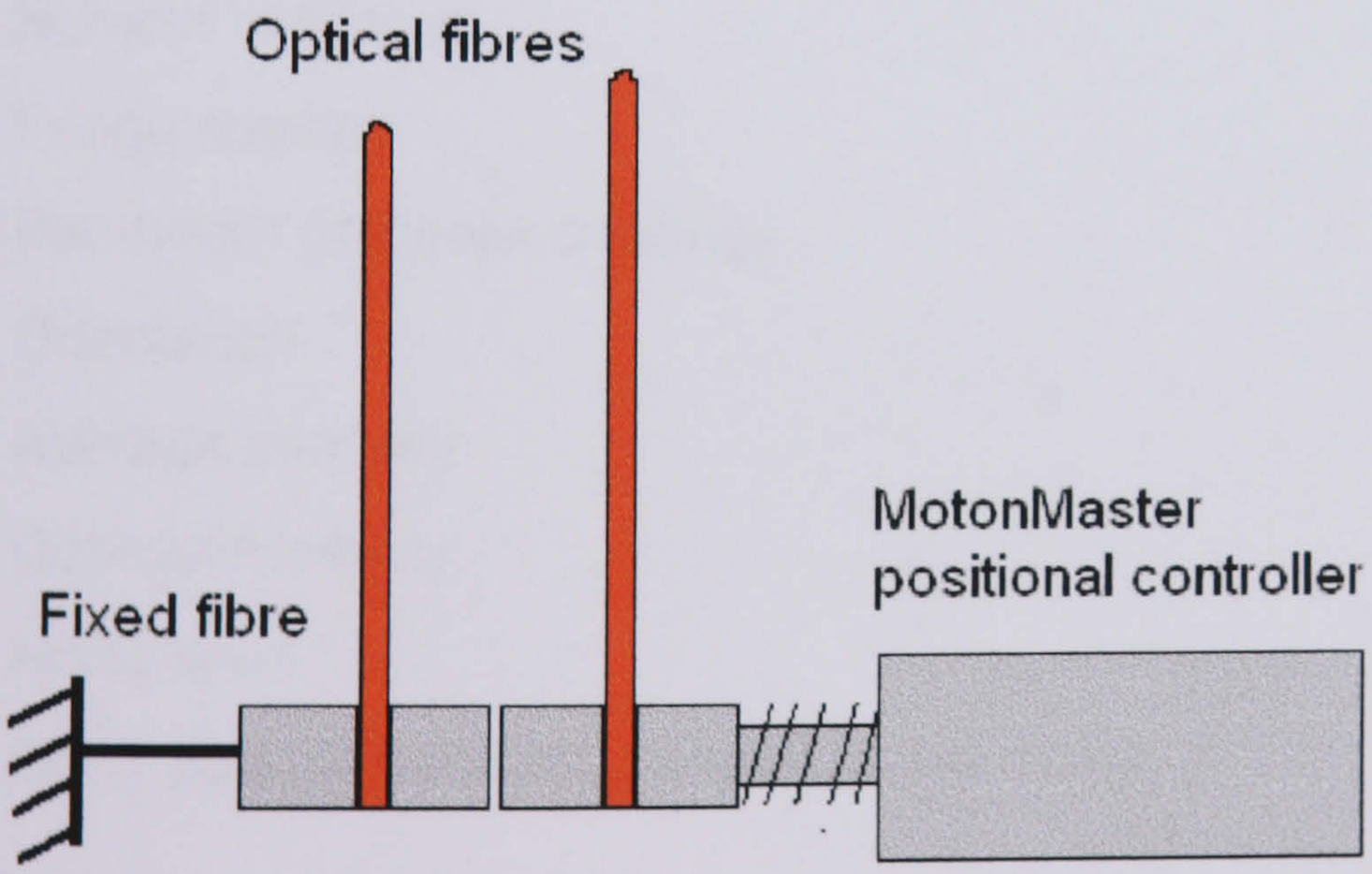


Figure 5.4: The adaptive interferometer internal arrangement.

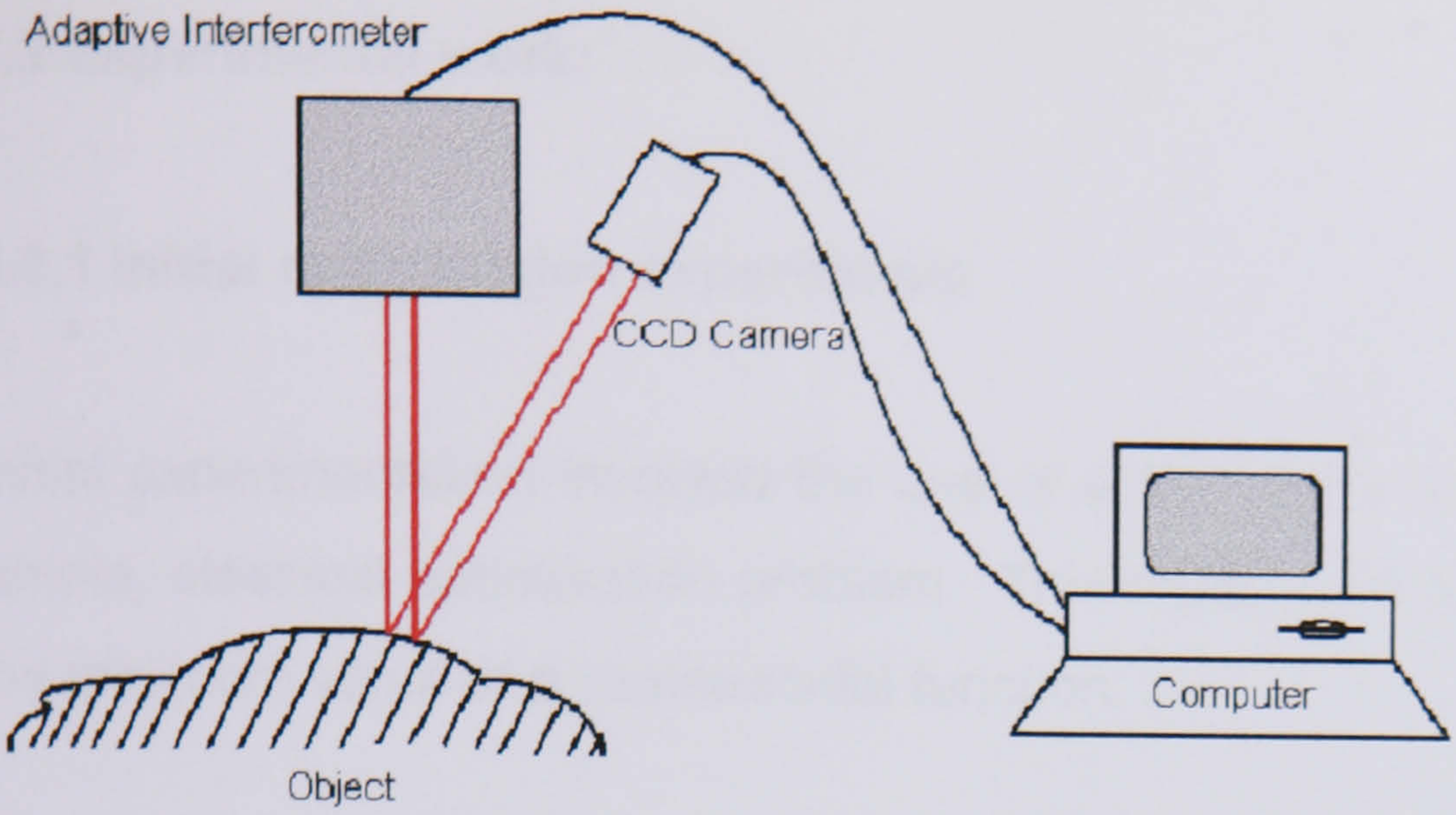


Figure 5.5: The "closed loop" system

How can a fringe pattern be defined?

A number of parameters can be used to define the quality a fringe pattern in some sense. These are:

- Number of fringes
- Fringe spacing
- Bandwidth (min/max spacing)
- Orientation
- Average intensity
- Contrast/visibility
- Noise level
- Other periodic features present in the image.

Again, this can be viewed as a problem of recognition, which leads to the question: can a neural network be trained to recognise a good or a bad fringe pattern?

5.2 Experimental work:

5.2.1 Initial optimisation experiments

Initial experimentation involved the use of small neural networks to solve a simple, classical optimisation problem. This experiment attempted to locate the minimum value of a cosinusoidal function.

Experiments with unsupervised learning

For this experiment, a Learning Vector Quantisation (LVQ) network was used. A network was configured with 10 input, 10 hidden and 10 output neurons. Figure 5.6 shows a schematic diagram of the network.

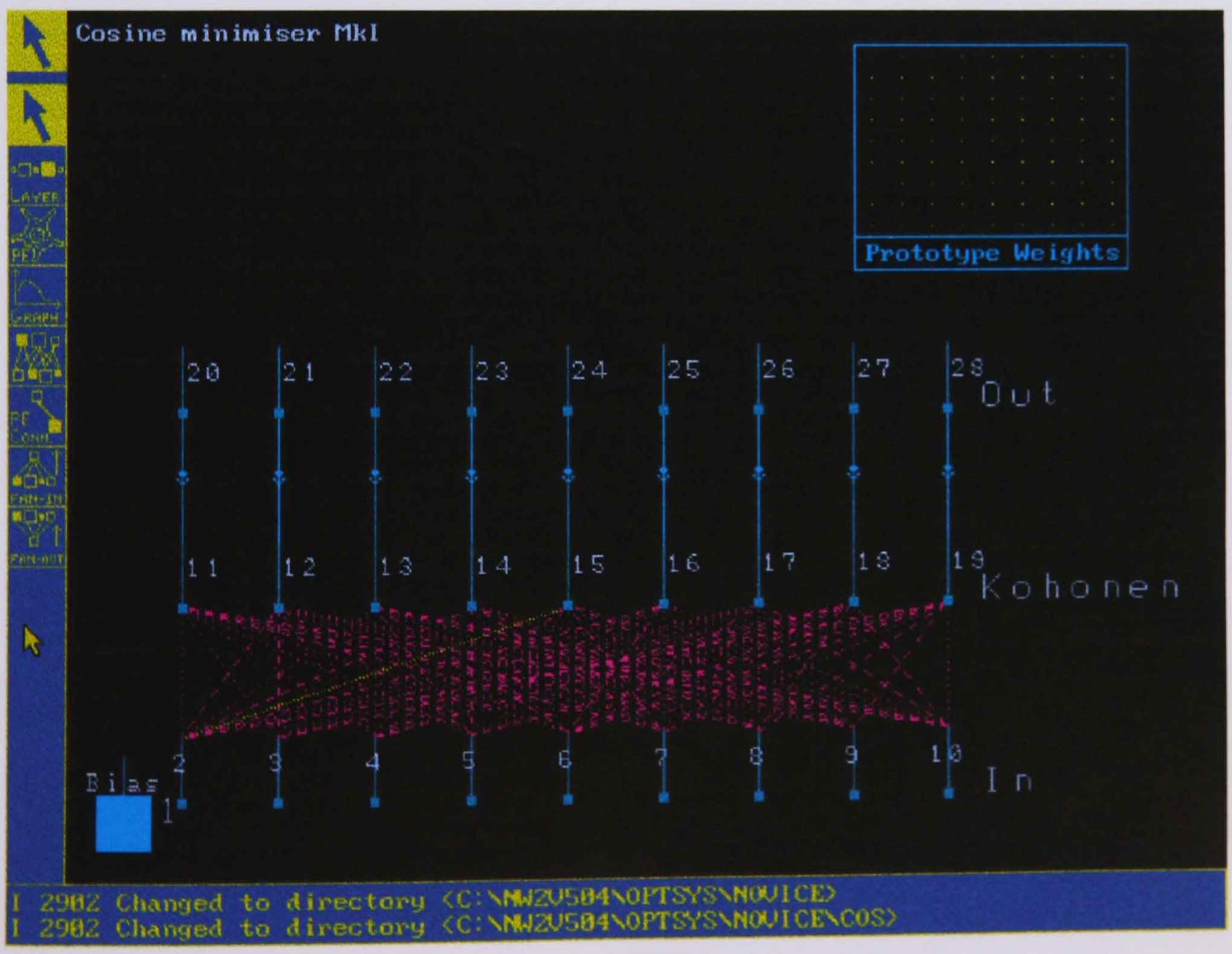


Figure 5.6: Neural network for detecting minima of a cosine function.

A training set was constructed using simulated data containing no noise or false minima. The completed training set contained 38 training vectors. When LVQ networks are used, the network specifies its own number of presentations of data, which is related to the number of vectors present in the training set. The network learned for 1710 presentations and when training ceased, the network was tested using a set of test data of the same size as the training set. The results obtained from the test set are given in. The minima were being correctly identified on approximately 60% of occasions.

Most of the incorrect identifications occurred when "false" minima were present in the data. Consider the graphs shown in figure 5.7. Figure 5.7(a) shows a cosine function and the network's response. The minimum of the function is correctly identified. However, if a function is introduced as in figure 5.7(b), the network shows a tendency to become confused, as the response confirms. The false minimum in the function tends to draw the output away from the actual minimum value.

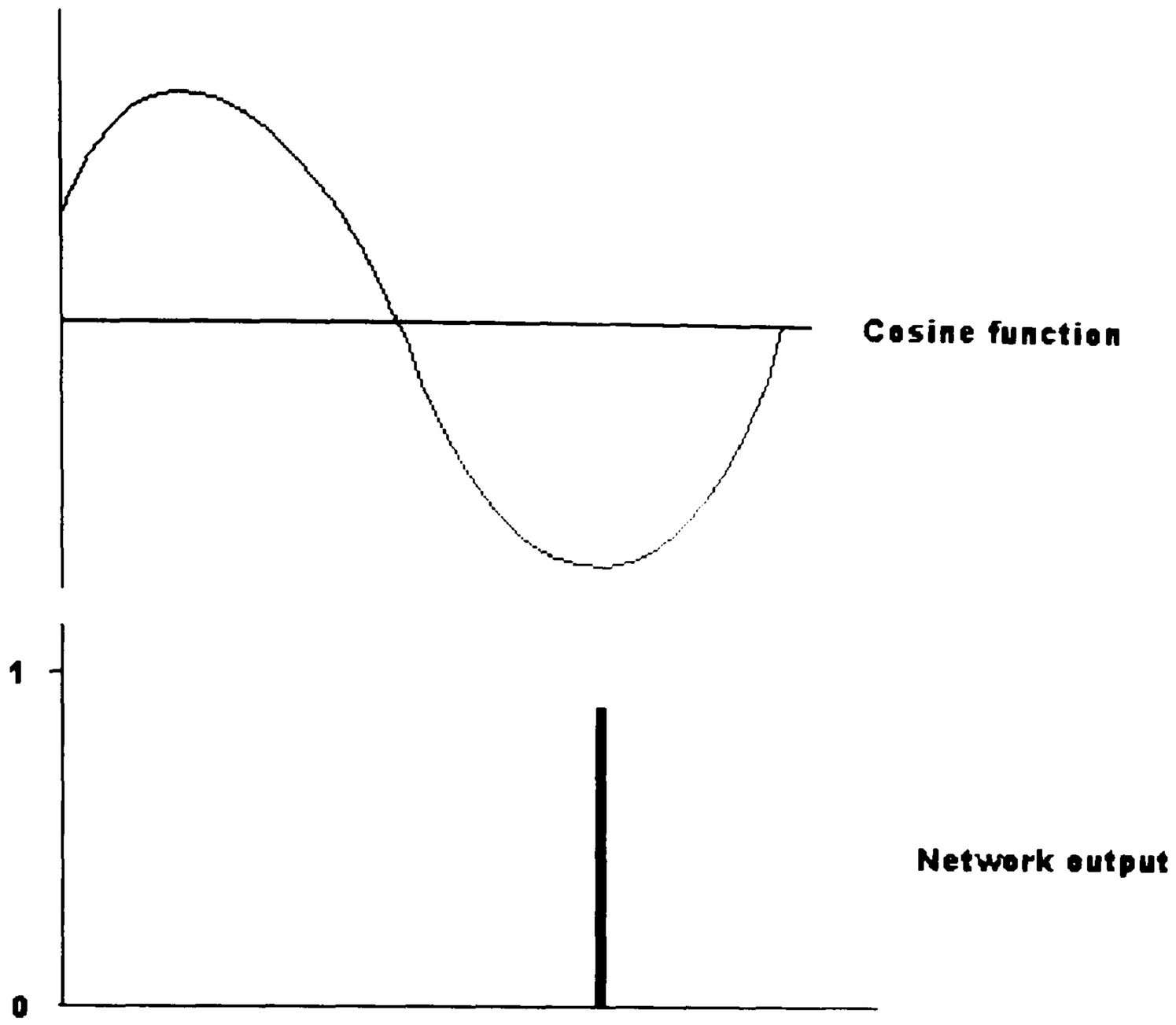


Figure 5.7(a): The network's response to a cosine function

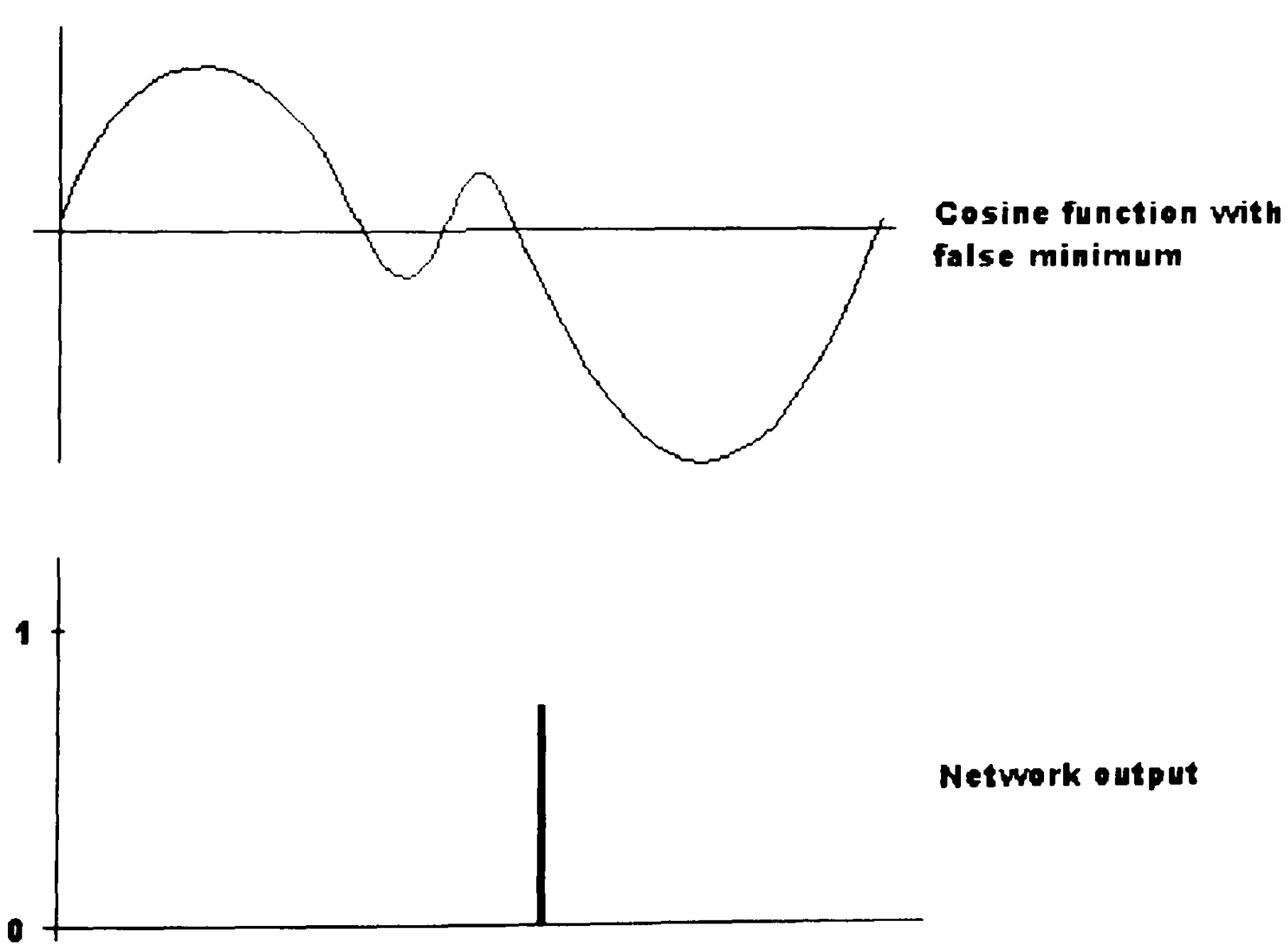


Figure 5.7(b): False minima in the signal

Experiments with supervised learning

A backpropagation network was configured, having 10 input, 10 hidden and 10 output neurons and the training data presented to it. Figure 5.8 shows the network, which was trained using the same data set as previously described. Training for 10,000 presentations caused the RMS error to drop gently and smoothly but not reach its convergence criterion. The weight distribution approximated a Gaussian distribution and the results file showed a slightly better response than for the LVQ network. The high output values were not as high as the LVQ, but 75-80% of them were on target. Further training was initiated, and the network was left to train for 50,000 passes. After this extended training period, the RMS error reached a point slightly higher than the specified convergence criterion. The weight histogram approximated a Gaussian distribution, but showed a disproportionately large number of weights at the extreme ends of the graph. The results file showed that the high values were approximately 0.997, low values were approximately 0.004 and were on target 85% the time.

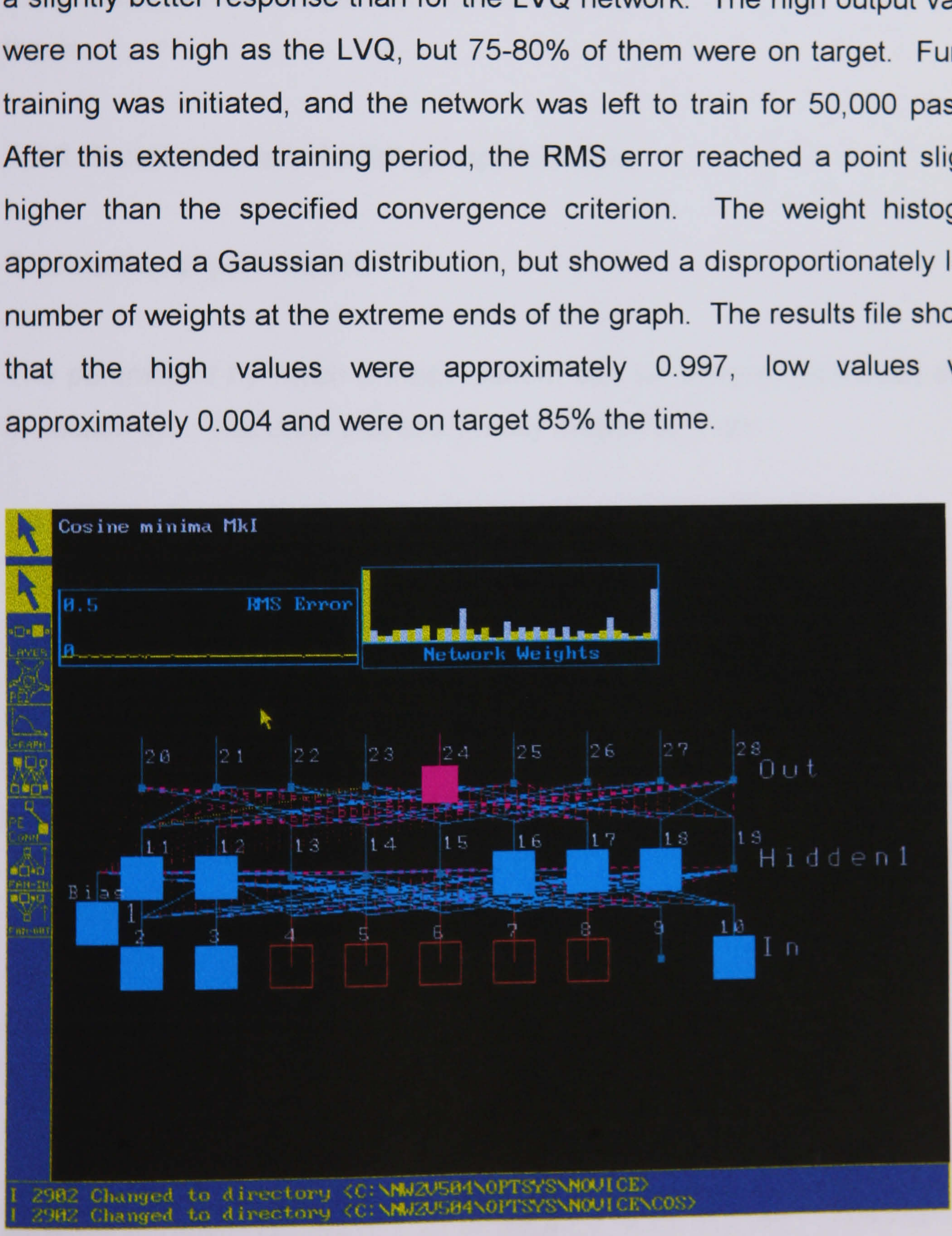


Figure 5.8: Cosine function minimum detector network.

Comparison of the two methods shows that the neural network approach to the optimisation problem yields some satisfactory results and that backpropagation networks are best suited to solving simple minimum location problems.

Considering the relative success of the neural networks is solving simple minimisation problems, it was decided to expand the scope of experimentation to include the problem specified in the original brief of this thesis.

5.2.2 Neural networks for fringe optimisation

Calculation of parameters

The parameters by which a fringe pattern can be defined have been stated in section 5.1. The ones that are directly measurable are:

- Mean intensity
- Contrast/visibility
- Number of fringes
- Fringe orientation

Mean intensity and fringe contrast/visibility are functions of the amount of light reflected from the surface that reaches the CCD array. Fringe number and orientation are the two parameters that are directly influenced by the position of the fibres in the adaptive interferometer. It is the latter two parameters which are most important in closing the loop, as they can be controlled directly by the computer which is performing the analysis. To fully automate the process, the camera would require a lens that can be operated automatically. To produce a completely "closed loop" system, a lens allowing automatic control of both focus and aperture would be desirable.

These parameters can be calculated thus:

Mean intensity:

For **n** pixels, the intensity value **I** at each pixel is recorded and the mean calculated by the following equation:

$$(I_1 + I_2 + I_3 + \ldots I_n) / n$$

Contrast and visibility

Contrast and visibility are calculated in similar ways, both involving a relationship between minimum and maximum intensity:

$$\text{Contrast} = (I_{max} - I_{min}) / I_{max}$$

$$\text{Visibility} = (I_{max} - I_{min}) / (I_{max} + I_{min})$$

To ascertain which of these values would be best to use for the analysis, both were calculated for a known, good fringe pattern. The fringe pattern used for the calculation was a pattern as shown in figure 5.9, having 64 fringes and a high signal-to-noise ratio

For a series of camera apertures, the maximum and minimum intensity values were recorded for the same fringe pattern, which is shown at its brightest (aperture=f2.8) in figure 5.9. From these values, the contrast and visibility were both calculated for each aperture. The results are shown in figure 5.10

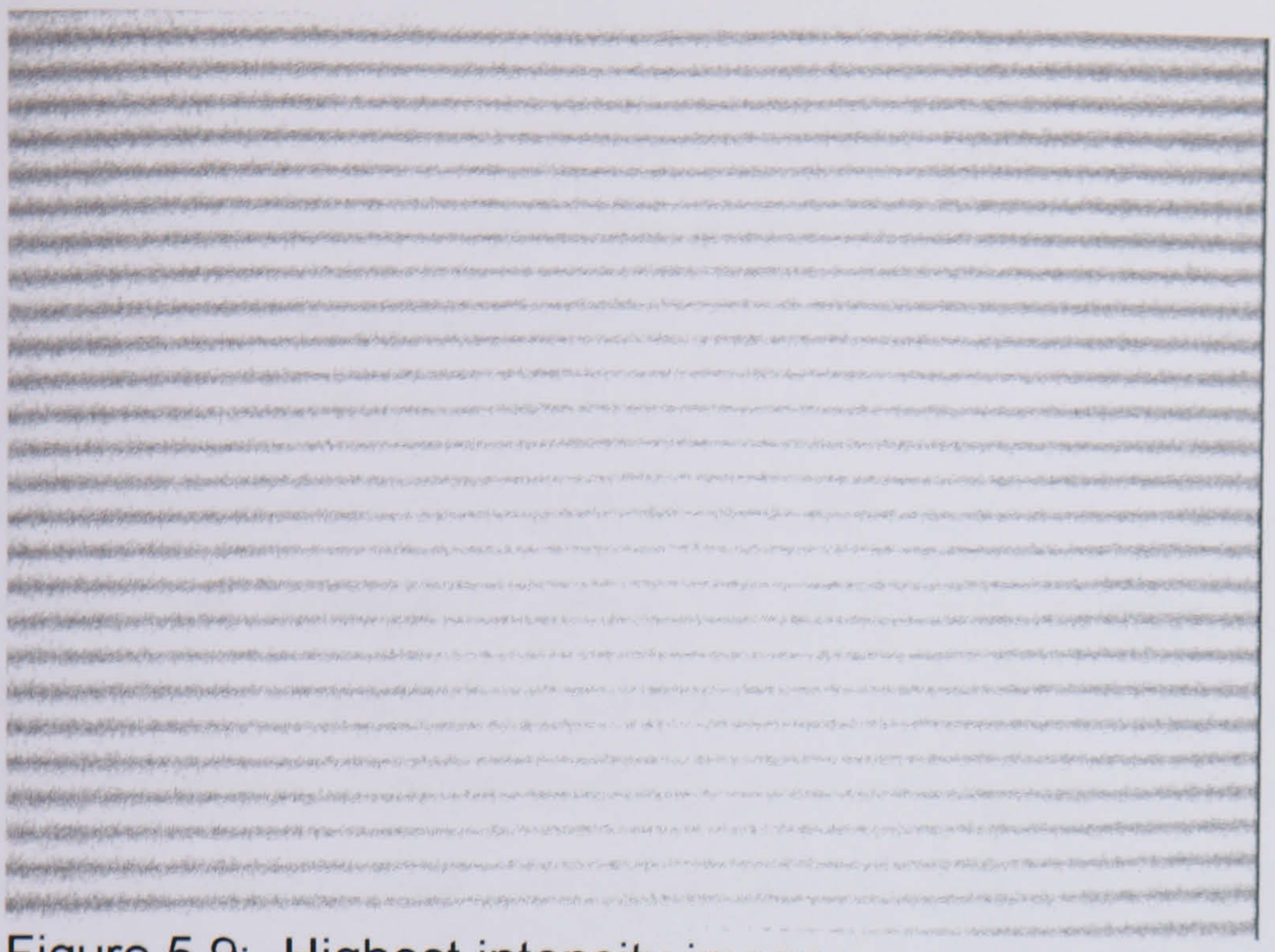


Figure 5.9: Highest intensity image

f-stop	I_{max}	I_{min}	Contrast	Visibility
2.8	240	140	0.5833	0.4118
4	175	75	0.5714	0.4000
5.6	135	65	0.5185	0.3500
8	105	60	0.4286	0.2727
11	90	60	0.3333	0.2500

Figure 5.10: C & V vs. aperture.

The table shows how contrast and visibility vary with camera aperture. It can be seen from the results that calculating contrast gives a range of values approximately 1.5 times that of visibility. This suggests that contrast would be a better value to use, particularly if the values of I_{max} and I_{min} are close together.

Fringe number:

Simply calculating the number of peaks and troughs across a fringe pattern is not an accurate enough representation of the number of fringes present.

This method will not allow for any localised maxima and minima. A better method for calculating this parameter is by mean subtraction. Firstly, the mean intensity value across a fringe pattern is calculated. This value is subtracted from each individual intensity value, leaving an intensity distribution containing both positive and negative values. The number of times the distribution crosses zero is then calculated, the number of zero crossings being twice the number of fringes. The spacing of the fringes can easily be changed using an adaptive interferometer, as previously described. Figure 5.11 shows an intensity profile across a cosinusoidal fringe pattern. The illumination is uniform across the whole image and signal to noise ratio is high. In this case, the mean subtraction method will give a value for the number of fringes present in the image.

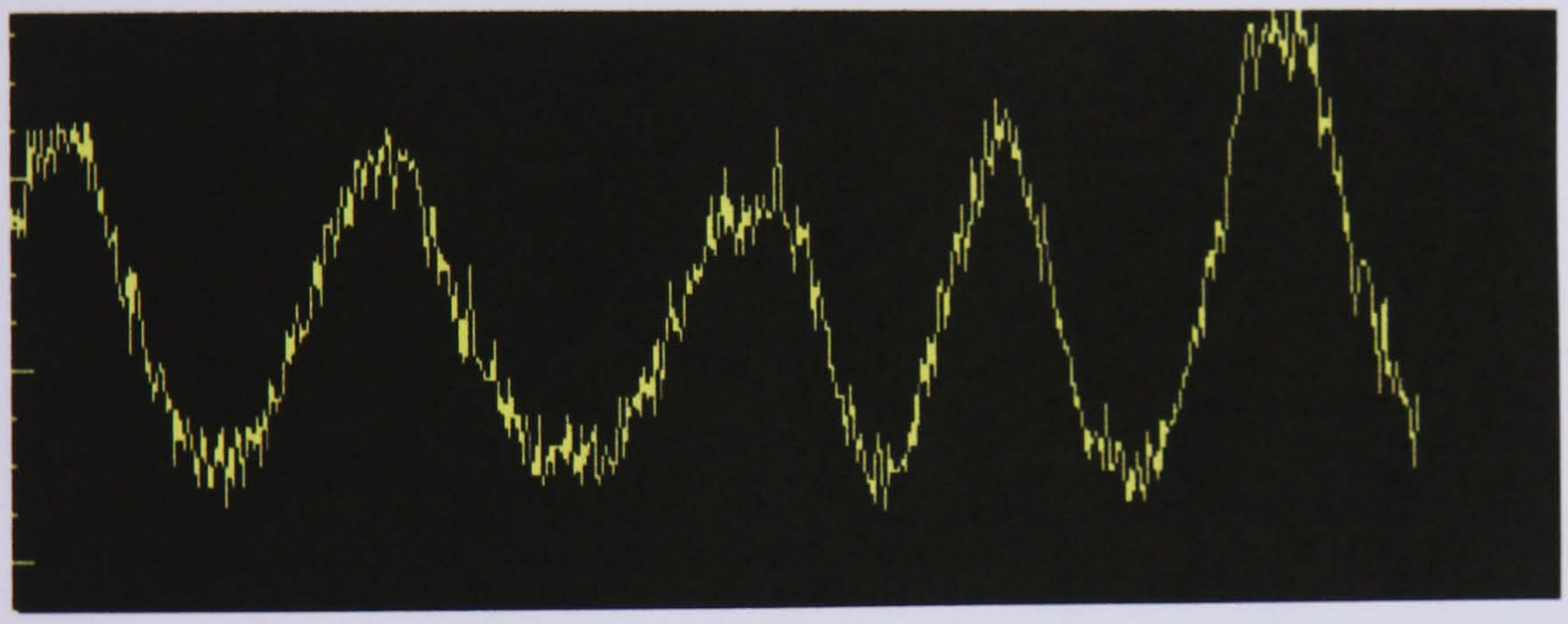


Figure 5.11: Typical intensity profile.

Fringe orientation:

Another parameter, which is easily controllable using adaptive interferometry, is fringe orientation, also referred to as *tilt*. A similar analysis as for fringe number is carried out at various orientations throughout the fringe pattern. The frequency of fringes in a particular direction will give an indication of their orientation.

5.2.3 Experimental methods 1

It was proposed that a neural network be configured to analyse these four parameters which would give an output dependent on the quality of the pattern. The output from the network would then be used in conjunction with the control system of an adaptive interferometer to adjust the fringe pattern until it was optimised.

Separate networks for each parameter

The first method proposed to solve the problem was to use a separate, discrete neural network for each parameter. The idea involved training each network to recognise the validity of a separate parameter and then combining the outputs from the array of networks to produce the final result. A schematic diagram of this layout is shown in figure 5.12.

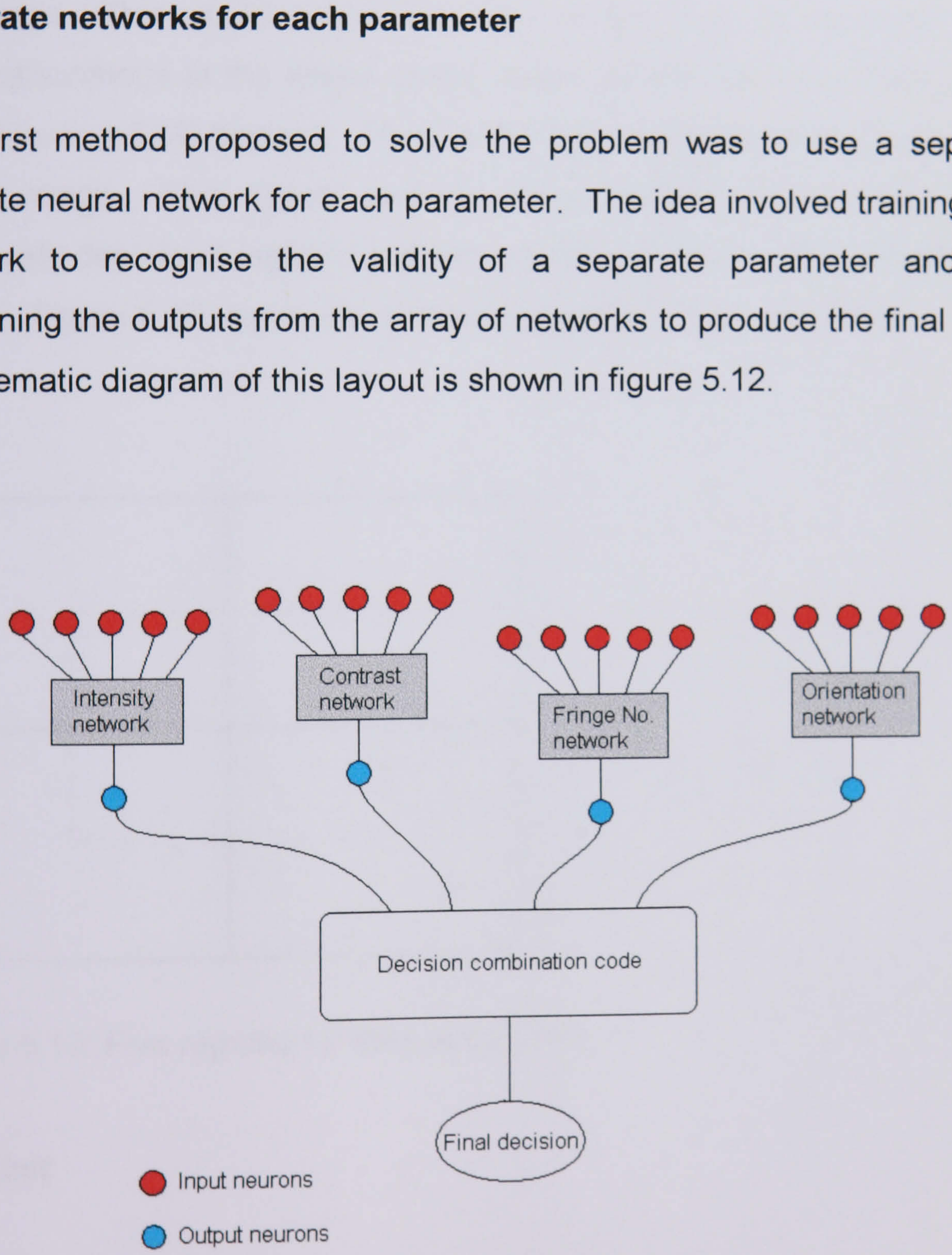


Figure 5.12: Multiple network configuration

For the purpose of training the networks, the image was split into separate regions of interest (ROIs). For the separate ROIs to retain the same aspect ratio as the original, image, it was necessary to keep to a "square" pattern. The smallest number of square ROIs was, therefore, four. Initial experimentation began by dividing the image into four ROIs, each having a size of 256x256 pixels. From this configuration, four contrast values could be calculated for each fringe pattern. It was also decided to take into account values at the centre of the fringe pattern, i.e. the brightest point in the image. This would lessen the effect on the data of any darkening or other aberrations at the edges of the image, which may have been caused by the lens or CCD camera. Thus, a fifth ROI was established at the centre of the image. This region was the same 256x256 pixel square as the previously described regions and was equidistant from all four edges of the image. Figure 5.13 shows the relative positions of the five regions.

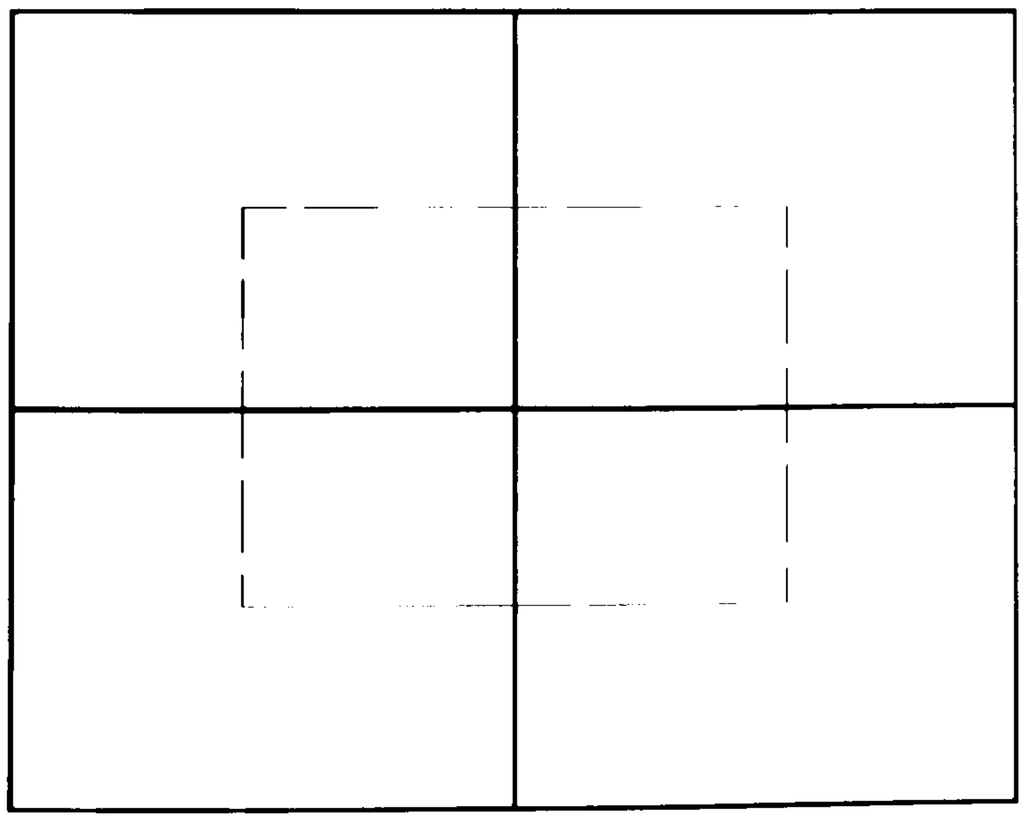


Figure 5.13: Five regions for data acquisition

Contrast

The first experiment used a backpropagation network to determine the contrast of a fringe pattern.

Data acquisition

To calculate contrast, an intensity profile was recorded for each region. The profile was take vertically through the centre of the ROI. Figure 5.14 shows a typical intensity profile of 256 pixels. From this profile it was possible to calculate the maximum and minimum intensity values for the region. Contrast could then be calculated from the equation

$$C = (I_{max} - I_{min}) / I_{max}$$

Application of this equation to each ROI resulted in a total of five values of contrast for each image. To complete the training set, supervised learning techniques required an output value. For this experiment it was decided to label regions whose contrast value was greater than or equal to 0.500 as "good" and those less than 0.500 as "bad". This could be simply represented by a high, or 1.000, value for each good image and a low, or 0.000, for each bad image. Fifty images of varying contrast were used to construct the training set.

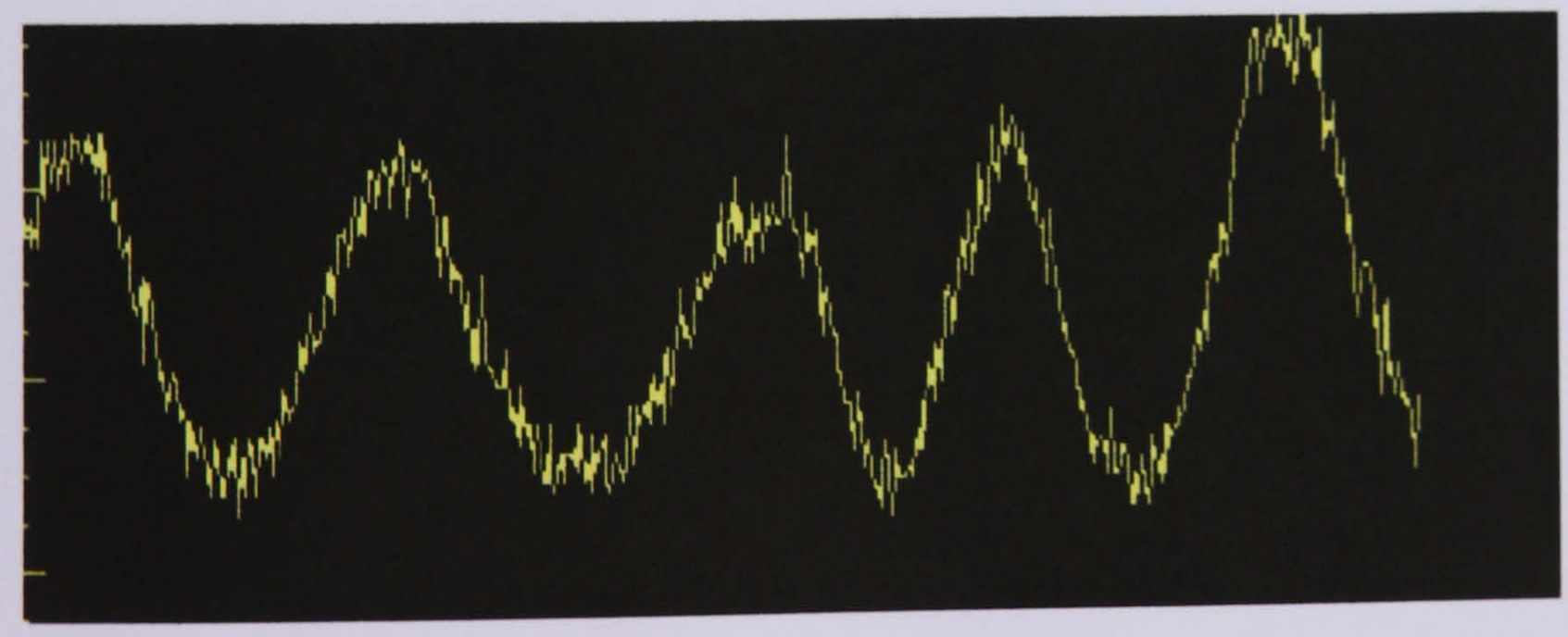


Figure 5.14: 256 pixel intensity profile.

Contrast network configuration

A backpropagation network was initially configured to analyse the contrast data. The configuration stemmed from earlier successes with backpropagation networks and consisted of the following

- 5 input neurons
- 5 hidden neurons
- 1 output neuron
- Sigmoid transfer function
- Normalised-cumulative-delta learning rule

The learning rule was selected to improve training by increasing the likelihood of random data presentation. Figure 5.15 shows the network.

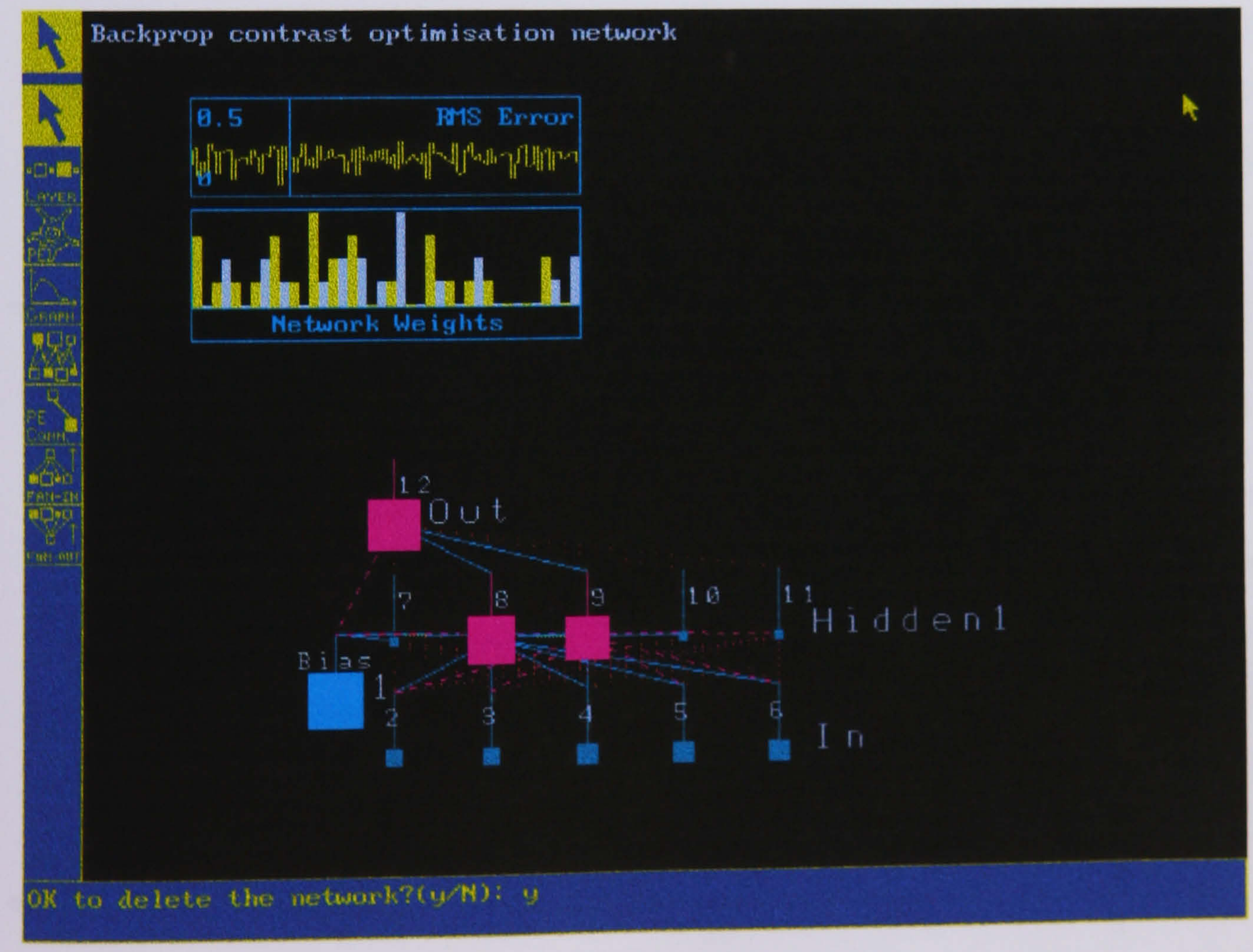


Figure 5.15: Contrast optimisation network.

Training the network

The training limit was set to 10,000 presentations and the network left to train. Initial behaviour of the network appeared to show that training was progressing badly. The RMS error graph showed no convergence and the spread of weights throughout the network appeared random, as was the initial state. Before attempting to rectify the training problem, it was necessary to ascertain how badly the network had actually trained. First, the network was tested using the same data as for the training set. This was done to test if the network had learned to recognise any of the specific data in that set. It appeared that the network had actually learned to recognise most of the data with which it had been presented. To test the network further, it was necessary to test using different data from the training set. A test set was constructed in the same manner as the training set and the presented to the network. The results showed that, although the RMS error and weight distribution pointed to poor training, the network was recognising the difference between good and bad contrast values on 93% of occasions. In each case, the wrongly categorised values were low desired outputs, which were considered by the network to be high.

Number of fringes

Experimentation was carried out to investigate the number of fringes in an image. The first network was a backpropagation network configured to analyse the number of fringes in each region of interest as previously defined. The network utilised 5 input, 5 hidden and 1 output neuron. The training data was taken from the same images as for the previous parameters.

Collecting the training data.

Previous experimentation has shown that better results can be achieved if a neural network is trained using real rather than simulated data. With this in mind, it was necessary to collect a number of representative data from real images in order to train the network. The image was divided into four regions and the number of fringes in each region counted. Firstly, an intensity profile was recorded across each of the regions. The average intensity value was calculated for each profile and this value was then subtracted from each value in that profile. The number of zero crossings was calculated and divided by two to give the total number of fringes in the region. The fringe number was calculated in this manner to avoid the effects of localised maxima and minima at the peaks and troughs of the intensity profiles. Intensity profiles were recorded for four regions in thirty images containing straight fringes of varying period. This enabled a set of 30 training vectors, each containing four fringe number values to be constructed. To complete the training vectors, the corresponding fringe number was added to present to the single output neuron.

Training the network

The training limit was set at 50,000 presentations and the network was left to train. At the end of the training period, the distribution of weights appeared to be very poor, as every weight was either high or low with no intermediate values present. The network was tested, but results were poor, with all outputs showing a value of 0.4285, regardless of input.

The hidden layer was pruned to investigate the gradual removal of hidden neurons. One by one, hidden neurons were removed and each time the network was trained with the same data. The results for the pruning exercise were as shown in figure 5.16.

No of hidden neurons	Effect
5	All outputs reading 0.5243
4	All outputs reading 0.5364
3	All input weights high, all hidden weights low, all outputs reading 0.5364
2	All weights high, all outputs reading 0.5364

Figure 5.16: Effect of pruning hidden neurons.

As the removal of hidden neurons was having no positive effect on the results, experimentation with multiple hidden layers was conducted. The hidden layer was returned to its original state and a second hidden layer was cloned from it. This was connected between the existing hidden and output layers. Training commenced using the same data as for the previous experiments and continued for 50,000 presentations. During the entire training period, the RMS error remained steady at approximately 0.5 and when training was complete, the network showed a reasonable spread of weight values. However, after testing and analysing the results file, the same phenomenon appeared to be occurring. All the output values were identical, regardless of the input.

Although backpropagation networks had been both simple to implement and reliable in their results for the earlier wrap detection problem, this initial experimentation showed them to be less suited to the problem of optimisation of this parameter. It was therefore decided to investigate the use of an alternative network type for this analysis.

The same problem of optimising fringe number was tackled using a General Regression Network (GRN), the theory of which has already been discussed in chapter 3. In order to make use of the existing training and test data, the network was configured to have five input neurons and one output neuron. The network does not contain a conventional hidden layer, but two

intermediate layers. These are the pattern and summation layers. The smallest default values for these intermediate stages given by NeuralWorks were 50 pattern and 2 summation neurons. The network, shown schematically in figure 5.17, was configured with the following parameters:

5 input neurons

50 pattern neurons

2 summation neurons

1 output neuron

Euclidean summation function

Tau = 1000.00

Reset factor = 0.0000

Radius of influence = 0.050

Sigma scale = 1.0000

Sigma exponent = 0.5000

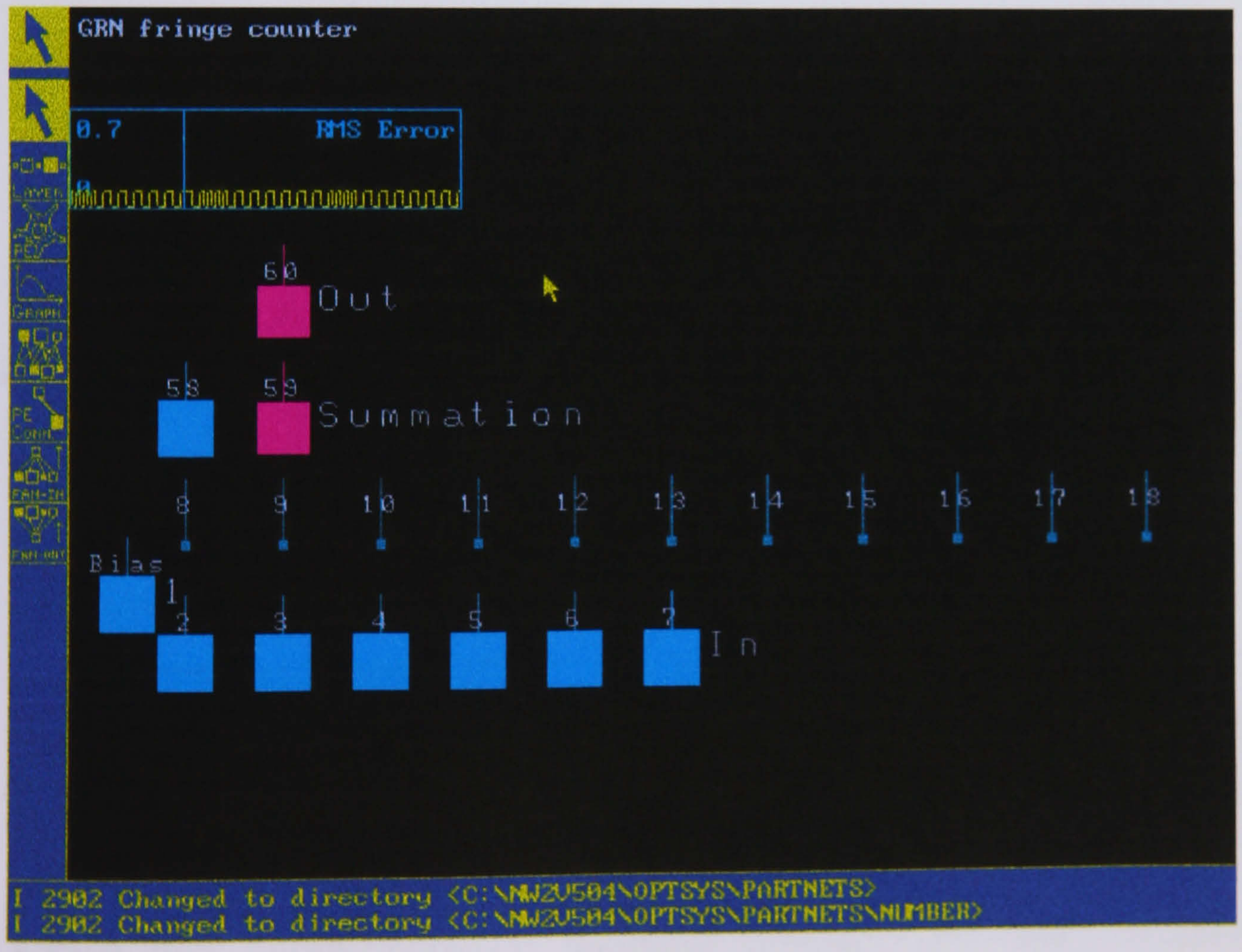


Figure 5.17: Fringe number GRN.

The initial network was trained as for previous experiments. During training, the RMS error appeared as shown in figure 5.18. The graph showed this throughout the duration of the training session. After 50,000 presentations

of the training data, training ceased, although there appeared to be an unusually large number of high weights. In spite of the weight distribution and unusual behaviour of the RMS error, the results showed an accuracy of 96.9%.

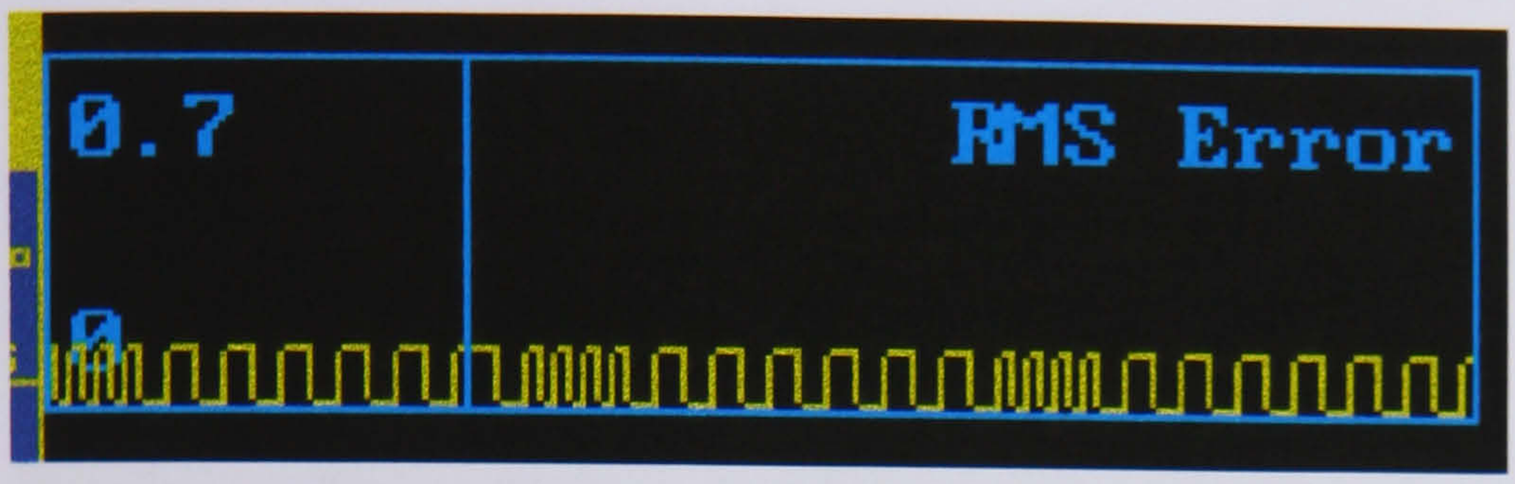


Figure 5.18: RMS error graph.

Mean intensity

Calculation of mean intensity followed the method shown previously in the calculation of fringe number. The same ROIs were used for collection of data and a single intensity profile taken across the centre of each. The standard mathematical method for mean calculation was employed, i.e. the sum of all values was calculated which was then divided by the number of values. The result of this was five mean intensity values for each image. As an ideal fringe pattern should contain equal regions of light and dark pixels, with intensity ranging from 0 to 255, it follows that the average intensity value for the image should be of the order of 127. A training set was created with low intensity values being assigned a low (0) value and high intensity values being assigned high (1) values. An intensity value that fell into the "good" region was assigned a value of 0.5. The network outputs were scaled to values between 0 and +1. Consultation of the NeuralWare reference guide [2] and results obtained from previous experimentation have shown that backpropagation networks function best when outputs are confined to this region. A backpropagation network was constructed as shown in figure 5.19 for analysis of this parameter. The network was set to train for 50,000 presentations. During the training process, the RMS error graph appeared to remain high, refusing to reach its convergence criterion

before all 50,000 passes were complete. The weight spread approximated a Gaussian distribution, but showed high values for extreme low and high weights. After training, the network was tested, again using different data from that used in the training set. The results file shows that the network achieved correct results on 81% of occasions. While this value is reasonably high, it does not compare favourably with the results given by previous experiments, which have been showing success rates in the region of 90-95%.

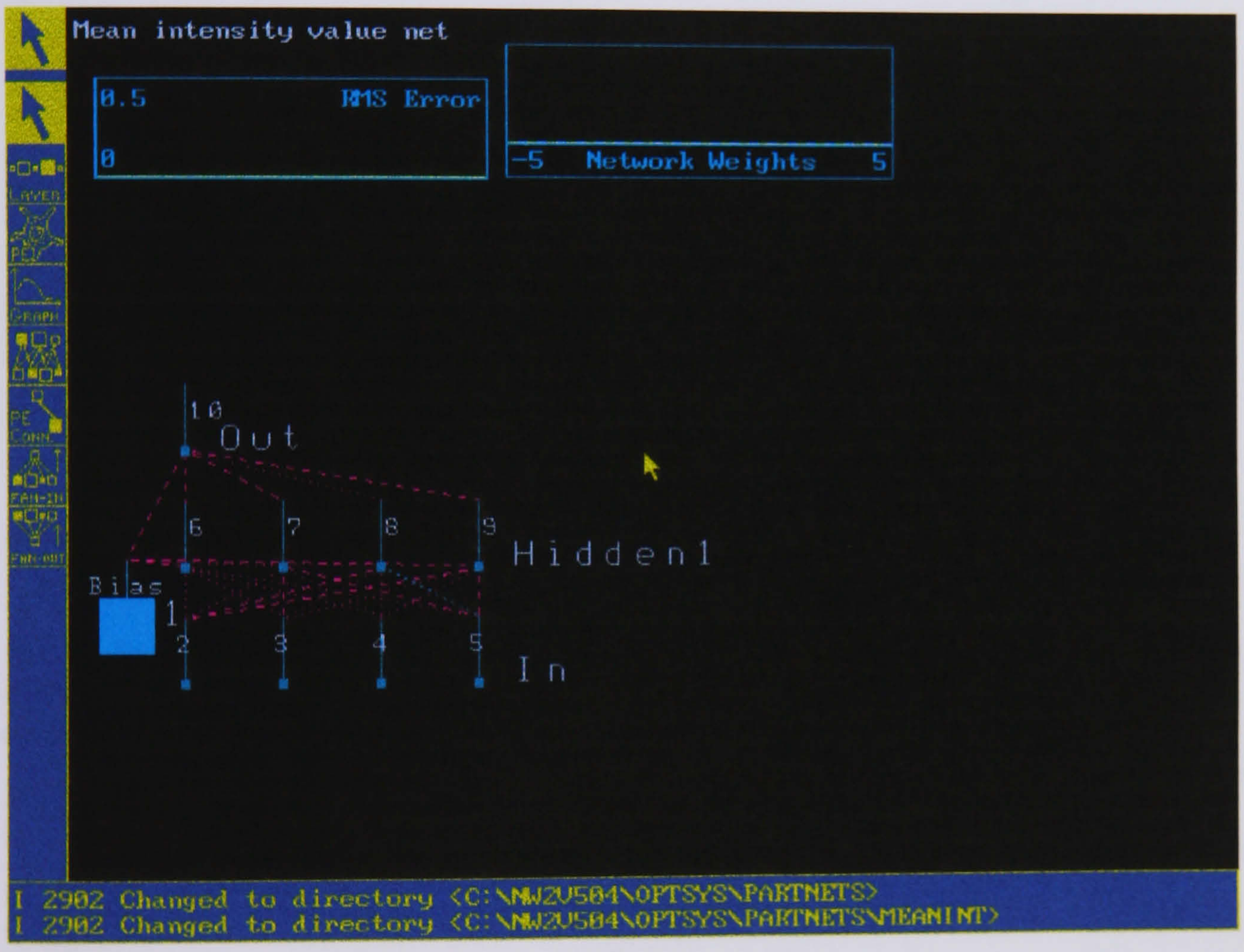


Figure 5.19: Backpropagation network for mean intensity analysis

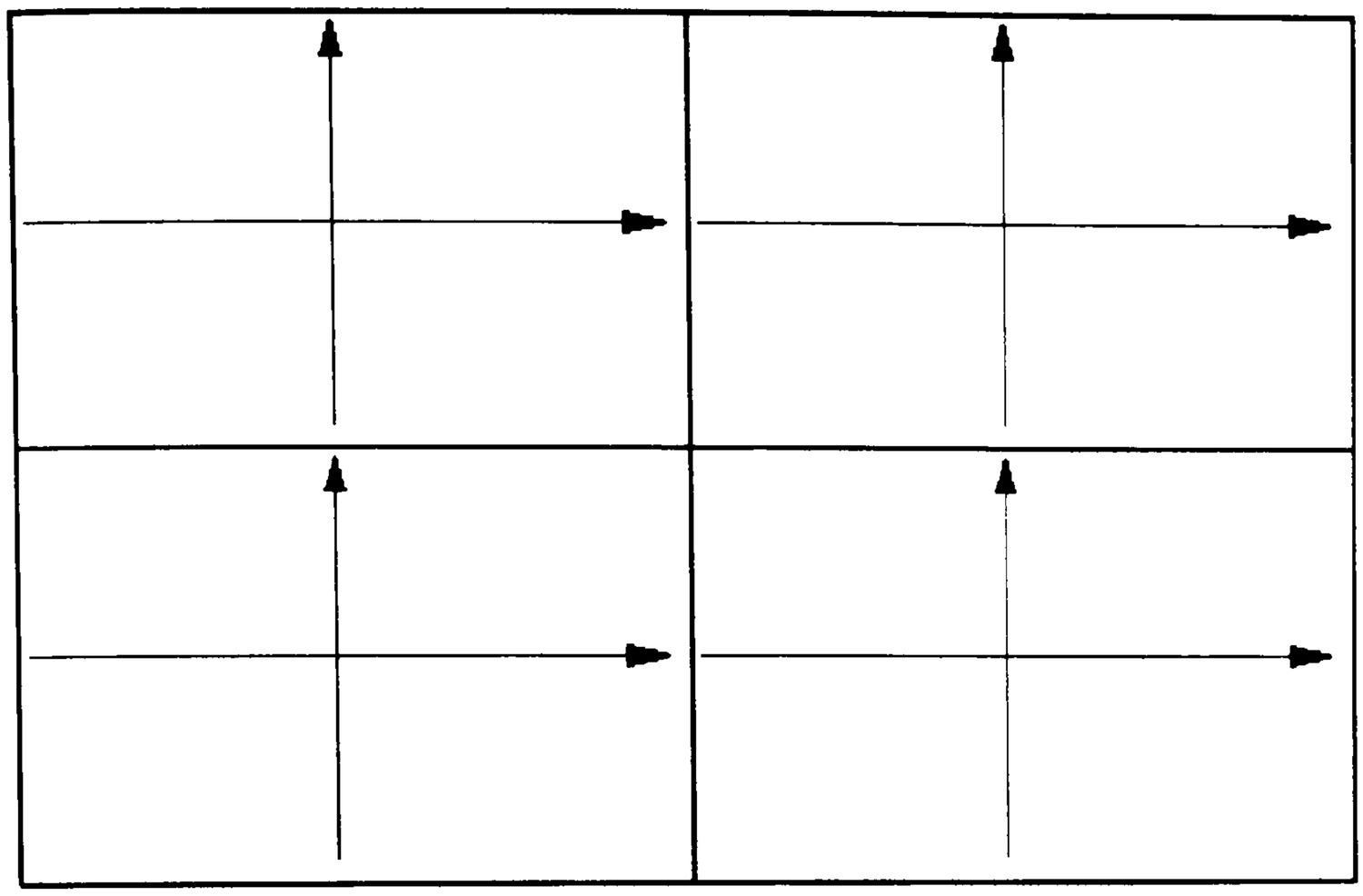
Fringe orientation

Two methods for determining the fringe orientation were considered:

1. Cross-profiles.

In each of the regions of interest, both horizontal and vertical intensity profiles were recorded. Figure 5.20 shows the directions in which the

profiles were recorded. The profiles were then compared to give an indication of the orientation of the fringes in that particular region of interest. The result from this method is five values for orientation.



NB Profiles were also recorded in the centre ROI

Figure 5.20: Cross-profiles

2. Fan-out profiles

A series of intensity profiles was recorded throughout the whole image in the directions shown in figure 5.21. The number of fringes in each direction was counted as previously described and the counts compared. The result from this method is five fringe counts, one for each direction shown in the diagram. This also provided the same number of inputs as for the previously described parameters.

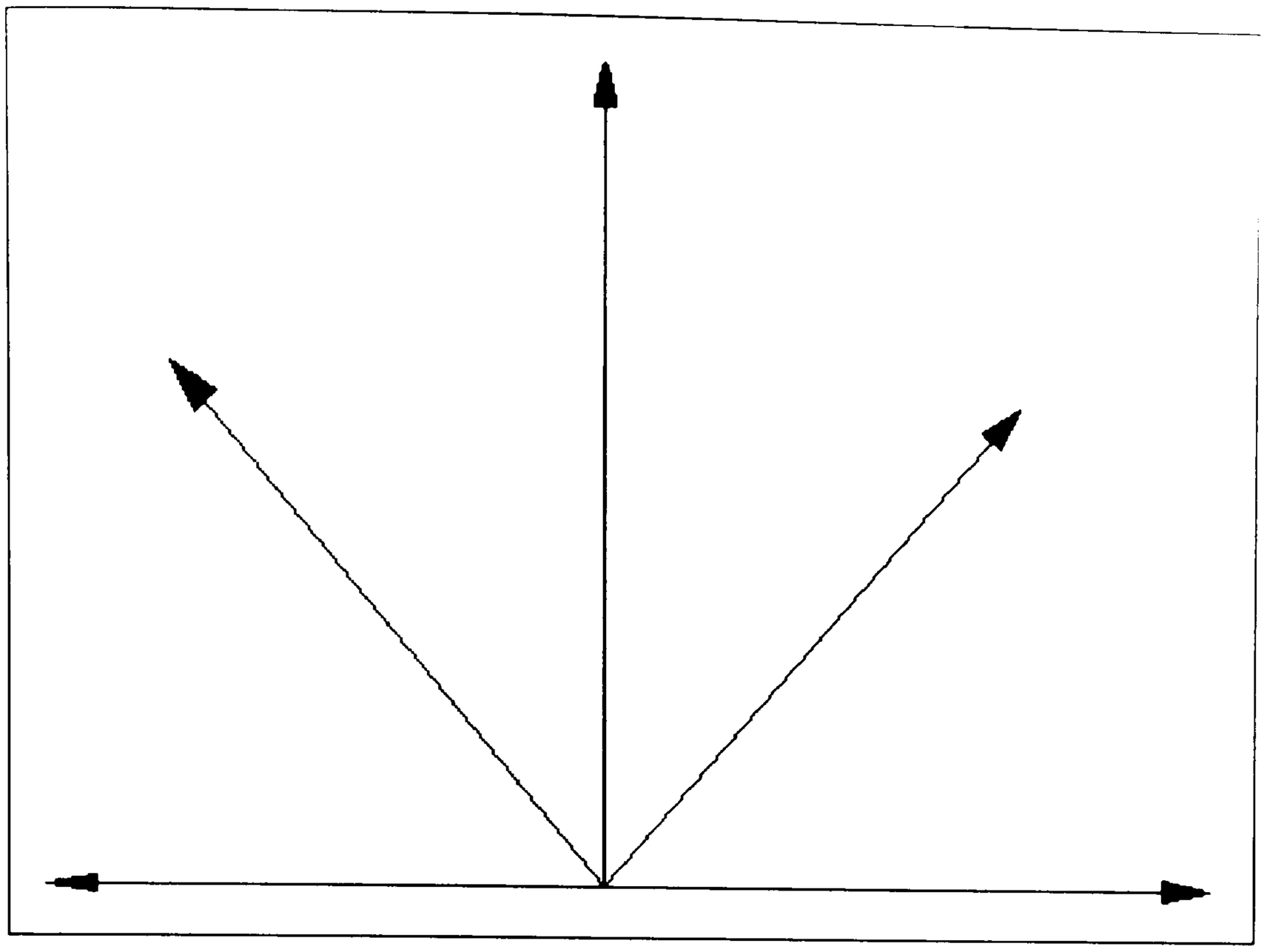


Figure 5.21: Fan-out profiles

From the variation in number of fringes with direction, it is possible to determine the orientation of the fringes in the pattern of interest. The problem remained of how to quantify the output of the network to determine the actual orientation of the fringes. Using a network of this size would cause problems in determining exactly how far from the horizontal the fringes were, so the task of defining simply whether the fringes were horizontal or not was initially addressed. The most complex output that the network would be asked to give was in which direction the fringes were tilted. The first experiment used the following system:

1. Determine fringe counts in each direction of the "fan-out" profiles.
2. If fringes are horizontal, assign a zero to the desired output.
3. If fringes tilt anticlockwise, assign -1 to the desired output.
4. If fringes tilt clockwise, assign +1 to the desired output.

A general regression network was configured to test this theory. Figure 5.22 shows the network used for this experiment.

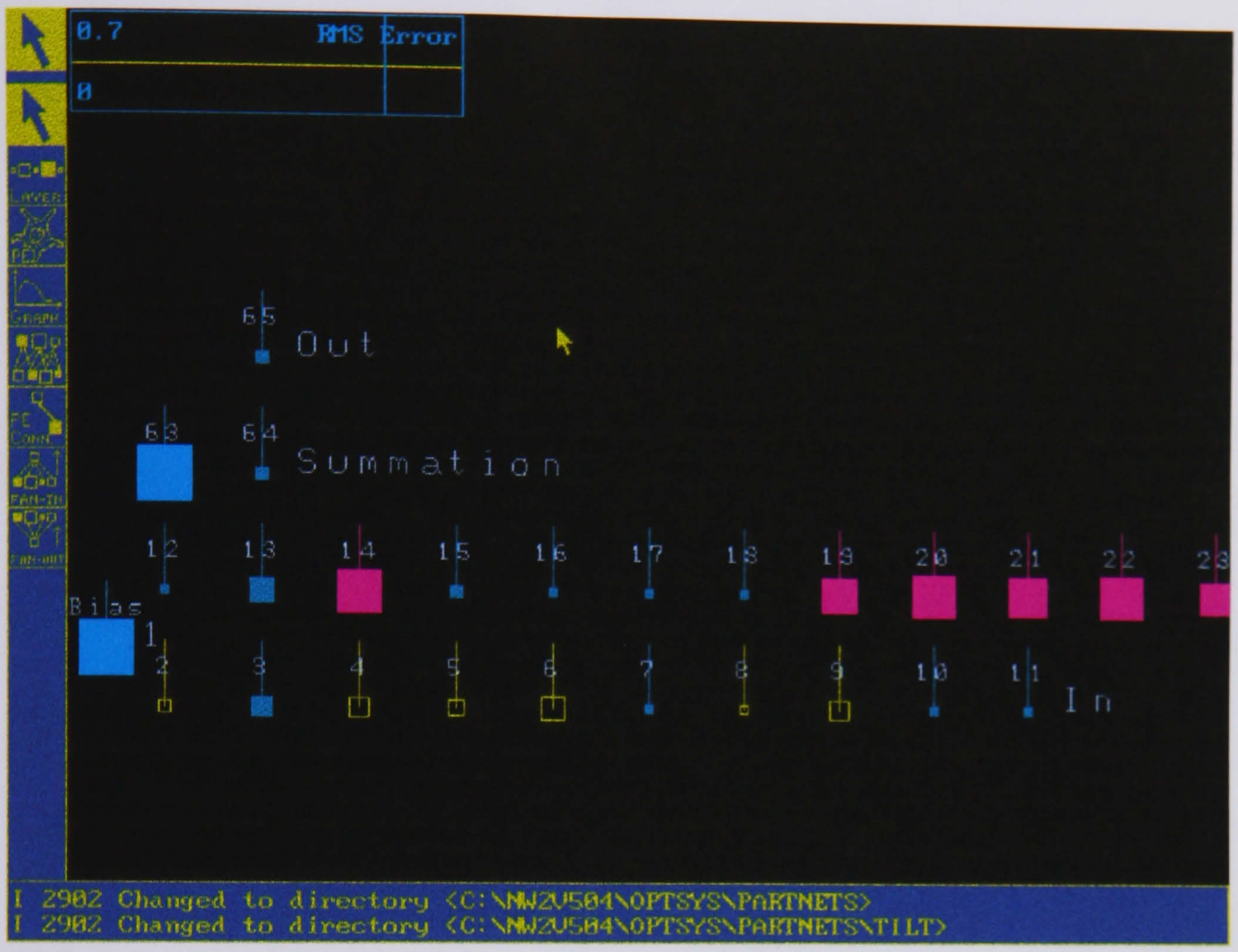


Figure 5.22: GRN for orientation analysis.

The training parameters were as described for previous networks. The network was trained for 10,000 presentations and tested. The network outputs for this experiment gave unsatisfactory results. Instead of showing the required directions, all outputs were in the same region, showing low values, all of which were positive. Again, previous experimentation was referred to. As described earlier, best results are achieved if the network outputs are between 0 and +1. In order to achieve better results with the orientation network, the training data were re-scaled to try and improve training. Instead of using values between -1 and +1 for the desired outputs, the values were re-scaled to be in the region 0 to +1. The new outputs were decided thus:

- If fringes are horizontal, assign a value of 0.5 to the desired output.
- If fringes are tilted anticlockwise, assign 0.
- If fringes are tilted clockwise, assign +1.

The network was re-trained using the re-scaled outputs. During the training period, the spread of weights approximated a Gaussian distribution, but the RMS error remained high and did not reach its convergence criterion. The first results file showed a success rate of 88%. Experimentation was carried out to reduce the number of pattern neurons to ascertain the effect on successful training. The default value set by the NeuralWorks package was 51 and the results for this are described above. The minimum number of pattern neurons allowed by the package is 10, so a network employing this minimum configuration was trained. The number of pattern neurons was gradually increased; the results were as shown in figure 5.23. It can be seen that increasing the number of pattern neurons gradually decreases the accuracy of the network until 14 are used, when accuracy increases to 88% and remains constant at this value. It follows from this that the minimum number of pattern neurons required to achieve maximum accuracy is 14.

Number of pattern neurons	**Network overall percentage accuracy (%)**
10	84
13	84
14	88
15	88
20	88
25	88
51	88

Figure 5.23: Effect of varying number of pattern neurons.

5.2.4 Experimental methods 2

The use of multiple networks to address the optimisation problem is extremely flexible, as, if it is necessary, each parameter's network can be re-trained separately. However, the need for extra processing to combine all of the network outputs adds a further level of complexity to the overall system. It was decided to follow a different approach to the problem by using a single, larger network to analyse all parameters simultaneously. Figure 5.24 shows a schematic diagram of the new "global" network.

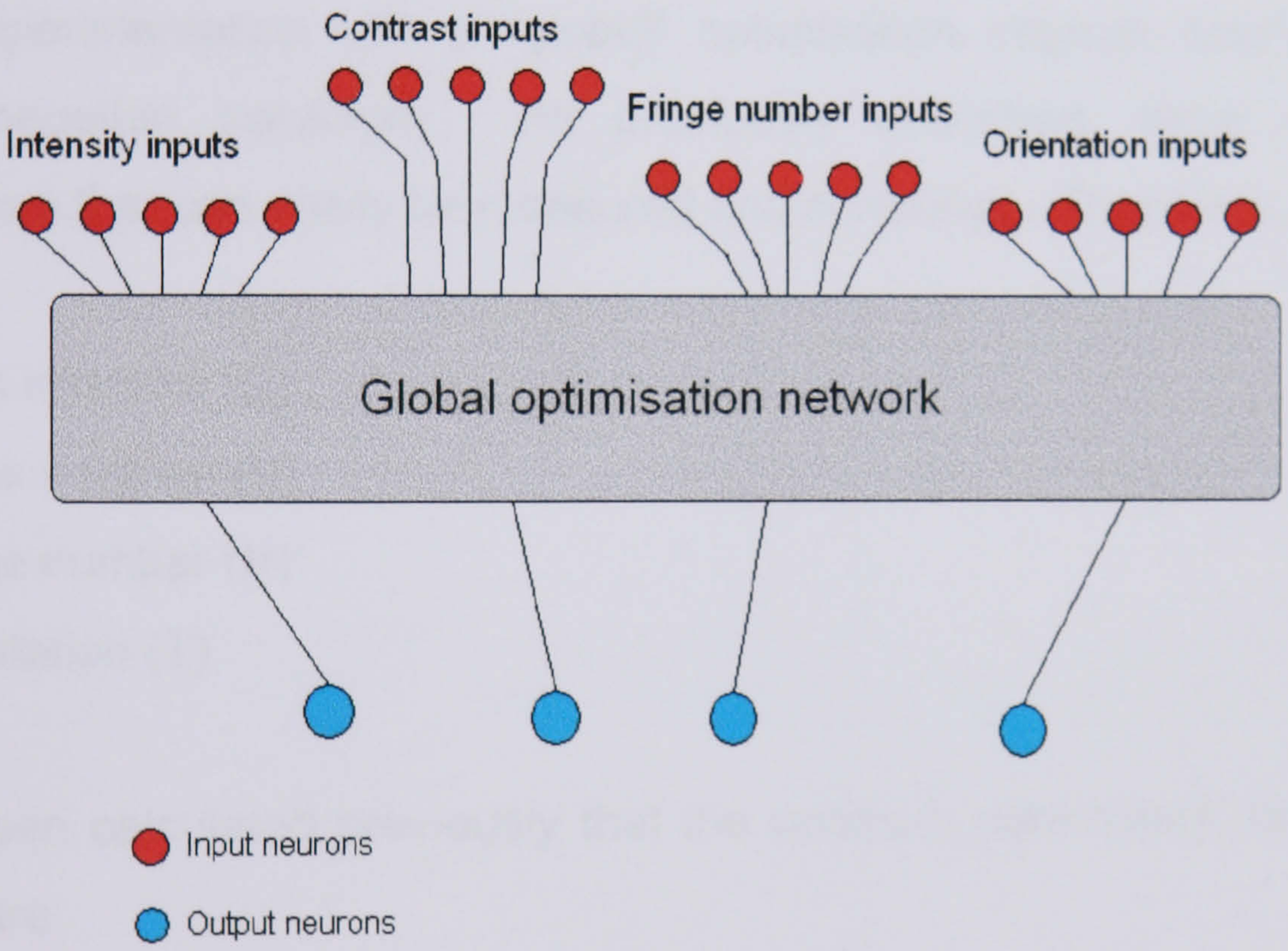


Figure 5.24: The "global" optimisation network

Data acquisition

The data was acquired in a similar manner to those experiments already described. The image was split into four regions and intensity profiles read in the centre of each region and a fifth intensity profile was read from the centre of the image, where it was likely to be brightest. From these five profiles, the three parameters mean intensity, number of fringes and visibility

could be calculated as described earlier in section 5.2.2. Instead of using four separate training sets, a larger, single data set was acquired. The net result of the data acquisition exercise was a set of data that could easily be turned into training vectors for a number of different paradigms. Each image yielded 20 input values, 5 for each parameter. An output response for each parameter in each training vector was added according to the user's decision on image quality to complete the training set. An example of a global training set is shown in appendix 3.

Experiments with backpropagation

Initial experimentation with a "global" optimisation network employed the backpropagation paradigm. As previously described, there are four parameters that can easily be measured and controlled. These are:

- Mean intensity (I_m)
- Fringe contrast (V)
- Fringe number (N)
- Orientation (T)

It has been calculated previously that the optimum parameters for a fringe pattern are:

$I_m = 127$

$V \geq 0.7$

$N = 64$

$T = 0.5$ (horizontal)

The tolerance bands for these parameters, as seen by the network, are

$100 \leq I_m \leq 150$

$55 \leq N \leq 65$

$0.4 \leq T \leq 0.6$

$0.5 \leq V \leq 1$

The acquisition of data for the global network was carried out in a similar manner as previously described. However, instead of collecting data for individual parameters, all data was collected simultaneously. The result of the data acquisition exercise was a set of 5 data per parameter, giving a total of 20 data values for each training vector. Instead of a single output, the network provided 4 outputs, one corresponding to each parameter. An example of a training set of this type is shown in appendix 3. The network used for this experimentation is shown in figure 5.25. The backpropagation network consisted of a single input neuron for each data value and 20 hidden neurons.

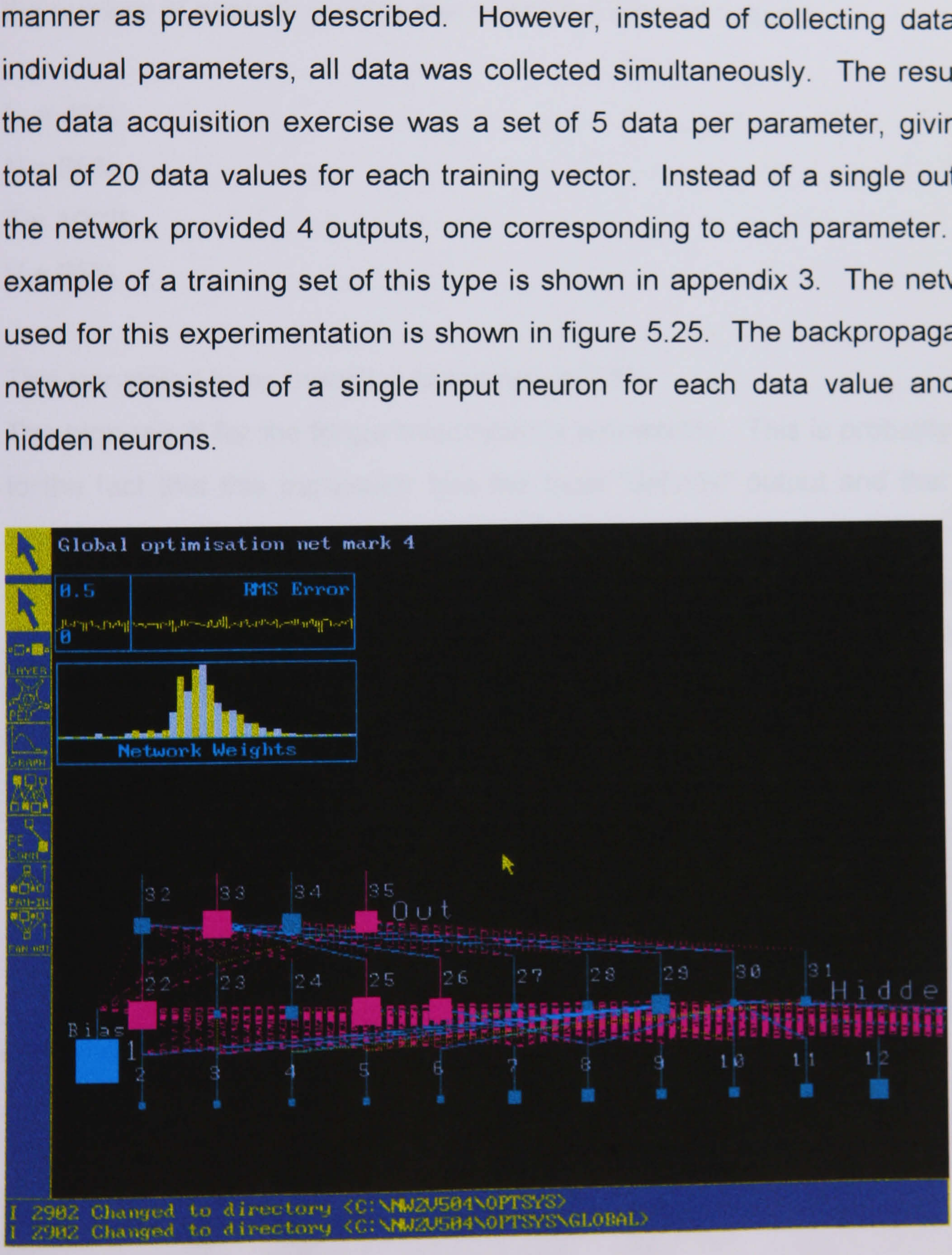


Figure 5.25: Global optimisation network.

The network was initially trained for 20,000 presentations. The RMS error remained high throughout training and the weight histogram showed a tendency towards overtraining. The results file showed limited success at this initial attempt. With the possible tendency towards overtraining, experimentation with fewer hidden neurons was carried out. Training and testing with 19 hidden neurons produced the following results. As before, the "success rate" of the exercise is given. This is shown as a percentage of the number of network outputs that match the desired outputs.

I_m = 40%

N = 65%

T = 100%

V = 95%

This translated to an overall success rate of 75%.

The high result for the fringe orientation is noteworthy. This is probably due to the fact that this parameter has the most "definite" output and that it is relatively easy for the network to learn three definite "states". The contrast result was also promising, with errors tending towards "low" values. I_m and N values were less satisfactory. The network appeared to be confused as to the validity of some of the N values. The parameter causing most concern was that of I_m, as, according to the results file as all I_m values were passed as "good". In an attempt to remedy these problems, new training data were taken from real images. The existing network was then trained for 20,000 presentations. Along with the new training data, experimentation was carried out with further reductions in the number of hidden neurons. The results of this experimentation are shown in figure 5.26.

Experiment	I_m %	N%	T%	C%	Overall %
Original data & network 20,000 passes	40	65	100	95	75
I_m re-scaled to $0 \leq I_m \leq 1$ 20,000 passes	91	57	100	98	87
I_m re-scaled, 50,000 passes	97	86	100	95	95
Original data, 18 hidden neurons, 20,000 passes	42	45	100	98	71
I_m re-scaled, 18 hidden, 20,000 passes	94	45	100	89	82

Figure 5.26: Results for training the global network.

The results show that the best success rate was achieved with the original network with the I_m values re-scaled to lie in the region $0 \leq I_m \leq 1$. Best results were also achieved when training was extended to 50,000 data presentations.

5.2.5 Comparison of the two methods

Experimentation has shown that it is possible to use a neural network to make decisions regarding the quality of a fringe pattern. The two methods described in section 5.2.2 show the relative merits of using either a separate network for each measurable parameter or a single network to analyse all parameters simultaneously. The basic difference in the two methods lies in their computational complexity. The use of separate networks for each parameter has the advantage of each network's ability to be trained or re-trained individually. If the performance of a network for one parameter becomes questionable, it can be re-trained singly, using a much smaller data set. This also means that training for a single parameter will not affect the network outputs for the other parameters. However, the necessity to combine the network outputs to produce a final result adds an extra level of

complexity to the system. The use of a "global" optimisation network, while not as flexible when re-training is required, combines the whole analysis into a more convenient package. The results shown in appendix 3 show that re-training can affect the outputs for other parameters. This, however, is not always detrimental to the system's operation. The net result of the combined global network is a simple, single piece of code to accomplish the complete analysis.

5.3 An operational optimisation system

When validation of the technique was complete, a system for driving the adaptive interferometer was configured, thus producing a system as shown in the schematic diagram in figure 5.27.

The computer was connected to both the adaptive interferometer and the CCD camera to achieve the closed loop system previously described. The camera was connected to a frame store within the PC to enable video frames of the fringe patterns under analysis to be taken. From these video frames, the above parameters were calculated. The results of these calculations were then passed to the neural network. The output from the network was passed to software driving the adaptive interferometer, completing the closed loop system that enabled the fringe pattern to be optimised before any analysis took place. The block diagram in figure 5.28 represents the closed loop system.

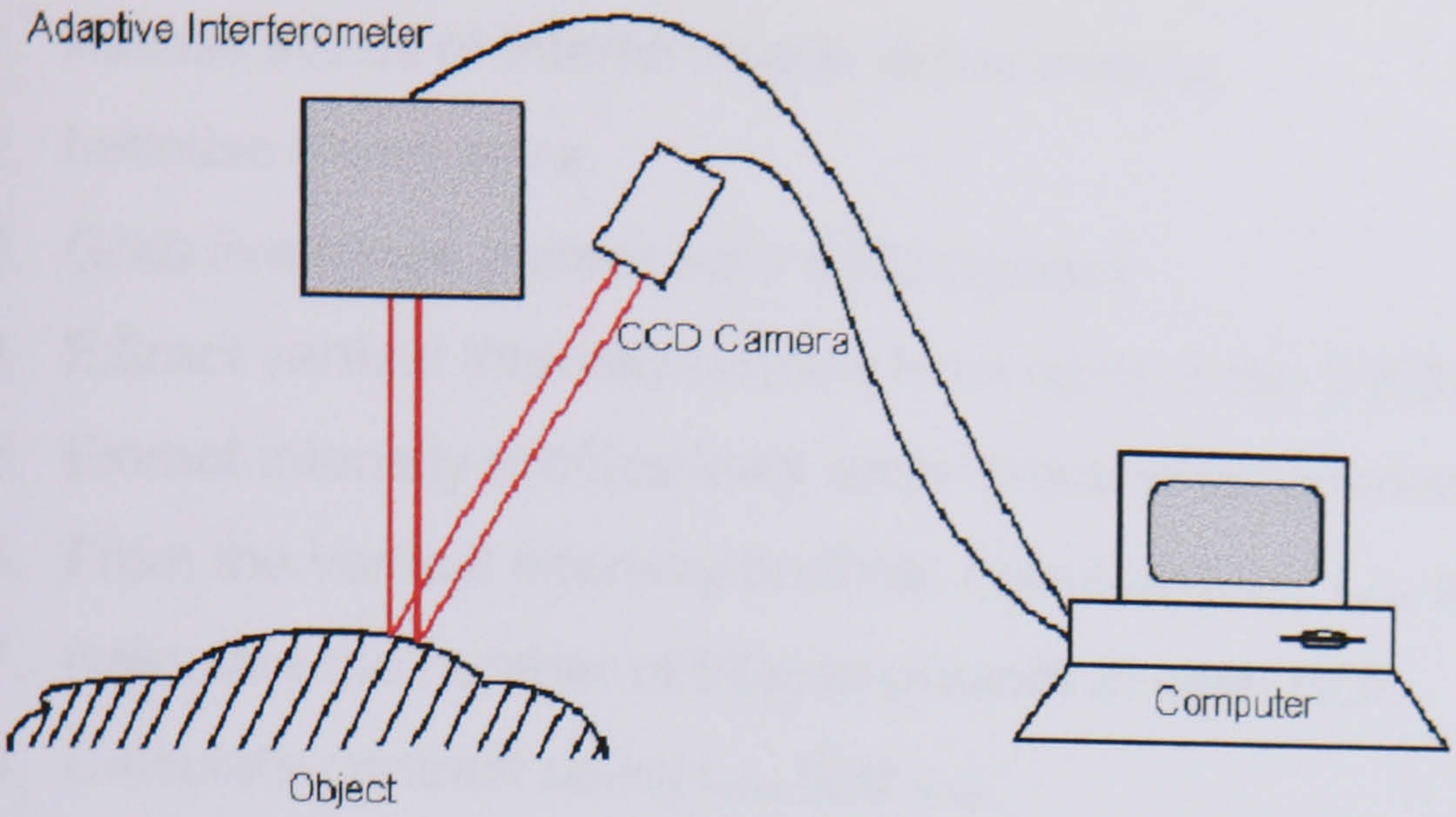


Figure 5.27: The closed-loop optimisation system

5.4 Evaluation of the optimisation system

The experiments described above have shown that it is possible to ascertain the quality of a fringe pattern and, therefore, its suitability for measurement of surface contour before actual measurement takes place. Two alternative approaches to the problem have been made and compared in section 5.2. This comparison has shown that the second approach, using a single network to analyse all parameters simultaneously, is the most effective. The fully trained network was FlashCoded as described earlier in the chapter. This resulted in "C" code that replicated the trained network's behaviour. Appendix 4 contains an example of the FlashCode for the global network. To achieve a fully operational optimisation system, it was necessary to embed the FlashCode into a program to perform all necessary data manipulation. The optimisation system was designed to operate in the following manner:

1. Assess status of interferometer driver motors.
2. Initialise frame store.
3. Grab live fringe pattern from CCD camera.
4. Extract vertical intensity profiles from each of the 5 ROIs.
5. Extract intensity profiles from each direction for orientation calculations.
6. From the vertical intensity profiles, calculate I_{mean}, I_{max} and I_{min}.
7. Calculate the number of fringes present in each ROI.
8. Calculate contrast using I_{max} and I_{min}.
9. Present the 20 values calculated in 4 to 8 above to the optimisation network.
10. Analyse the output from the network and if any faults exist pass a control signal to the interferometer drivers to index the fringes in the correct direction or alert the user to the status of intensity and contrast.
11. Analyse the fringe pattern again to ascertain the effect of this change.

This is repeated until the network recognises a fringe pattern that conforms to all the optimisation criteria and which will be suitable for measurement or until a default number of iterations has been reached. For the purpose of experimentation, this value was set to 15. A finite number of iterations was specified, as some fringe patterns are unlikely to meet any of the optimisation criteria. These are ones that contain faults such as excessive noise or where the object surface does not reflect the fringe pattern. In cases such as this, no amount of adjustment of the fringe pattern will produce a satisfactory result.

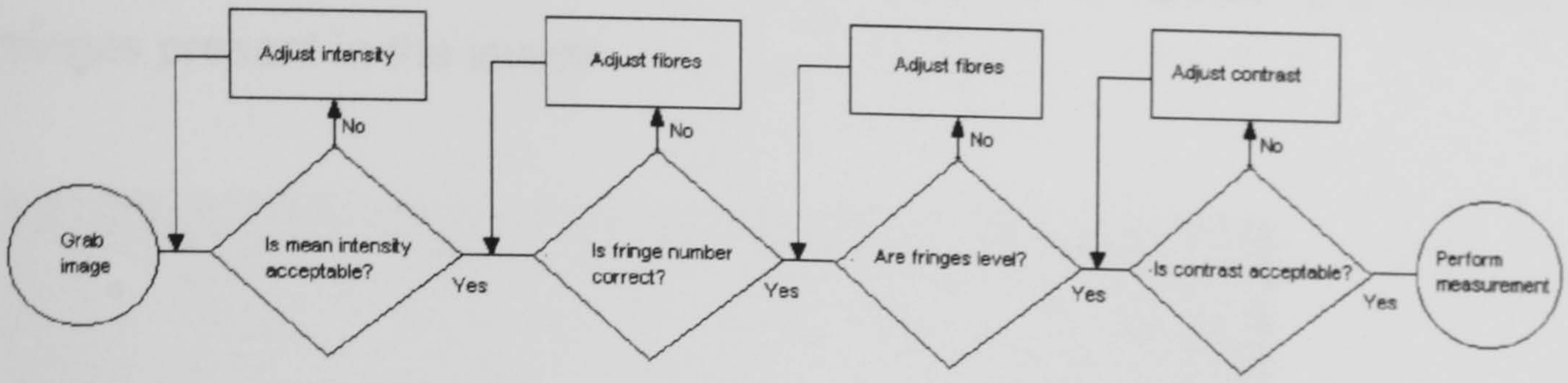


Figure 5.28: Block diagram of the final optimisation system.

The following figures show how the system behaves when presented with a poor quality fringe pattern. Figure 5.29 shows the initial fringe pattern presented to the optimisation system. The pattern was projected onto a flat, matt white surface and the interferometer adjusted to give too many fringes. Here, the fringe number is 85, 19 more than an ideal pattern. The other parameters were left in the "ideal" range for the purpose of this initial demonstration.

Figure 5.29: Pattern with too many fringes

Figure 5.30(a) shows the fringe pattern after the first iteration. The neural network has analysed the image and reached its first conclusion. Figure 5.30(b) shows the output from the network for this iteration. The network has decided that there are too many fringes in the image and moved the

interferometer stepper motors a finite amount to reduce the number of fringes present in the image.

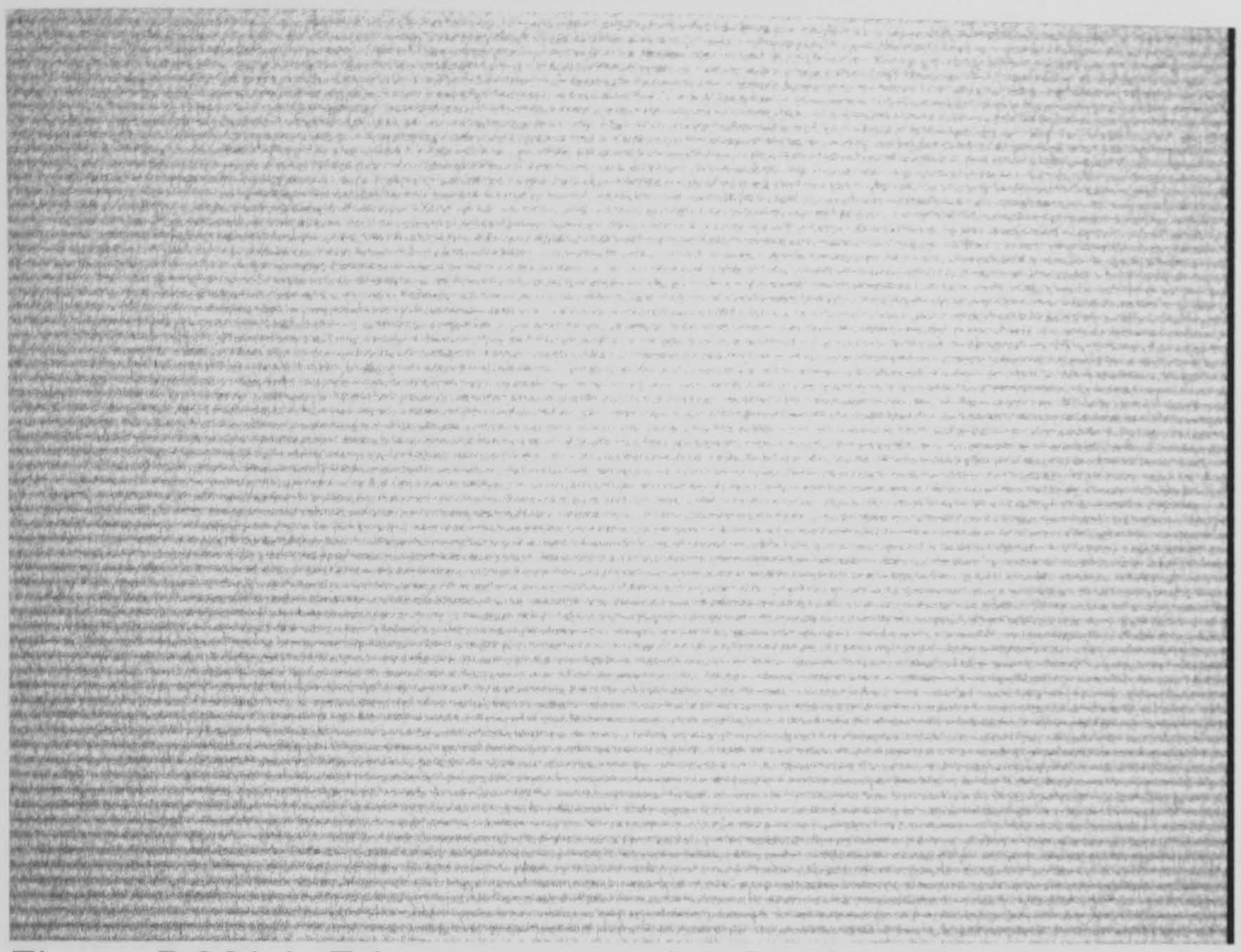


Figure 5.30(a): Fringe pattern after first iteration.

```
Backprop optimisation system version 4.30.
Current MotionMaster error state is: E00 NOERROR
** Loop iteration number 1 **
Snapping current image…

Mean values are:       0.411000 0.122000 0.123000 0.136000 0.145000
Fringe counts are:     0.310000 0.300000 0.360000 0.430000 0.430000
Visibility values are: 0.463415 0.460526 0.458599 0.413333 0.445483
Tilt values are:       0.550000 0.500000 0.430000 0.480000 0.470000

Network outputs are:   0.282032 0.826404 0.367829 0.513631

Mean intensity is too low.
There are too many fringes in this image
Visibility is OK
Fringes are level

Adjusting fibres…
Current MotionMaster error state is: E00 NOERROR
```

Figure 5.30(b): Network outputs for 1st iteration.

Figure 5.31 shows the output from the system after 7 iterations. The system has decided that, after adjusting the interferometer 7 times, it had achieved an "ideal" image, with the optimum number of fringes.

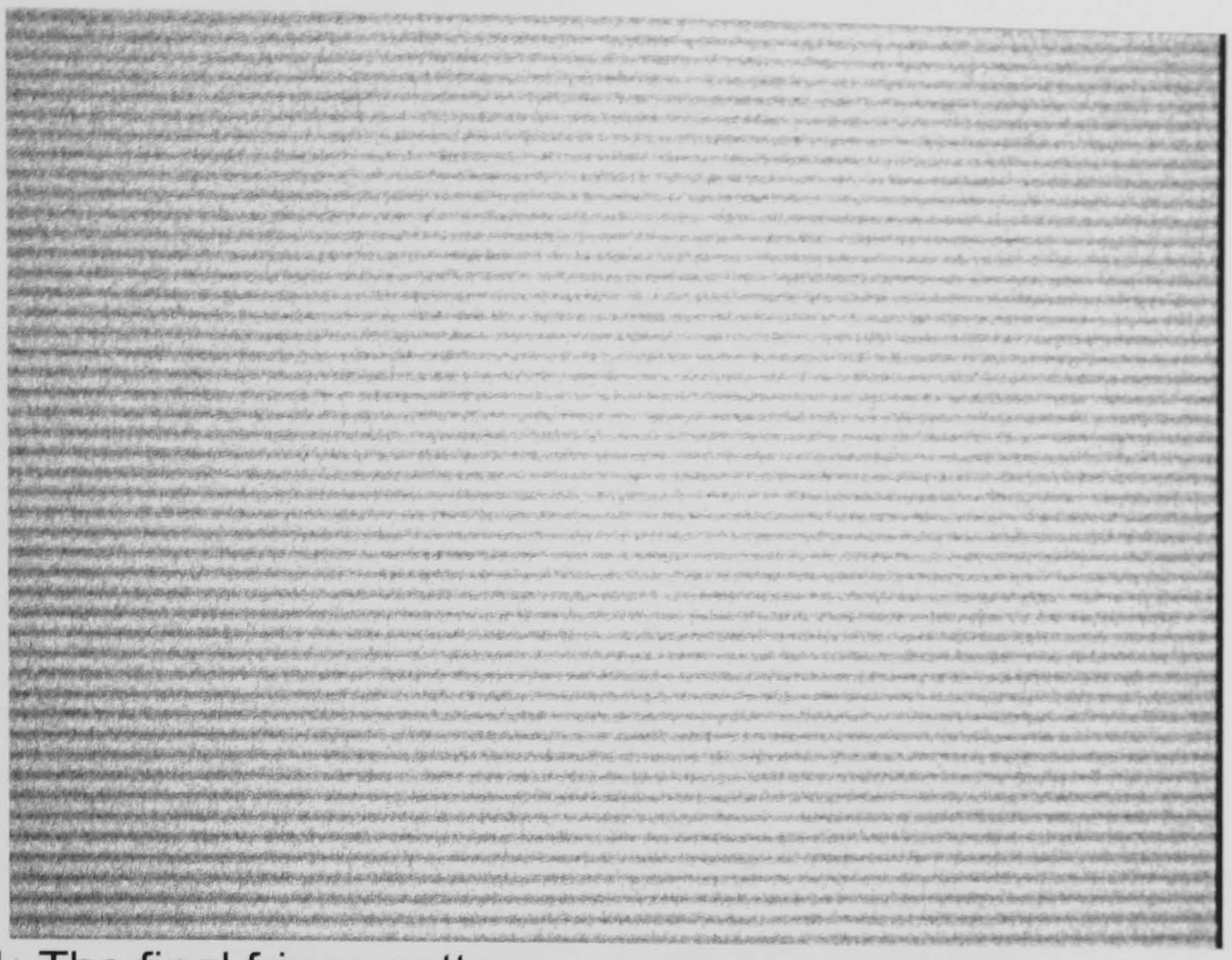


Figure 5.31: The final fringe pattern.

Figure 5.32 shows a more complex image. Here, more than one parameter deviated from the ideal. It can be seen that not only are there too many fringes, but the mean intensity is too low to enable an accurate measurement to be executed. The first reaction of the system is to try to correct the intensity problem. The output from the neural network is shown in figure 5.33. At this stage, the lenses used with the system were not fully automatic, so the aperture had to be corrected by the operator. When this had been done, the system was able to continue with its optimisation procedure. The second image presented to the network was as shown in figure 5.34. Here, although the mean intensity was now correct, the problem of too many fringes was still present. The system then behaved as previously described to produce an optimised fringe pattern. Figure 5.35 shows the resultant fringe pattern after 9 iterations. The final result is a fringe pattern that conforms to the previously defined optimisation criteria.

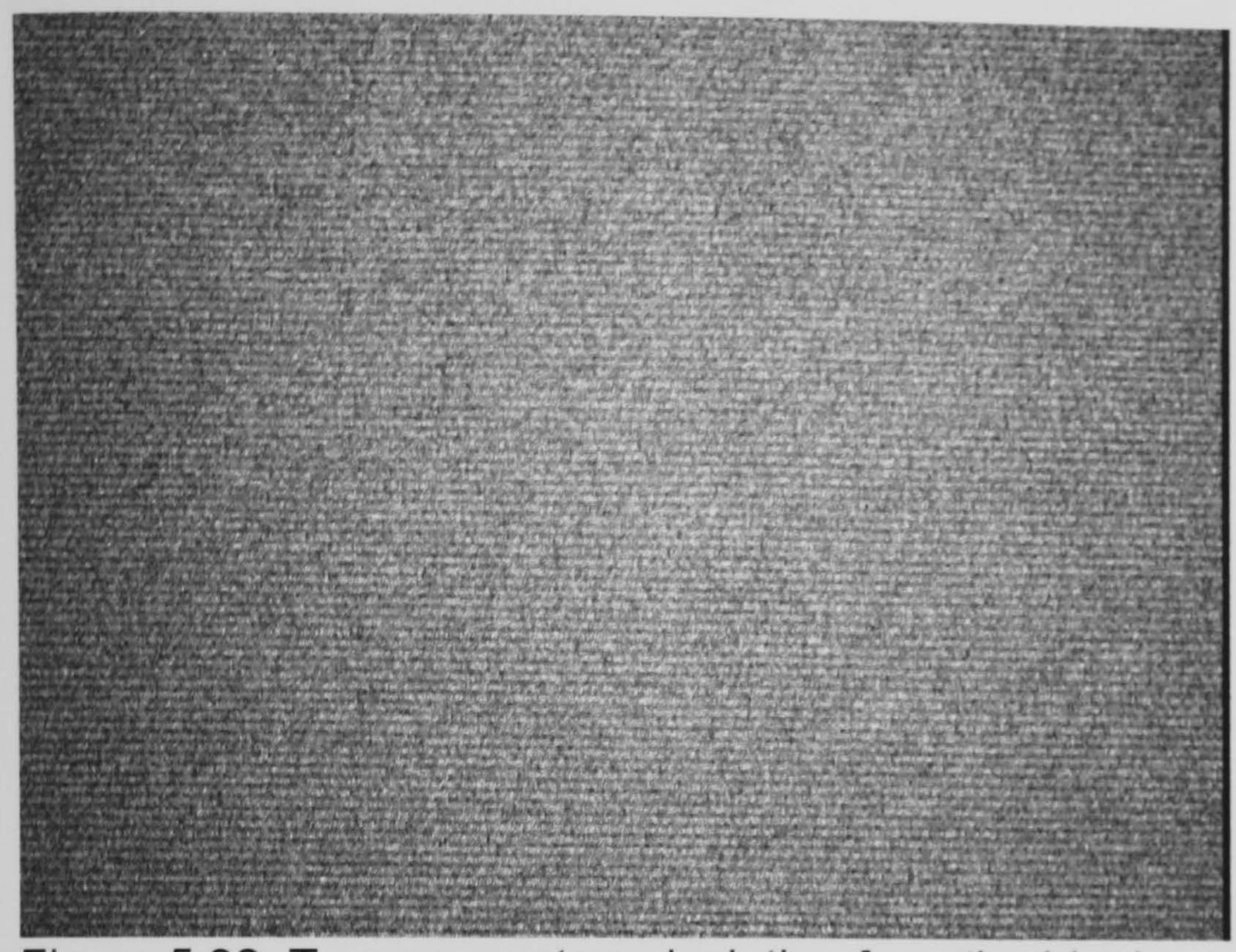


Figure 5.32: Two parameters deviating from the ideal.

```
Backprop optimisation system version 4.30.
Current MotionMaster error state is: E00 NOERROR
** Loop iteration number 1 **
Snapping current image…

Mean values are:       0.411000 0.122000 0.123000 0.136000 0.145000
Fringe counts are:     0.310000 0.300000 0.360000 0.430000 0.430000
Visibility values are: 0.463415 0.460526 0.458599 0.413333 0.445483
Tilt values are:       0.550000 0.500000 0.430000 0.480000 0.470000

Network outputs are:   0.282032 0.826404 0.367829 0.513631

Mean intensity is too low.
There are too many fringes in this image
Visibility is OK
Fringes are level

Adjusting fibres…
Current MotionMaster error state is: E00 NOERROR
```

Figure 5.33: Network outputs for image shown in figure 5.32.

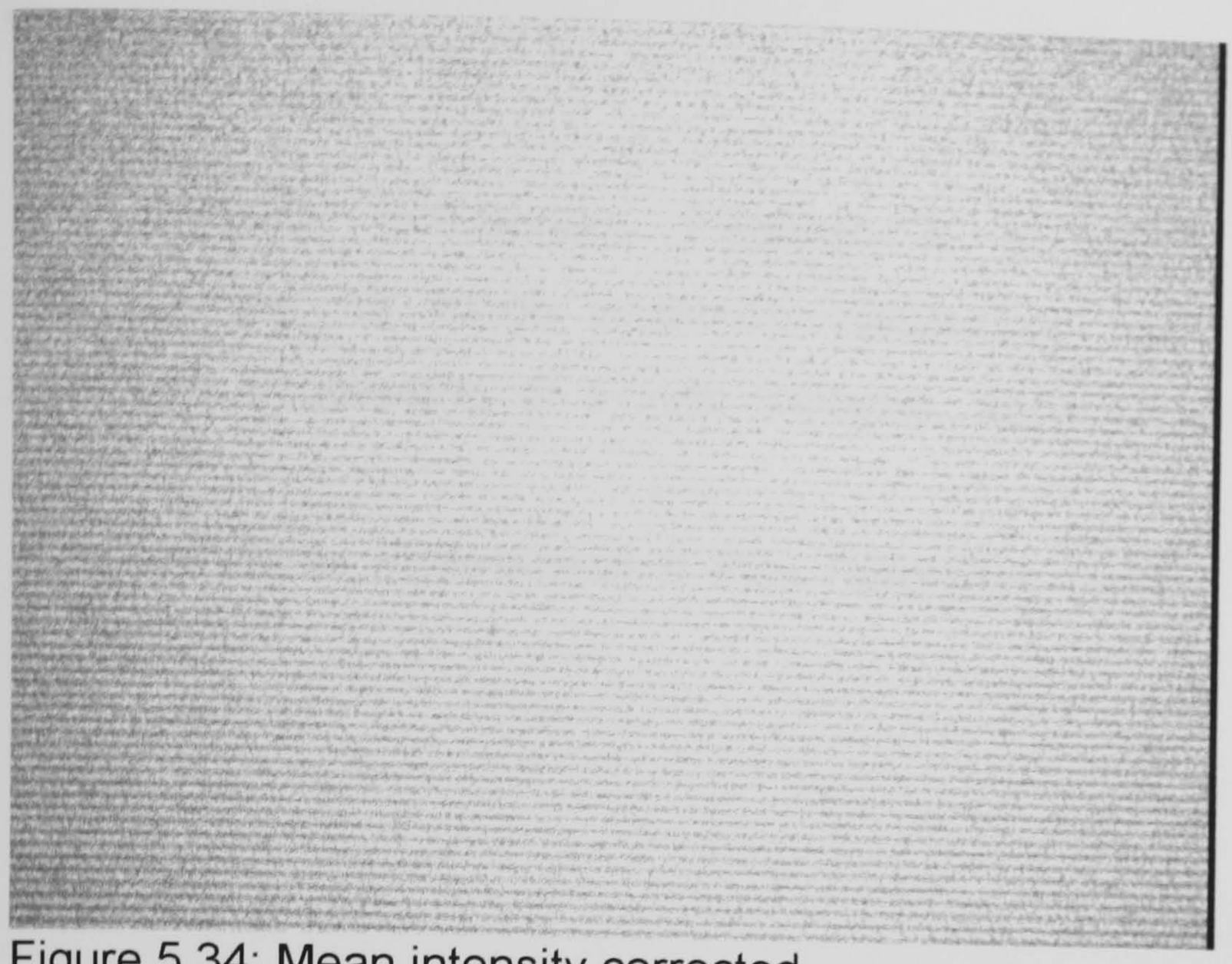


Figure 5.34: Mean intensity corrected.

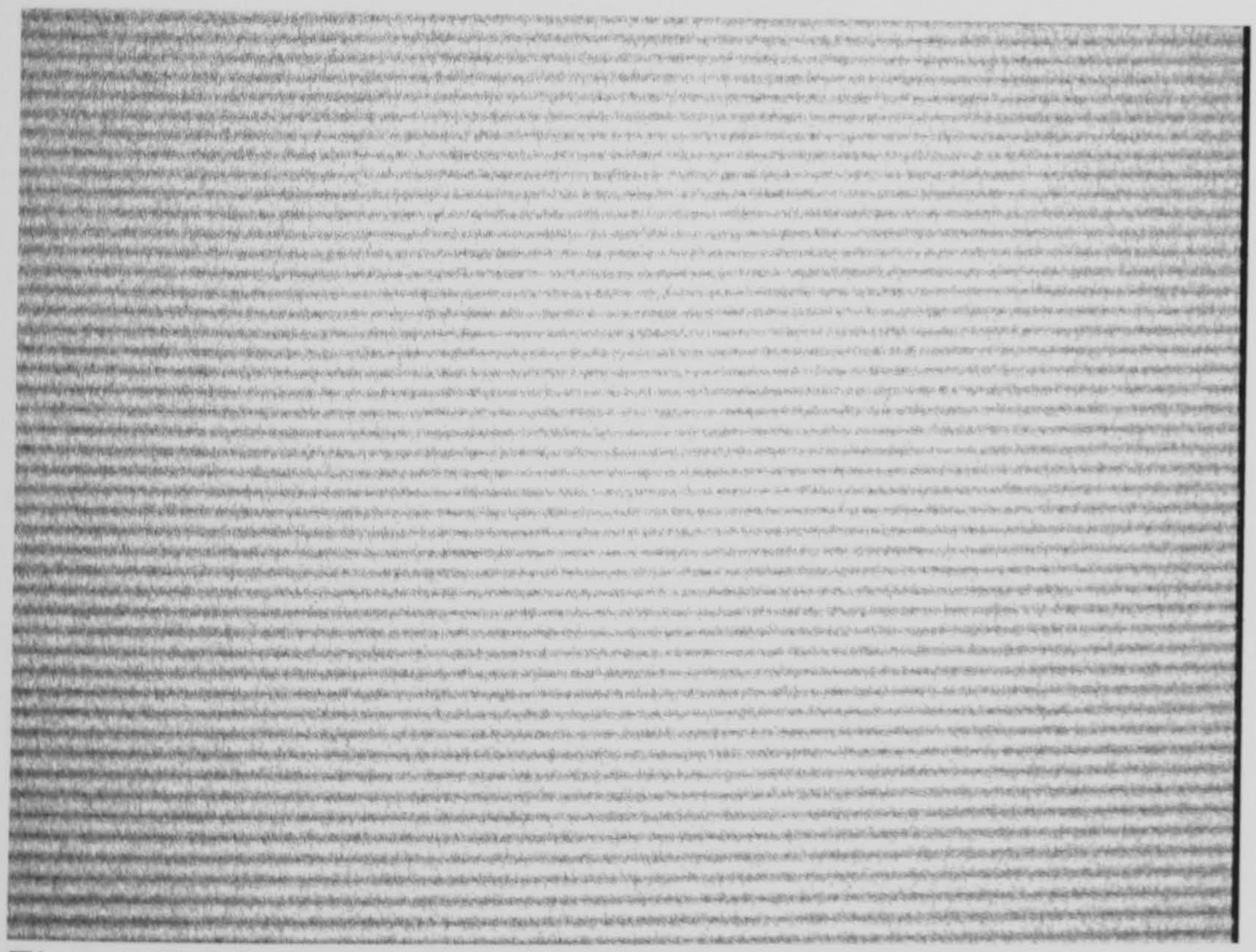


Figure 5.35: Final optimised image

References

1.J.T.Atkinson, D.R.Burton, P.Barton & M.J.Lalor. A fibre optic interferometer for fringe projection contouring. 2nd Int. Workshop on Automatic Processing of Fringe Patterns, pp242-248, Akedemie Verlag, 1993.

2. NeuralWorks Professional II Plus Reference Guide. NeuralWare, Inc.,

Chapter 6: Discussion, conclusion and further work

Chapter 6: Discussion

This thesis has described novel approaches to two problems in fringe analysis. The use of interference fringes as a measurement tool is widely recognised and has developed over recent decades into a well established science. All the stages of the process, from image capture to height calculation, have been extensively researched and many solutions to the problems inherent in the system have been proposed. Also, almost parallel with regards to its timescale, the neural network has been developed as an alternative computing tool. Over the past twenty years, neural computing has matured as a science and its applications are increasing. It is noteworthy that, to date, little research has been carried out to link these two branches of science. One field in which the neural network excels over conventional computing is that of pattern recognition. Neural networks are particularly suited to this type of problem due to their inherently parallel architecture and the nature of the problem. The work presented in this thesis has attempted to marry the two fields to produce a parallel approach to some of the more difficult problems in fringe analysis, namely phase unwrapping and fringe pattern optimisation.

6.1 Phase unwrapping

Chapter 4 has dealt with the phase unwrapping problem. With few exceptions, the general approach to this particular problem has been algorithmically based. The philosophy behind the work presented here was to treat the detection of phase discontinuities as a variation on the theme of pattern recognition and introduce a method to solve the problem with a parallel computing architecture. Initial experimentation attempted to solve a much simplified version of the problem in order to investigate whether a neural network was capable of addressing the problem with any degree of success. The first findings of this methodology showed that the approach appeared to be flawed in its capability of detecting multiple phase

discontinuities in a given section of phase data. It is likely that in adopting this method, one of two problems was present. The network was either attempting to deal with too much data to reach an adequate conclusion, or the network itself was simply too complex.

If too many data were present, it is likely that the network was being given too many alternative answers to process. Two states are common in neural computing when network errors fail to converge during training. If a large and complex network is presented with a small number of training data, training will be incomplete as the network will not have enough data from which to gain experience. Unless the situation arises where the network contains an extremely large number of processing elements and the training sets are very small, this state rarely occurs. More likely is the problem of over-training. This is a phenomenon that can be attributed to several factors and can be common with an inexperienced user. If the network is presented with a large number of training data and, more importantly, if the data is presented in a highly ordered manner, the network will over-train. When this happens, some or all of the processing elements will "lock". Here, each element will give a single result regardless of the input that is applied to it. Over-training also gives rise to a situation in which the network learns to recognise only one particular training set. When this occurs, presentation of data that differ even slightly from the original training data will give highly erroneous results. It is for these reasons that learning rules such as the cumulative-delta and normalised-cumulative delta were developed. These learning rules attempt to address the problem by ensuring that data are always presented randomly to the network. However, using these rules in the initial experiment to ensure that the training data were applied in a completely random manner did not improve the observed results.

If a network contains a large number of processing elements, this can also hinder effective training. If too many hidden neurons are present in the network, they become under used and it is possible to "lose" information.

The decision to abandon the use of the 64 input network was taken as it was thought that this approach was far too complex for the early stage of the experimentation. The initial idea had simply been to prove that a neural network could be used to differentiate between phase wraps and discontinuities caused by noise, etc., rather than to produce a fully functional unwrapping system. Use of smaller networks showed far more promising results. Here, the networks were only called upon to recognise a single phase wrap in a given portion of a wrapped phase distribution. This immediately helped to provide a solution by drastically reducing the complexity of the problem. It also meant that the complexity of the training data could be reduced, thus reducing training time. The simplification immediately showed an improvement in results. With the reduction in computational complexity, the network was able to recognise a single phase wrap in a given section of phase data. It is interesting to note the difference in results given for different types of training data. If the results files for networks trained with real and simulated phase data are compared, the networks to which data from real phase distributions has been applied show a higher percentage accuracy. It was this discovery that forced the decision to use data from real images for training the networks that were to make up the final operational system. It may be argued here that training with "real" rather than "simulated" data allows the network to have experience of noise contained within the phase distribution. It will be able to learn when noise is present in order to reject it. If noise free training data are presented, it is possible that the network will become confused when presented with noisy, "real world" data.

The convolution of the single wrap detector with a complete one-dimensional array of wrapped phase data showed the network's effectiveness. The comparison with Schafer's algorithm in chapter 4 is indicative of this ability. If the results are studied closely, it can be seen that, when a relatively clean wrapped phase distribution is used, both methods achieve a similar result. This is a very simplistic application and most wrapped phase distributions that are created during the fringe analysis process contain a significant

amount of noise. It is when spot noise spikes are present in the wrapped phase distribution that the neural network method excels, as subsequent results show. When a single spot noise spike is encountered by Schafer's algorithm, the algorithm immediately interprets the noise as an extra phase wrap. The result of this is that an extra 2π phase shift is introduced and that this error is propagated throughout the remaining portion of the phase distribution. The network, however, treats the noise spike as an error and effectively "ignores" it, ensuring that error propagation does not occur. Further proof of the method's effectiveness is shown when a number of spot noise spikes are present in the wrapped phase distribution. Again, Schafer's algorithm sees the noise as additional phase wraps and compensates accordingly. The result of this is an accumulation of errors, which are propagated throughout the final phase distribution. The neural network method, however, treats the multiple noise spikes as errors and again ignores them, leading to a correctly unwrapped final result. The results have proved that the simple network can adequately differentiate between spikes that were present due to noise and genuine phase wraps. The embedding of the neural network in code that performed the unwrapping operations showed that with the network used to detect the wraps and conventional code used to perform the necessary 2π phase shifts, this arrangement could form the basis of a possible solution to the unwrapping problem. The convolution process also showed that detection of discontinuities could be hampered when the phase wrap reaches the end of the wrap detector "window". Careful construction of the actual unwrapping code can, however, compensate for this anomaly.

It may appear at this stage in the experimentation, that the approach is over-simplistic and highly unrepresentative of the "real-world" situation where the phase unwrapping process is concerned. When a fringe pattern is used to measure a surface, the resultant image is invariably a two dimensional array of data. Typically, these arrays contain around 250,000 values that include varying amounts of noise and phase wraps, which do not follow easily

detectable straight lines. A "real-world" wrapped phase distribution will contain clean wraps, spot noise and features such as wrap bifurcations etc.. It is highly unlikely that one will encounter a perfectly noise-free phase distribution. At this point, it must be borne in mind that the one-dimensional wrap detector described above is not an attempt to create a fully functional two dimensional phase unwrapping system. The initial experimentation was carried out only to prove whether a backpropagation neural network was capable of detecting wraps in a phase distribution and recognising the difference between genuine phase wraps and noise spikes. It is evident from the results presented in this thesis that, at this level, a backpropagation network is ideally suited to the task. Its parallel processing architecture, well suited to recognition problems, appears to make it possible for the network to recognise the difference between the two instances.

It is this success in recognising the difference between phase wraps and noise that has led to the development of a two-dimensional neural network assisted phase unwrapping system. The second stage of experimentation showed how a square "tile" of pixels could be used to detect the presence of phase wraps in a two dimensional wrapped phase distribution. The key point to address here was to develop a 2-D unwrapper with an optimum tile size. The need for this was shown by reference to the original work of both Schafer and Takeda that was described in chapter 4. It was desired to produce a system that could detect the presence of phase wraps in an image without the computational complexity associated with previous solutions to the problem. The two extremes can be described thus: the largest wrapped phase distribution that is likely to be encountered is a 512x512-pixel image. If one were to configure a network to analyse the entire image in a single pass, the least number of input neurons required would be 512x512, that is, 262,144 neurons. Considering that the system would require at least the same number of output neurons, the number of neurons would then increase to 524,288. This figure still does not take into account any hidden neurons that will be required for correct functionality. The training and testing data sets will also be suitably large. Consider a training set for a network

conforming to the supervised learning model. One training vector will require 262,144 input values and an equal number of output values. If, say, 100 training vectors are used, the training set will contain 52,428,800 values. With networks and training sets of this size, training times will be correspondingly long. Also, problems may be encountered as described earlier, which involve training the network to recognise multiple wraps in a single presentation. It would appear sensible procedure to attempt to reduce the size and complexity of the networks to conserve both time and computing power. At the other end of the scale, if the network is made too small, using only, say, 2 input neurons, such a network would run the risk of behaving as an extremely computationally expensive version of Schafer's original point-to-point algorithm. The first "small" networks investigated were an attempt to reduce the regional analysis to the smallest possible area without encountering the previously described problem. It was this criterion that gave rise to the 3x3 tile wrap detector.

Extending the system into two dimensions was a logical step to take to address the complete phase unwrapping problem. Results of the two dimensional unwrapping experiments showed that the networks were capable of detecting the presence of phase discontinuities over a defined region with similar accuracy to that produced by the single line unwrappers. To complete the system, it was necessary to convolve the nine pixel "window" with a complete phase distribution to provide a final unwrapped phase distribution. The results shown in chapter 4 demonstrate the effectiveness of this operation. The completed system demonstrated its ability to differentiate between phase discontinuities and noise in a 512x512 array of wrapped phase data.

Fringe Optimisation

The problem of fringe pattern optimisation is a relatively new science. It was not until the development of the twin-fibre adaptive interferometer described in chapter 2 that successful automation of this process became possible. Until the interferometer was developed, the adjustment of fringe patterns was an extremely labour intensive task that was not suited to measurement and inspection of surface profiles in situations where time was critical. For example, in measurement of components with various surface profiles on line, the adjustment of a fringe pattern was impossible in the time allotted using a conventional interferometer or fringe projection system. Also, the use of a human operator to determine the quality of a fringe pattern is questionable. The decision is likely to vary from operator to operator, plus the decisions made by the same operator under different conditions are likely to be unreliable. Although human operators are much faster than digital computers at recognition tasks, their concept of quality is highly variable. It is mainly due to this inability to perform on-line optimisation that there has been no previous research into the subject. The use of a neural network to perform the qualitative analysis of fringe patterns arose as a compromise between these two situations. The ideal situation is one in which the system making decisions on quality has the parallel processing speed to match the human brain's recognition ability but also has the reliability and repeatability of a digital computer. The experiments described in chapter 5 of this thesis have attempted to show that a system based on this philosophy is possible to implement. The parameters chosen for analysis were chosen on the basis that they could easily be both measured and changed. Fringe number and fringe orientation were used, as these were the parameters that could be controlled by the adaptive interferometer. Fringe contrast and average intensity were used as these could be easily controlled using the lens attached to the CCD camera. For the purpose of the experiments, the lens used was a conventional 55mm Nikon item with manual focusing and aperture setting. This obviously meant that any

changes that were required to intensity and visibility and, therefore, must be made by changing lens settings, had to be made manually, by the operator. To fully automate the system, a lens featuring autofocus and automatic aperture setting would be desirable. Even though an operator was required at this point, it must be stressed that this did not detract form the original specification, as the final decision on fringe quality was ultimately made by the neural network system.

The two approaches to the problem were variations on the same basic theme, but relied on slightly different network configurations. The initial experimentation involving a separate network to analyse each parameter was carried out to prove that the theory was viable. The results given by this experimentation showed that it appeared to be a viable proposal to follow this route to achieve a fringe pattern optimisation system. Consider the results achieved by the first fringe number network. Simply counting the number of peaks and troughs is not an adequate method of determining the number of fringes present in the image. Calculating the mean value, subtracting this from the intensity distributions and counting the number of times the resultant distribution crosses the zero point is a far more reliable method. Using this method, it is possible to compensate for any false maxima and minima that occur at the fringe peaks due to noise in the image. The use of this piece of mathematics at the initial data acquisition stage helps to simplify the later analysis. This simplification is important when any neural network is used for analysis. The simpler the input data for the network, the easier the network is to train. Experimentation has shown that training is most successful when both input and output values lie in the region of $0 \leq x \leq 1$. The two approaches can be appraised as follows. The use of separate networks for each parameter was chosen primarily to test the feasibility of the idea. The four parameters were chosen due to their ease of calculation and the fact that they can easily be changed to meet the optimisation criteria. The parameters were:

1. Fringe number. This can be calculated by a simple fringe count and can automatically be controlled by the adaptive interferometer. Adjusting the spacing of the fibre ends changes the number of fringes present in the image.

2. Fringe orientation. This can be calculated by ascertaining the number of fringes at various orientations throughout the image. Again changing the fibre spacing within the interferometer will change the orientation of the fringes.

3. Mean intensity. This can be calculated by taking an average of the intensity values in a given intensity profile. It can be controlled by varying the camera aperture.

4. Fringe contrast. This is calculated by the equation

$$C = (I_{max} - I_{min}) / (I_{max} + I_{min})$$

It can be taken as an indication of the focus of the image.

The use of separate networks was originally designed to enable the user to re-train a network for one parameter without affecting the outputs of the other parameters. Chapter 5 shows the results given by this approach. The networks were capable of recognising the quality of the given patterns according to the defined parameters. However, the outputs required further post-processing to enable the system to adjust the fringe pattern to produce the required result. The second stage of optimisation experiments attempted to reduce the need for excessive post-processing of the data. The approach, using a single network to analyse all the parameters simultaneously was employed and showed results comparable with the previous, separate network experiments. Re-training the network for a single parameter did not have a great adverse effect on the outputs of the

other parameters and added to the overall performance of the optimisation system.

It was from these results that the final optimisation system was developed. Combination of the neural network outputs and conventional code enabled an operational system to be constructed. While this can be thought of as a "hybrid" system, containing both neural network and conventional code, it is noteworthy that all the decisions required of the system are made by neural network. The conventional code is used simply to read images from a frame-grabber and ultimately to adjust the position of the fibres within the interferometer.

Conclusions

The work described within this thesis has shown that it is possible to apply neural networks to the solution of problems in fringe analysis. The two major problems addressed, phase unwrapping and fringe optimisation, have been thoroughly investigated and operational systems to attempt to solve these problems have been developed.

Phase unwrapping continues to be one of the most difficult tasks inherent in the fringe analysis process. The system described utilises a backpropagation neural network to distinguish phase wraps from noise present in a wrapped phase distribution. Small neural networks were used for the system to preserve its simplicity. Use of large networks containing thousands of neurons to analyse large areas or whole images were thought to be impractical due to the number of data required for training and testing. Using conventional code to perform the 2π phase adjustments ensured that the networks were only required to ascertain the position of the phase wraps in the distribution. The result was a highly robust phase unwrapping system. As with all phase unwrapping algorithms, it will not be efficient in 100% of cases. Certain wrapped phase distributions, by virtue of excessive noise

etc., will be extremely difficult to unwrap, whichever method is used. Although the neural network based system is extremely efficient at differentiating noise from genuine phase wraps, a "generic" phase unwrapper has yet to be designed.

The fringe optimisation system addresses a problem fairly new to the science of fringe analysis. The work described in this thesis has shown that it is possible to produce a "closed" loop system that optimises the quality of a fringe pattern before analysis takes place. The work has been carried out using the minimum number of parameters required to calculate fringe pattern quality, but it can be concluded that a neural network can be employed to satisfactorily analyses these parameters. The system has been designed as a "hybrid" system to ensure that analysis can be carried out adequately. The data is both pre- and post-processed using conventional code. Pre-processing is carried out to ensure that the networks can easily understand the data presented to them. It was found that the networks operated most satisfactorily when all the input data were in the same range. Also, a degree of post-processing is required to enable the fringe pattern to be fully optimised. The translation stages used to carry the optical fibres within the interferometer are operated by a controller which relies on "C" code to send the necessary instructions. It was these considerations which led to the development of a hybrid system for optimisation. Combining conventional code and neural networks in this way made it possible to "close the loop" in the fringe analysis process.

Further work

It must be stated that the aim of this work was merely to ascertain whether neural networks are a viable tool for addressing the stated problems in fringe analysis. In neither case has a complete solution to those problems been offered. Rather, the work has proved that neural networks can form a basis from which reliable solutions can be obtained. This has been successfully shown with the use of backpropagation and general regression networks as a basis for solutions to the problems of phase unwrapping and fringe pattern optimisation. Only small networks have been employed throughout the project in order to keep training data manageable and training times low. The nine-neuron phase unwrapping tile was used as this was the smallest possible tile that could be used without returning to a variation on a point-to-point algorithm. The reasoning behind this approach was partly due to the hardware and software capabilities of the computing systems economically viable at the beginning of the project. It has already been stated that the work described herein was carried out using a machine containing a 486 processor with a clock speed of 66MHz, 16MB RAM and DOS-based neural network simulation software. Training becomes difficult with such a system when a large number of neurons are used and large training sets become unmanageable. Although this does not directly rule out using larger tiles than 3x3, it certainly will make training networks for tiles greater than 10x10 extremely difficult. Future investigation into an optimum tile size for regional phase unwrapping may be possible. Considering the pace at which development takes place in the computing world, the constraints applied at the beginning of the project are no longer a problem. The use of a machine containing a Pentium III processor with 600MHz clock speed and 128MB RAM would make further experimentation much easier. With a significant (almost ten-fold) increase in processing speed, training times will be drastically reduced. This will, of course, negate some of the concerns voiced early in the project. It must also be borne in mind the systems used here are all single-processor systems. Advances in hardware technology

now also mean that multi-processor systems using genuine hardware-based parallel processing are becoming more widely available. More advanced versions of neural network simulation software also need to be investigated.

Although small regions of an image have been analysed for the reasons already stated, larger regions are likely to provide more error-free results. Further investigation needs to be carried out into increasing the size of the tile model to produce a regional analysis system similar to those described in chapter four, but utilising a more parallel approach. Similarly, it may be possible to investigate the use of a neural network based approach to techniques such as the data validation procedure described in chapter four. Here, edges of regions could be processed to phase wrap continuity and assigning regions to be processed before a much simplified unwrap takes place.

Many algorithms already proposed to deal with the unwrapping problem may be compatible with a parallel approach. Although completely re-working a proven solution may be unnecessary and impractical, the application of neural network techniques to regional or temporal methods may be feasible.

A number of neural network architectures were investigated for purposes of solving the phase unwrapping problem. However, backpropagation networks proved to be the most suitable for the task. This has shown that it is possible to achieve a solution using systems not adherent to formal rules or logic. An extension of this finding may suggest that investigation be carried out into methods of a similar nature. While a number of alternative supervised learning networks are available, it may be feasible to look into the benefit using alternative systems. Certain alternative neural network architectures exist, such as self-organising networks, Fuzzy ART (Adaptive Resonance Theory), etc. Non-neural network based computing paradigms not reliant on absolute logic also exist. One such paradigm such as this is Fuzzy Logic where, instead of simply using binary criteria, a series of intermediate states, depending on system input, are also used. The benefit

of Genetic Algorithms may also be investigated. Another alternative paradigm, the GA begins with an algorithm that may provide a solution to a problem which evolves over to produce an optimum solution.

Many of the arguments outlined above can also be applied to the second problem, namely that of fringe pattern optimisation. Alongside the criteria of reducing the size of the training sets and training time for the optimisation networks, both of which can be addressed as previously described, the decisions regarding optimisation parameters were based on those most readily measurable and their ability to be adjusted automatically. At this stage of the project, the adjustment of both lens aperture and focus, which relate to mean image intensity and fringe visibility, are still carried out by the operator. A very simple improvement in the system would be to substitute the lens with one that can be operated fully automatically. This would mean that the current system would then become fully "closed loop", meaning that all adjustment to the necessary fringe pattern parameters could be performed by the computer.

Further work could build alternative methods of pre-processing optimisation parameters. As already stated, experimentation has been carried out on parameters that are easily adjustable and directly measurable from an interference fringe pattern. More parameters can be investigated, such as fringe spacing, bandwidth and noise level. Fringe spacing and bandwidth (or minimum and maximum spacing) can be calculated in a similar way to how the fringe number was calculated as described in chapter 5. Noise level is a much more involved calculation. The systematic analysis of noise in an image has already been described in chapter 4, the calculations being related to the level of noise at which the neural network phase unwrapping system is efficient. Noise in a fringe pattern can be calculated from that fringe pattern's Fourier transform, however, the original specification of the optimisation system was to ascertain the image's suitability for measurement before the process takes place. As the Fourier transform is an integral part of the measurement process, this method of calculation introduces further

computational complexity into the process. As described above, specifications of computer hardware have significantly improved since the beginning of the project, making the introduction of extra computation less important. Also, with the improvement of parallel hardware, it may also be possible to reduce computation time further by use of such systems. In both areas of research covered in this thesis, it would appear that the use of parallel hardware may be the next stage of investigation. Both the phase unwrapping and fringe pattern optimisation systems described used a software based neural network simulation package to achieve a solution. Now that the idea of neural networks applied to these problems has been proved to be a viable solution, investigation of hardware based parallel processing systems would be a logical following step. Software analysis may still be carried forward using alternative paradigms such as fuzzy logic or genetic algorithms.

The work in this thesis has proved that neural networks are able to provide a viable solution to problems within the field of Fourier fringe analysis. To achieve a measurement system, it is necessary to design a specification for a complete measurement tool. A hybrid system that combines some of the elements of conventional Fourier fringe analysis with the neural network techniques already described above may be a suitable approach to the problem. The final solution should include an adaptive interferometer to project a fringe pattern on the surface to be measured, a neural network based optimisation system to analyse the resultant fringe pattern and feed the results of this operation back to the adaptive interferometer, conventional Fourier fringe analysis software to perform the phase calculations required for a measurement and a neural network based phase unwrapper to complete the process. While this is the most desirable situation at present. future work may also include the use of neural networks to assist in the main body of the measurement software.

Having proved the suitability of neural networks to assist in the Fourier fringe analysis process it is considered that further investigation of this paradigm is extremely feasible.

Appendix 1

Phase Unwrapping Training and test data

1. Training file for 6-input wrap detector network - simulated data

The following file uses simulated data for training a 6-input line unwrapping network. The first six values of each vector are those presented to the input layer of the network, whereas the last six are the desired output values presented to the output layer.

```
! Training file for edge_6.nna using real data from
! SIMULATED FRINGES.
3.141593 2.932769 2.722865 2.510879 2.295950 2.077394 0 0 0 0 0 0
1.854724 1.627636 1.395989 1.159768 0.919060 0.674012 0 0 0 0 0 0
0.424816 0.171687 -0.085152 -0.345478 -0.609068 -0.875706 0 0 0 0 0 0
-1.145179 -1.417276 -1.691783 -1.968480 -2.247139 -2.527518 0 0 0 0 0 0
-2.809359 -3.092389 2.906866 2.622335 2.337511 2.052702 0 0 1 0 0 0
1.768203 1.484292 1.201220 0.919197 0.638390 0.358916 0 0 0 0 0 0
0.080841 -0.195821 -0.471105 -0.745085 -1.017874 -1.289613 0 0 0 0 0 0
-1.560460 -1.830589 -2.100177 -2.369397 -2.638419 -2.907402 0 0 0 0 0 0
-2.907402 3.106697 2.837378 2.567713 2.297608 2.026985 0 1 0 0 0 0
1.755783 1.483956 1.211474 0.938323 0.664502 0.390023 0 0 0 0 0 0
0.114913 -0.160789 -0.437033 -0.750426 -0.990889 -1.268349 0 0 0 0 0 0
-1.546047 -1.823892 -2.101789 -2.379642 -2.657361 -2.934859 0 0 0 0 0 0
-2.379642 -2.657361 -2.934859 3.071124 2.794281 2.517851 0 0 0 1 0 0
2.241868 1.966353 1.691311 1.416733 1.142595 0.868864 0 0 0 0 0 0
0.595495 0.322438 0.049638 -0.222964 -0.495426 -0.767806 0 0 0 0 0 0
-1.040160 -1.312541 -1.584994 -1.857564 -2.130289 -2.403199 0 0 0 0 0 0
-2.676323 -2.949683 3.059893 2.786022 2.511886 2.237485 0 0 1 0 0 0
1.962825 1.687917 1.412777 1.137427 0.861891 0.586198 0 0 0 0 0 0
0.310380 0.034473 -0.241488 -0.517465 -0.793422 -1.069325 0 0 0 0 0 0
-1.345140 -1.620839 -1.896397 -2.171793 -2.447012 -2.722046 0 0 0 0 0 0
-2.996886 3.011651 2.737190 2.462910 2.188799 1.914843 0 1 0 0 0 0
1.641027 1.367332 1.093741 0.820232 0.546786 0.273383 0 0 0 0 0 0
0.000000 -0.273383 -0.546786 -0.820232 -1.093740 -1.367332 0 0 0 0 0 0
-1.641027 -1.914843 -2.188799 -2.462910 -2.737190 -3.011651 0 0 0 0 0 0
-1.914843 -2.188799 -2.462910 -2.737190 -3.011651 2.996886 0 0 0 0 0 1
2.722046 2.447012 2.171793 1.896397 1.620839 1.345140 0 0 0 0 0 0
1.069325 0.793422 0.517465 0.241488 -0.034473 -0.310380 0 0 0 0 0 0
-0.586198 -0.861891 -1.137427 -1.412777 -1.687917 -1.962825 0 0 0 0 0 0
-2.237485 -2.511886 -2.786022 -3.059893 2.949683 2.676324 0 0 0 0 1 0
2.403199 2.130288 1.857564 1.584994 1.312541 1.040160 0 0 0 0 0 0
0.767807 0.495426 0.222964 -0.049638 -0.322438 -0.595495 0 0 0 0 0 0
-0.868864 -1.142595 -1.416733 -1.691311 -1.966353 -2.241868 0 0 0 0 0 0
-2.517851 -2.794281 -3.071124 2.934859 2.657361 2.379642 0 0 0 1 0 0
2.101789 1.823892 1.546047 1.268349 0.990889 0.713758 0 0 0 0 0 0
0.437033 0.160789 -0.114913 -0.390023 -0.664502 -0.938323 0 0 0 0 0 0
-1.211474 -1.483956 -1.755783 -2.026985 -2.297608 -2.567713 0 0 0 0 0 0
-2.837378 -3.106697 2.907402 2.638419 2.369397 2.100177 0 0 1 0 0 0
1.830589 1.560460 1.289613 1.017874 0.745085 0.471105 0 0 0 0 0 0
0.195821 -0.080841 -0.358916 -0.638390 -0.919197 -1.201220 0 0 0 0 0 0
```

2. Training file for 6-input wrap detector network - real data

The following values are as described in appendix 1.1, but gathered from data taken from real wrapped phase distributions.

```
!Training data for 6-input wrap detector
!Compiled from REAL DATA

3.1416 -2.8274 -2.5130 -2.1991 -1.8849 -1.5708 0 1 0 0 0 0
2.8274 3.1416 -2.8274 -2.5130 -2.1991 -1.8849 0 0 1 0 0 0
2.5130 2.8274 3.1416 -2.8274 -2.5130 -2.1991 0 0 0 1 0 0
2.1991 2.5130 2.8274 3.1416 -2.8274 -2.5130 0 0 0 0 1 0
1.8849 2.1991 2.5130 2.8274 3.1416 -2.8274 0 0 0 0 0 1
1.5708 1.8849 2.1991 2.5130 2.8274 3.1416 0 0 0 0 0 0
1.2256 1.5708 1.8849 2.1991 2.5130 2.8274 0 0 0 0 0 0
3.1416 -2.7489 -2.3562 -1.9635 -1.5708 -1.1781 0 1 0 0 0 0
2.7489 3.1416 -2.7489 -2.3562 -1.9635 -1.5708 0 0 1 0 0 0
2.3562 2.7489 3.1416 -2.7489 -2.3562 -1.9635 0 0 0 1 0 0
1.9635 2.3562 2.7482 3.1416 -2.7489 -2.3562 0 0 0 0 1 0
1.5708 1.9635 2.3562 2.7489 3.1416 -2.7489 0 0 0 0 0 1
1.1781 1.5708 1.9635 2.3562 2.7489 3.1416 0 0 0 0 0 0
0.7854 1.1781 1.5708 1.9635 2.3562 2.7489 0 0 0 0 0 0
-8.3723 -7.5633 -6.6345 -5.4542 -4.4520 -3.0987 0 0 0 0 0 0
8.0164 -8.8100 -7.4566 -6.0909 -5.0246 -4.3358 0 1 0 0 0 0
7.5243 8.3981 -8.3412 -7.8906 -6.1415 -5.0742 0 0 1 0 0 0
6.9022 7.1000 8.6110 -8.7063 -7.5500 -6.4306 0 0 0 1 0 0
5.0305 6.3101 7.0925 8.1111 -8.6516 -7.0001 0 0 0 0 1 0
4.0097 5.3399 6.1701 7.1134 8.2205 -8.6673 0 0 0 0 0 1
3.0987 4.7634 5.2367 6.6721 7.1762 8.0045 0 0 0 0 0 0
-12.9222 -10.7000 -8.2468 -6.0704 -4.2928 -2.9205 0 0 0 0 0 0
12.6505 -12.0456 -10.6453 -8.3456 -6.5634 -4.5623 0 1 0 0 0 0
10.4638 12.3924 -12.3041 -10.1111 -8.7777 -6.4532 0 0 1 0 0 0
8.0986 10.6534 12.3625 -12.0121 -10.5553 -8.6664 0 0 0 1 0 0
6.5876 8.7106 10.5269 12.1710 -12.1957 -10.1822 0 0 0 0 1 0
4.1969 6.6100 8.1007 10.1212 12.6520 -12.1069 0 0 0 0 0 1
2.8888 4.8511 6.1098 8.2495 10.1597 12.0948 0 0 0 0 0 0
-50.6734 -45.6914 -40.9163 -35.3945 -30.0153 -25.8421 0 0 0 0 0 0
50.1842 -50.3701 -45.4801 -40.0348 -35.2905 -30.9191 0 1 0 0 0 0
45.4678 50.3460 -50.9987 -45.6457 -40.6345 -35.2134 0 0 1 0 0 0
40.6450 45.2343 50.6534 -50.6354 -45.3645 -40.3450 0 0 0 1 0 0
35.9995 40.0202 45.8516 50.6764 -50.0519 -45.2052 0 0 0 0 1 0
30.2052 35.4343 40.9086 45.4611 50.6123 -50.9153 0 0 0 0 0 1
25.6506 30.7101 35.0933 40.6640 45.7007 50.7170 0 0 0 0 0 0
-18.6459 -15.3452 -12.8967 -9.2397 -6.4793 -3.5910 0 0 0 0 0 0
18.1969 -18.1958 -15.1939 -12.1966 -9.1985 -6.1944 0 1 0 0 0 0
15.6505 18.6110 -18.7063 -15.5531 -12.9003 -9.2220 0 0 1 0 0 0
12.2220 15.7382 18.7382 -18.7382 -15.8181 -12.9911 0 0 0 1 0 0
9.5672 12.4637 15.9090 18.9090 -18.1334 -15.9911 0 0 0 0 1 0
6.0333 9.9045 12.1334 15.9045 18.5531 -18.0000 0 0 0 0 0 1
3.3708 6.1275 9.1098 12.2495 15.2996 18.6577 0 0 0 0 0 0
```

3. Test file for 6-input wrap detector network - real data

Test data is presented to the network in the same manner as training data, but must vary from the original training set:

```
!Test data for 6-input unwrapper
!Compiled from REAL DATA

-8.3723 -7.5633 -6.6345 -5.4542 -4.4520 -3.0987 0 0 0 0 0 0
8.0164 -8.8100 -7.4566 -6.0909 -5.0246 -4.3358 0 1 0 0 0 0
7.5243 8.3981 -8.3412 -7.8906 -6.1415 -5.0742 0 0 1 0 0 0
6.9022 7.1000 8.6110 -8.7063 -7.5500 -6.4306 0 0 0 1 0 0
5.0305 6.3101 7.0925 8.1111 -8.6516 -7.0001 0 0 0 0 1 0
4.0097 5.3399 6.1701 7.1134 8.2205 -8.6673 0 0 0 0 0 1
3.0987 4.7634 5.2367 6.6721 7.1762 8.0045 0 0 0 0 0 0
-12.9222 -10.7000 -8.2468 -6.0704 -4.2928 -2.9205 0 0 0 0 0 0
12.6505 -12.0456 -10.6453 -8.3456 -6.5634 -4.5623 0 1 0 0 0 0
10.4638 12.3924 -12.3041 -10.1111 -8.7777 -6.4532 0 0 1 0 0 0
8.0986 10.6534 12.3625 -12.0121 -10.5553 -8.6664 0 0 0 1 0 0
6.5876 8.7106 10.5269 12.1710 -12.1957 -10.1822 0 0 0 0 1 0
4.1969 6.6100 8.1007 10.1212 12.6520 -12.1069 0 0 0 0 0 1
2.8888 4.8511 6.1098 8.2495 10.1597 12.0948 0 0 0 0 0 0
-50.6734 -45.6914 -40.9163 -35.3945 -30.0153 -25.8421 0 0 0 0 0 0
50.1842 -50.3701 -45.4801 -40.0348 -35.2905 -30.9191 0 1 0 0 0 0
45.4678 50.3460 -50.9987 -45.6457 -40.6345 -35.2134 0 0 1 0 0 0
40.6450 45.2343 50.6534 -50.6354 -45.3645 -40.3450 0 0 0 1 0 0
35.9995 40.0202 45.8516 50.6764 -50.0519 -45.2052 0 0 0 0 1 0
30.2052 35.4343 40.9086 45.4611        50.6123 -50.9153 0 0 0 0 0 1
25.6506 30.7101 35.0933 40.6640 45.7007 50.7170 0 0 0 0 0 0
-18.6459 -15.3452 -12.8967 -9.2397 -6.4793 -3.5910 0 0 0 0 0 0
18.1969 -18.1958 -15.1939 -12.1966 -9.1985 -6.1944 0 1 0 0 0 0
15.6505 18.6110 -18.7063 -15.5531 -12.9003 -9.2220 0 0 1 0 0 0
12.2220 15.7382 18.7382 -18.7382 -15.8181 -12.9911 0 0 0 1 0 0
9.5672 12.4637 15.9090 18.9090 -18.1334 -15.9911 0 0 0 0 1 0
6.0333 9.9045 12.1334 15.9045 18.5531 -18.0000 0 0 0 0 0 1
3.3708 6.1275 9.1098 12.2495 15.2996 18.6577 0 0 0 0 0 0
0.0000 3.0550 6.6742 9.0963 12.6523 15.1212 0 0 0 0 0 0
-15.6409 -12.5634 -9.9074 -3.9071 -3.5507 0 0 0 0 0 0 0
-125.3616 -124.5508 -123.3408 -122.2881 -121.5761 -120.9055 0 0 0 0 0 0
125.5665 -125.3421 -124.3358 -123.6723 -122.6577 -121.6723 0 1 0 0 0 0
124.8745 125.4532 -125.1047 -124.3846 -123.2649 -122.6206 0 0 1 0 0 0
123.5347 124.3693 125.3693 -125.6734 -124.3748 -123.6751 0 0 0 1 0 0
122.8981 123.3616 124.9999 125.0451 -125.5611 -124.9575 0 0 0 0 1 0
121.8582 122.6374 123.6110 124 124.5508 -125.5443 0 0 0 0 0 1
119.9022 120.8981 121.6374 122.6577 123.7063 124 0 0 0 0 0 0
-117.1701 -116.9022 -115.3616 -114.8981 -113.7063 -112.6110 0 0 0 0 0 0
50.2525 51.2525 52.2525 -52.2525 -51.2525 -50.2525 0 0 0 1 0 0
99.5876 101.5876 -101.5876 -99.5876 -97.5876 -95.5876 0 0 1 0 0 0
22.3652 24.3971 26.3601 29.5897 -28.3481 -26.3467 0 0 0 0 1 0
-40.4534 -39.6781 -37.1345 -34.8734 -33.7384 -30.2134 0 0 0 0 0 0
```

4. Training file for 3x3 tile network

The following data are examples of training vectors for a 3x3 tile network. The first 9 values are presented to the input layer of the network and the remaining values presented to the output layer as desired outputs. The ampersands present inform NeuralWorks that the line is a continuation of a single training vector. Only a small representation of a training set is shown here for clarity. A complete training set will typically contain 70 to 100 vectors, each containing 18 values.

```
!1
 0.171687 0.171687  0.171687
& -0.085152 -0.085152 -0.085152
& -0.345478 -0.345478 -0.345478
&  0 0 0
&  0 0 0
&  0 0 0

!2
 -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 2.622335 2.622335 2.622335
&  1 1 1
&  0 0 0
&  0 0 0

!3
 -2.809359 -2.809359 -2.809359
& -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 1 1 1
& 1 1 1
& 0 0 0

!4
-2.527518 -2.527518 -2.527518
& -2.809359 -2.809359 -2.809359
& -3.092389 -3.092389 -3.092389
& 0 0 0
& 0 0 0
& 0 0 0

!5
 -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 2.622335 2.622335 2.622335
& 1 1 1
& 0 0 0
& 0 0 0

!6
 -2.809359 -2.809359 -2.809359
& -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 1 1 1
& 1 1 1
& 0 0 0

!7
 -2.809359 -2.809359 -2.809359
& -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 1 1 1
& 1 1 1
& 0 0 0

!8
 -2.809359 -2.809359 -2.809359
& -3.092389 -3.092389 -3.092389
& 2.906866 2.906866 2.906866
& 1 1 1
& 1 1 1
& 0 0 0

!9
 -2.809359 -3.092389 -2.809359
& -3.092389 2.906866 -3.092389
& 2.906866 2.622335 2.906866
& 1 1 1
& 1 0 1
& 0 0 0

!10
-3.092389 -3.092389 -2.809359
& 2.906866 2.906866 -3.092389
& 2.622335 2.622335 2.906866
& 1 1 1
& 0 0 1
& 0 0 0
```

5. Test file for 3x3 tile network

Shown below is a section of a typical test file for a 9-input network as described previously:

```
3.141607 3.141607 3.141607
& 3.141607 3.141607 3.141607
& 2.944487 2.944487 2.944487
& 0 0 0
& 0 0 0
& 0 0 0

2.747367 2.747367 2.747367
& 2.550247 2.550247 2.550247
& 2.550247 2.550247 2.550247
& 0 0 0
& 0 0 0
& 0 0 0

0.899367 0.899367 0.899367
& 0.677607 0.677607 0.677607
& 0.677607 0.677607 0.677607
& 0 0 0
& 0 0 0
& 0 0 0

0.234087 0.234087 0.234087
& 0.012327 0.012327 0.012327
& 0.012327 0.012327 0.012327
& 0 0 0
& 0 0 0
& 0 0 0

0.012327 0.012327 0.012327
& 0.012327 0.012327 0.012327
& -0.209433 -0.209433 -0.209433
& 0 0 0
& 0 0 0
& 0 0 0

0.012327 0.012327 0.012327
& -0.209433 -0.209433 -0.209433
& -0.209433 -0.209433 -0.209433
& 0 0 0
& 0 0 0
& 0 0 0

-0.874713 -0.874713 -0.874713
& -1.096473 -1.096473 -1.096473
& -1.096473 -1.096473 -1.096473
& 0 0 0
& 0 0 0
& 0 0 0

-1.811033 -1.811033 -1.811033
& -2.032793 -2.032793 -2.032793
& -2.032793 -2.032793 -2.032793
& 0 0 0
& 0 0 0
& 0 0 0

-2.279193 -2.279193 -2.279193
& -2.279193 -2.279193 -2.279193
& 2.944487 2.944487 2.944487
& 0 0 0
& 0 0 0
& 1 1 1

-2.279193 -2.279193 -2.279193
& 2.944487 2.944487 2.944487
& 2.944487 2.944487 2.944487
& 0 0 0
& 1 1 1
& 0 0 0

2.944487 2.944487 2.944487
& 2.944487 2.944487 2.944487
& 2.722727 2.722727 2.722727
& 0 0 0
& 0 0 0
& 0 0 0

2.476327 2.476327 2.476327
& 2.229927 2.229927 2.229927
& 2.229927 2.229927 2.229927
& 0 0 0
& 0 0 0
& 0 0 0
```

6. Results file for 6-input wrap detector

Shown below is a section of a typical results file for the 6-input wrap detector network. Each line shows 12 values, the first six being the desired output for a given test vector, the remaining values being the network's actual outputs.

```
! Date: Tue Apr  7 17:04:09 1995
! Result File: real_tes.nnr   Input File: real_tes.nna
! Network: Original edge detector
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000183    0.001781    0.003427    0.000078
     0.001112    0.001717
     0.000000    0.000000    0.000000    1.000000    0.000000
     0.000000    0.037535    0.045618    0.200727    0.730302
     0.093005    0.069129
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000297    0.005755    0.007189    0.000181
     0.001588    0.000708
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000199    0.002337    0.003006    0.000077
     0.001212    0.002002
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000778    0.000516    0.000400    0.003123
     0.008278    0.012993
     0.000000    1.000000    0.000000    0.000000    0.000000
     0.000000    0.007939    0.919821    0.044496    0.015709
     0.007932    0.007844
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000263    0.004397    0.005232    0.000136
     0.001451    0.001049
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000201    0.002340    0.002970    0.000079
     0.001223    0.002027
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000774    0.000487    0.000419    0.003158
     0.008260    0.012865
     0.000000    1.000000    0.000000    0.000000    0.000000
     0.000000    0.007937    0.919756    0.044451    0.015696
     0.007923    0.007838
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000221    0.002953    0.003790    0.000096
     0.001283    0.001535
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000213    0.002208    0.002638    0.000093
     0.001329    0.002190
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000887    0.000357    0.000595    0.003895
     0.010437    0.016609
     0.000000    0.000000    0.000000    1.000000    0.000000
     0.000000    0.039308    0.051813    0.185710    0.729952
     0.098647    0.075479
     0.000000    0.000000    0.000000    0.000000    0.000000
     0.000000    0.000467    0.015770    0.017736    0.000460
     0.002170    0.000240
```

Appendix 2

Phase unwrapping code

Code for unwrapping 1-D phase distributions using the six input network

```
/*nwcall11.c*/
/*(c) DJT 1995*/

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

/*Declare variables*/

int i, offset;
float Yin[6], Yout[6], Xout[20], buf, diff;
float inpixels[256], outpixels[256], check, pixold[6]={0,0,0,0,0,0};
float unwrap=0.0, Ywrap[256], pi=3.1415927, phi[256];
FILE *invals, *outvals;
char name[20];

/*Main program*/

main()
{

/* Open input file and scan values... */
printf("\nSpecify input file...");
scanf("%s", &name);
invals=fopen(name,"r");
for(i=0;i<256;i++)
        {
        fscanf(invals,"%f\n",&buf);
        inpixels[i]=buf;
        }
fclose(invals);

for(i=0;i<256;i++)
  phi[i]=(0.0246399*inpixels[i])-3.1415927;

for(offset=0;offset<251;offset++)
  {
  for (i=0;i<6;i++)
    Yin[i]=phi[offset+i];

/* Following code generated by NeuralWorks Professional II Plus*/

  /* Generating code for PE 0 in layer 3 */
  Xout[8]  = (float)(.24440868) + (float)(1.1412765) * Yin[0] +
    (float)(-0.075542443) * Yin[1] + (float)(-1.5840563) * Yin[2] +
    (float)(1.6389939) * Yin[3] + (float)(-.16028091) * Yin[4] +
    (float)(-.94976723) * Yin[5];
  Xout[8] = 1.0 / (1.0 + exp( -Xout[8] ));
```

```
/* Generating code for PE 1 in layer 3 */
Xout[9]  = (float)(.10932224) + (float)(.67197585) * Yin[0] +
   (float)(.26122794) * Yin[1] + (float)(-1.7042081) * Yin[2] -
   (float)(-.37387753) * Yin[3] + (float)(.55939311) * Yin[4] -
   (float)(.91165006) * Yin[5];
Xout[9] = 1.0 / (1.0 + exp( -Xout[9] ));

/* Generating code for PE 2 in layer 3 */
Xout[10]  = (float)(.50727248) + (float)(1.2653984) * Yin[0] +
   (float)(.47456065) * Yin[1] + (float)(.23860323) * Yin[2] +
   (float)(-.23652704) * Yin[3] + (float)(-.5585525) * Yin[4] +
   (float)(-.96953464) * Yin[5];
Xout[10] = 1.0 / (1.0 + exp( -Xout[10] ));

/* Generating code for PE 3 in layer 3 */
Xout[11]  = (float)(.52389967) + (float)(-.47292721) * Yin[0] +
   (float)(2.0644493) * Yin[1] + (float)(.15990584) * Yin[2] +
   (float)(-.1451918) * Yin[3] + (float)(-.46502253) * Yin[4] +
   (float)(-.85294384) * Yin[5];
Xout[11] = 1.0 / (1.0 + exp( -Xout[11] ));

/* Generating code for PE 4 in layer 3 */
Xout[12]  = (float)(.58212614) + (float)(1.1654203) * Yin[0] +
   (float)(.22101815) * Yin[1] + (float)(-.27567723) * Yin[2] +
   (float)(-.39259917) * Yin[3] + (float)(-1.4805681) * Yin[4] +
   (float)(.23438235) * Yin[5];
Xout[12] = 1.0 / (1.0 + exp( -Xout[12] ));

/* Generating code for PE 5 in layer 3 */
Xout[13]  = (float)(.68133008) + (float)(1.0202456) * Yin[0] +
   (float)(.22769448) * Yin[1] + (float)(-.10406712) * Yin[2] +
   (float)(-1.4373332) * Yin[3] + (float)(1.2467792) * Yin[4] +
   (float)(-.75219929) * Yin[5];
Xout[13] = 1.0 / (1.0 + exp( -Xout[13] ));

/* Generating code for PE 0 in layer 4 */
Yout[0]  = (float)(-3.6656587) + (float)(-.96997684) * Xout[8] +
   (float)(-1.0610986) * Xout[9] + (float)(-1.030337) * Xout[10] +
   (float)(-1.0052155) * Xout[11] + (float)(-.93514925) * Xout[12] +
   (float)(-.98882818) * Xout[13];
Yout[0] = 1.0 / (1.0 + exp( -Yout[0] ));

/* Generating code for PE 1 in layer 4 */
Yout[1]  = (float)(-2.9236183) + (float)(-2.4284985) * Xout[8] +
   (float)(-1.6994104) * Xout[9] + (float)(-1.6407439) * Xout[10] +
   (float)(5.9125438) * Xout[11] + (float)(-1.879591) * Xout[12] +
   (float)(-1.647258) * Xout[13];
Yout[1] = 1.0 / (1.0 + exp( -Yout[1] ));
```

```
/* Generating code for PE 2 in layer 4 */
Yout[2]  = (float)(2.7719913) + (float)(-2.5445874) * Xout[8] +
   (float)(-3.8594689) * Xout[9] + (float)(-1.3257178) * Xout[10] +
   (float)(-5.6907859) * Xout[11] + (float)(-1.7482276) * Xout[12] +
   (float)(-2.1713367) * Xout[13];
Yout[2] = 1.0 / (1.0 + exp( -Yout[2] ));

/* Generating code for PE 3 in layer 4 */
Yout[3]  = (float)(-3.3216081) + (float)(4.3592968) * Xout[8] +
   (float)(1.9537264) * Xout[9] + (float)(-1.6712166) * Xout[10] +
   (float)(-1.9428953) * Xout[11] + (float)(-1.5179806) * Xout[12] +
   (float)(-2.3145306) * Xout[13];
Yout[3] = 1.0 / (1.0 + exp( -Yout[3] ));

/* Generating code for PE 4 in layer 4 */
Yout[4]  = (float)(-3.2834849) + (float)(-2.3915703) * Xout[8] +
   (float)(2.311928) * Xout[9] + (float)(-1.4123503) * Xout[10] +
   (float)(-1.6932757) * Xout[11] + (float)(-1.9878503) * Xout[12] +
   (float)(3.4246485) * Xout[13];
Yout[4] = 1.0 / (1.0 + exp( -Yout[4] ));

/* Generating code for PE 5 in layer 4 */
Yout[5]  = (float)(-2.8727694) + (float)(-2.0236669) * Xout[8] +
   (float)(1.8336757) * Xout[9] + (float)(-1.1584884) * Xout[10] +
   (float)(-1.6798445) * Xout[11] + (float)(3.1066597) * Xout[12] +
   (float)(-1.5766689) * Xout[13];
   Yout[5] = 1.0 / (1.0 + exp( -Yout[5] ));

check=pixold[1]-Yout[0];
if(check>0.7)
  Yout[0]=pixold[1];

for(i=0;i<6;i++)
      {
      buf=Yout[i];
      outpixels[offset+i]=buf;
      }

for(i=0;i<6;i++)
      pixold[i]=Yout[i];
   }

/*Once network has detected wraps, update where necessary by 2 pi*/

for(i=0;i<256;i++)
      {
      diff=outpixels[i]-outpixels[i+1];
      if(diff>0.80)
            {
```

```
            unwrap=unwrap+2*pi;
            }

        Ywrap[i]=phi[i]-unwrap;

        printf("%f %f %f\n", phi[i], outpixels[i], Ywrap[i]);
        }
printf("\nSpecify output filename...");
scanf("%s", name);
outvals=fopen(name,"w");
for(i=0;i<256;i++)
        fprintf(outvals, "%f\n", Ywrap[i]);
fclose(outvals);
}
```

Appendix 3

Fringe Optimisation
Training and test data

1. Global optimisation network training data
2. Global optimisation network test data

Training set for global optimisation network

The training set is presented to the network as described in Appendix 1.

```
!GLOBAL_L.nna
!Training file for global optimisation network.

0.127 0.640 0.000 0.750
& 0 0 0 0

0.075 0.850 0.000 0.600
& -1 1 0 0

0.130 0.600 -1.000 0.590
& 0 0 -1 0

0.168 0.630 0.000 0.444
& 1 0 0 -1

0.127 0.640 0.000 0.250
& 0 0 0 -1

0.127 0.640 -1.000 0.750
& 0 0 -1 0

0.127 0.640 1.000 0.750
& 0 0 1 0

0.127 0.640 -1.000 0.250
& 0 0 -1 -1

0.127 0.640 1.000 0.250
& 0 0 1 -1

0.127 0.800 0.000 0.750
& 0 1 0 0

0.127 0.400 0.000 0.750
& 0 -1 0 0

0.127 0.800 0.000 0.250
& 0 1 0 -1

0.127 0.400 0.000 0.250
& 0 -1 0 -1

0.127 0.800 1.000 0.750
& 0 1 1 0

0.127 0.800 -1.000 0.750
& 0 1 -1 0

0.127 0.400 -1.000 0.750
& 0 -1 -1 0

0.127 0.400 1.000 0.750
& 0 -1 1 0

0.127 0.800 1.000 0.250
& 0 1 1 -1

0.127 0.800 -1.000 0.250
& 0 1 -1 -1

0.127 0.400 -1.000 0.250
& 0 -1 -1 -1

0.127 0.400 1.000 0.250
& 0 -1 1 -1

0.200 0.640 0.000 0.750
& 1 0 0 0

0.075 0.640 0.000 0.750
& -1 0 0 0

0.200 0.640 0.000 0.250
& 1 0 0 -1

0.075 0.640 0.000 0.250
& -1 0 0 -1

0.200 0.640 1.000 0.750
& 1 0 1 0

0.200 0.640 -1.000 0.750
& 1 0 -1 0

0.075 0.640 1.000 0.750
& -1 0 1 0

0.075 0.640 -1.000 0.750
& -1 0 -1 0

0.200 0.640 1.000 0.250
& 1 0 1 -1

0.200 0.640 -1.000 0.250
& 1 0 -1 -1

0.075 0.640 1.000 0.250
& -1 0 1 -1

0.075 0.640 -1.000 0.250
& -1 0 -1 -1

0.200 0.700 0.000 0.750
& 1 1 0 0

0.200 0.400 0.000 0.750
& 1 -1 0 0

0.200 0.400 1.000 0.750
& 1 -1 1 0

0.200 0.400 -1.000 0.750
& 1 -1 -1 0

0.075 0.700 1.000 0.750
& -1 1 1 0

0.075 0.700 -1.000 0.750
& -1 1 -1 0

0.075 0.400 1.000 0.750
& -1 -1 1 0

0.075 0.400 -1.000 0.750
& -1 -1 -1 0

0.200 0.700 1.000 0.250
& 1 1 1 -1

0.200 0.700 -1.000 0.250
& 1 1 -1 -1

0.200 0.400 1.000 0.250
& 1 -1 1 -1

0.200 0.400 -1.000 0.250
& 1 -1 -1 -1

0.127 0.640 0.000 0.750
& 0 0 0 0

0.125 0.620 0.000 0.600
& 0 0 0 0

0.130 0.650 0.000 0.550
& 0 0 0 0

0.140 0.600 0.000 0.700
& 0 0 0 0
```

Test results for global optimisation network

The test set is as described in Appendix 1.

```
! Date: Tue Sep  5 10:17:38 1995
! Result File: global_r.nnr   Input File: global_r.nna
! Network: Global optimisation network MkI
        0.000000        0.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
       -1.000000        1.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        0.000000       -1.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        1.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        0.000000       -1.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        0.000000        1.000000        0.000000        0.004825
        0.014323        0.525409        0.000000
        0.000000        0.000000       -1.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        0.000000        1.000000       -1.000000        0.004975
        0.014264        0.677121        0.000000
        0.000000        1.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        1.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000        1.000000        1.000000        0.000000        0.001447
        0.016829        0.000000        0.000000
        0.000000        1.000000       -1.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000       -1.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        1.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
       -1.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000        1.000000        0.000000        0.004975
        0.014264        0.677121        0.000000
        0.000000        1.000000        1.000000       -1.000000        0.004975
        0.014264        0.676704        0.000000
        0.000000        1.000000       -1.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000       -1.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
        0.000000       -1.000000        1.000000       -1.000000        0.004975
        0.014264        0.677121        0.000000
        1.000000        0.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
       -1.000000        0.000000        0.000000        0.000000        0.001435
        0.016848        0.000000        0.000000
        1.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
       -1.000000        0.000000        0.000000       -1.000000        0.001435
        0.016848        0.000000        0.000000
```

Appendix 4

Code

6. Final optimisation system

```
/* Global optimisation network version 4.30 */
/* (c) DJT 18/6/96 */

#include <stdio.h>
#include <math.h>
#include <itexvsp.h>
#include <gaoi.h>
#include <string.h>

/* ITEX variables */

int v_pixarray[512], loop_counter, loop_max=10;
long int v_pixval[512], v_total, v_count;
float v_mean, v_number, v_shift[512];
float index, tilt_value, scaled_mean, scaled_count, v_product;
float mean_val[5], fringe_number[5], vis_val[5], tilt[5];
float list[512], temp, visibility, contrast;
int out, in, count, a, gaoi, i;
float Yin[20], Yout[4];
char name[20];
FILE *fp;

/* mm2000 variables */

int mm_add=0x300;
char response[80];


main()
  {
  printf("\nBackprop optimisation system version 4.30.");
  loop_counter=0;
  load_cnf("\\vi\\lib\\vsp.cnf");
  initsys();
  vfg_init();
  gaoi=gaoi_create(VFG,0,I,0,0,512,512,0,8,NONE);
  mmsend("1VA200,1AC100\r",mm_add);
  mmsend("1TB\r",mm_add);
  mmread(response,mm_add);
  printf("\nCurrent MotionMaster error state is: %s", response);
  do
  {
  printf("\n** Loop iteration number %d. **", loop_counter+1);
  printf("\nSnapping current image...");
  vfg_snap(gaoi,CAMERA);
```

```
/* top left */
vfg_rvline(gaoi,128,0,256,v_pixarray);
calculations();
mean_val[0]=scaled_mean;
fringe_number[0]=scaled_count;
vis_val[0]=visibility;

/* top right */
vfg_rvline(gaoi,384,0,256,v_pixarray);
calculations();
mean_val[1]=scaled_mean;
fringe_number[1]=scaled_count;
vis_val[1]=visibility;

/* bottom left */
vfg_rvline(gaoi,128,256,256,v_pixarray);
calculations();
mean_val[2]=scaled_mean;
fringe_number[2]=scaled_count;
vis_val[2]=visibility;

/* bottom right */
vfg_rvline(gaoi,384,256,256,v_pixarray);
calculations();
mean_val[3]=scaled_mean;
fringe_number[3]=scaled_count;
vis_val[3]=visibility;

/* centre */
vfg_rvline(gaoi,256,128,256,v_pixarray);
calculations();
mean_val[4]=scaled_mean;
fringe_number[4]=scaled_count;
vis_val[4]=visibility;

/* tilt pass 2 */
tilt[2]=fringe_number[4];

/* tilt pass 0 */
vfg_rhline(gaoi,128,384,256,v_pixarray);
calculations();
tilt[0]=scaled_count;

/* tilt pass 1 */
vfg_rline(gaoi,75,203,256,384,v_pixarray);
calculations();
tilt[1]=scaled_count;

/* tilt pass 3 */
```

```
vfg_rline(gaoi,256,384,437,203,v_pixarray);
calculations();
tilt[3]=scaled_count;

/* tilt pass 4 */
vfg_rhline(gaoi,256,384,256,v_pixarray);
calculations();
tilt[4]=scaled_count;

printf("\n\n Mean values are:       ");
for(i=0;i<5;i++)
  printf("%f  ", mean_val[i]);
printf("\n Fringe counts are:     ");
for(i=0;i<5;i++)
  printf("%f  ", fringe_number[i]);
printf("\n Visibility values are: ");
for(i=0;i<5;i++)
  printf("%f  ", vis_val[i]);
printf("\n Tilt values are:       ");
for(i=0;i<5;i++)
  printf("%f  ", tilt[i]);

for(i=0;i<5;i++)
  {
  Yin[i]=mean_val[i];
  Yin[i+5]=fringe_number[i];
  Yin[i+10]=vis_val[i];
  Yin[i+15]=tilt[i];
  }

flashcode();


printf("\n\n");
if(Yout[0]<0.3)
  printf("\nMean intensity is too low.");
else if(Yout[0]>0.75)
  printf("\nMean intensity is too high.");
else
  printf("\nMean intensity is OK.");
if(Yout[1]>0.7)
  printf("\nThere are too many fringes in this image.");
else if(Yout[1]<0.3)
  printf("\nThere are not enough fringes in this image.");
else
  printf("\nNumber of fringes is OK.");
if(Yout[2]<0.35)
  printf("\nVisibility is too low.");
else
  printf("\nVisibility is OK.");
```

```
if(Yout[3]<0.3)
  printf("\nFringes tilt anticlockwise.");
else if(Yout[3]>0.7)
  printf("\nFringes tilt clockwise.");
else
  printf("\nFringes are level.");
printf("\n\n");

if(Yout[1]>0.7)
  call_mm2000_axis1();
loop_counter++;
}
  while(Yout[1]>0.7&&loop_counter<loop_max);

  }

calculations()
  {
  calc_means();
  fringe_counts();

  scaled_mean=v_mean/1000;
  scaled_count=v_number/100;
  }

calc_means()
  {
  v_total=0;
  for(a=0;a<256;a++)
    v_total+=v_pixarray[a];
  v_mean=v_total/256;
  }

fringe_counts()
  {
  for(a=0;a<256;a++)
    v_shift[a]=v_pixarray[a]-v_mean;
  v_count=0;
  for(a=0;a<256;a++)
    {
  v_product=v_shift[a]*v_shift[a+1];
  if(v_product<0.0)
    v_count++;
  }
v_number=v_count/2;

for(a=0;a<256;a++)
  list[a]=v_pixarray[a];
for(out=0;out<256;out++)
  for(in=out+1;in<256;in++)
```

```
        if(list[out]>list[in])
           {
           temp=list[in];
           list[in]=list[out];
           list[out]=temp;
           }

  visibility=(list[255]-list[0])/(list[255]+list[0]);
  }

flashcode()
{
/* Tue Mar 12 12:59:54 1996 (glnet414.c) */
/* Recall-Only Run-time for <gl400> */
/* Control Strategy is: <backprop> */

 float  Xout[36], Xsum[36];  /* work arrays */
 long   ICmpT;  /* temp for comparisons */

 /* *** WARNING: Code generated assuming Recall = 0 *** */

 /* Read and scale input into network */

 /* Generating code for PE 0 in layer 3 */
 Xsum[22]  = (float)(0.51291823) + (float)(1.1200382) * Yin[0] +
    (float)(0.75770199) * Yin[1] + (float)(0.94658583) * Yin[2] +
    (float)(0.74116755) * Yin[3] + (float)(0.86861163) * Yin[4] +
    (float)(-0.11502891) * Yin[5] + (float)(-0.55039108) * Yin[6] +
    (float)(-0.42057505) * Yin[7] + (float)(-0.43457967) * Yin[8] +
    (float)(-0.63996595) * Yin[9];
 Xsum[22] += (float)(0.52264649) * Yin[10] + (float)(0.12820528) * Yin[11]
     + (float)(0.59060228) * Yin[12] + (float)(0.085514538) * Yin[13] +
    (float)(-0.066451773) * Yin[14] + (float)(0.59558135) * Yin[15] +
    (float)(-0.38854396) * Yin[16] + (float)(-0.60187453) * Yin[17] +
    (float)(-0.32544455) * Yin[18] + (float)(-0.5333479) * Yin[19];

 /* Generating code for PE 1 in layer 3 */
 Xsum[23]  = (float)(0.41995716) + (float)(0.31209549) * Yin[0] +
    (float)(-0.042181447) * Yin[1] + (float)(0.17271914) * Yin[2] +
    (float)(0.052135143) * Yin[3] + (float)(0.10904433) * Yin[4] +
    (float)(-0.27623224) * Yin[5] + (float)(-0.57313389) * Yin[6] +
    (float)(-0.61444414) * Yin[7] + (float)(-0.5735063) * Yin[8] +
    (float)(-0.48885798) * Yin[9];
 Xsum[23] += (float)(-0.37919202) * Yin[10] + (float)(-0.1654823) * Yin[11]
     + (float)(-0.16504616) * Yin[12] + (float)(-0.13199218) * Yin[13] +
    (float)(-0.083629504) * Yin[14] + (float)(-0.39430356) * Yin[15] +
    (float)(-0.57887322) * Yin[16] + (float)(-0.5394156) * Yin[17] +
    (float)(-0.45572984) * Yin[18] + (float)(-0.47956526) * Yin[19];
```

```
/* Generating code for PE 2 in layer 3 */
Xsum[24]  = (float)(-0.19887754) + (float)(0.36199242) * Yin[0] +
   (float)(0.22469084) * Yin[1] + (float)(0.12849589) * Yin[2] +
   (float)(0.1453943) * Yin[3] + (float)(0.25397474) * Yin[4] +
   (float)(0.058965519) * Yin[5] + (float)(0.099251397) * Yin[6] +
   (float)(0.17505574) * Yin[7] + (float)(0.075894788) * Yin[8] +
   (float)(0.26560712) * Yin[9];
Xsum[24] += (float)(-0.69191957) * Yin[10] + (float)(-0.13290976) * Yin[11]
    + (float)(-0.34417331) * Yin[12] + (float)(-0.53462374) * Yin[13] +
   (float)(-0.25832948) * Yin[14] + (float)(0.11986393) * Yin[15] +
   (float)(-0.048770867) * Yin[16] + (float)(0.07210812) * Yin[17] +
   (float)(0.23395152) * Yin[18] + (float)(-0.13973524) * Yin[19];

/* Generating code for PE 3 in layer 3 */
Xsum[25]  = (float)(-0.81400353) + (float)(-0.91880965) * Yin[0] +
   (float)(-0.64511019) * Yin[1] + (float)(-0.77062106) * Yin[2] +
   (float)(-0.623357) * Yin[3] + (float)(-0.67265278) * Yin[4] +
   (float)(0.30170408) * Yin[5] + (float)(0.65375519) * Yin[6] +
   (float)(0.47448733) * Yin[7] + (float)(0.60599393) * Yin[8] +
   (float)(0.40640268) * Yin[9];
Xsum[25] += (float)(0.29200983) * Yin[10] + (float)(0.41514421) * Yin[11]
    + (float)(0.1017247) * Yin[12] + (float)(0.28909576) * Yin[13] +
   (float)(0.492726) * Yin[14] + (float)(0.024698786) * Yin[15] +
   (float)(0.1335637) * Yin[16] + (float)(0.79508579) * Yin[17] +
   (float)(0.63817501) * Yin[18] + (float)(0.64715672) * Yin[19];

/* Generating code for PE 4 in layer 3 */
Xsum[26]  = (float)(-0.7262122) + (float)(-0.43169123) * Yin[0] +
   (float)(-0.11978691) * Yin[1] + (float)(-0.084116541) * Yin[2] +
   (float)(-0.35286617) * Yin[3] + (float)(-0.14322607) * Yin[4] +
   (float)(0.79265034) * Yin[5] + (float)(1.0748861) * Yin[6] +
   (float)(0.86745197) * Yin[7] + (float)(0.51031208) * Yin[8] +
   (float)(0.73114151) * Yin[9];
Xsum[26] += (float)(-0.77761388) * Yin[10] + (float)(-0.25369096) * Yin[11]
    + (float)(-0.4436211) * Yin[12] + (float)(-0.62311947) * Yin[13] +
   (float)(-0.057868429) * Yin[14] + (float)(0.25251833) * Yin[15] +
   (float)(0.68171388) * Yin[16] + (float)(0.64202815) * Yin[17] +
   (float)(0.76848215) * Yin[18] + (float)(0.38853914) * Yin[19];

/* Generating code for PE 5 in layer 3 */
Xsum[27]  = (float)(-0.077568538) + (float)(0.25395143) * Yin[0] +
   (float)(0.3964766) * Yin[1] + (float)(0.040671058) * Yin[2] +
   (float)(-0.016429115) * Yin[3] + (float)(0.086688489) * Yin[4] +
   (float)(-0.47465959) * Yin[5] + (float)(-0.52561438) * Yin[6] +
   (float)(-0.18413264) * Yin[7] + (float)(-0.45139271) * Yin[8] +
   (float)(-0.20107581) * Yin[9];
Xsum[27] += (float)(-0.39822865) * Yin[10] + (float)(-0.28633997) * Yin[11]
    + (float)(0.115711) * Yin[12] + (float)(0.0018749614) * Yin[13] +
   (float)(-0.2547664) * Yin[14] + (float)(-0.21335608) * Yin[15] +
   (float)(-0.53907776) * Yin[16] + (float)(-0.25679442) * Yin[17] +
```

```
    (float)(-0.29961729) * Yin[18] + (float)(-0.55880654) * Yin[19];

/* Generating code for PE 6 in layer 3 */
Xsum[28]  = (float)(0.22698164) + (float)(-0.30818063) * Yin[0] +
    (float)(-0.063706584) * Yin[1] + (float)(-0.21834689) * Yin[2] +
    (float)(-0.41436657) * Yin[3] + (float)(-0.13816226) * Yin[4] +
    (float)(-0.6952529) * Yin[5] + (float)(-0.61804426) * Yin[6] +
    (float)(-0.475234) * Yin[7] + (float)(-0.021478163) * Yin[8] +
    (float)(-0.46860024) * Yin[9];
Xsum[28] += (float)(0.34855863) * Yin[10] + (float)(-0.20231442) * Yin[11]
     + (float)(-0.17604984) * Yin[12] + (float)(0.22505289) * Yin[13] +
    (float)(0.29778031) * Yin[14] + (float)(-0.50480717) * Yin[15] +
    (float)(-0.30938923) * Yin[16] + (float)(-0.53164971) * Yin[17] +
    (float)(-0.20265885) * Yin[18] + (float)(-0.3119747) * Yin[19];

/* Generating code for PE 7 in layer 3 */
Xsum[29]  = (float)(-0.12704676) + (float)(1.4800271) * Yin[0] +
    (float)(1.2682054) * Yin[1] + (float)(1.0784848) * Yin[2] +
    (float)(0.95180172) * Yin[3] + (float)(1.3675277) * Yin[4] +
    (float)(0.32492545) * Yin[5] + (float)(-0.11640698) * Yin[6] +
    (float)(-0.2024758) * Yin[7] + (float)(-0.52650571) * Yin[8] +
    (float)(-0.099969789) * Yin[9];
Xsum[29] += (float)(-0.47220179) * Yin[10] + (float)(-0.28866726) * Yin[11]
     + (float)(0.2041236) * Yin[12] + (float)(-0.54379195) * Yin[13] +
    (float)(-0.57043999) * Yin[14] + (float)(0.54553699) * Yin[15] +
    (float)(0.079543613) * Yin[16] + (float)(-0.030147526) * Yin[17] +
    (float)(-0.43671697) * Yin[18] + (float)(-0.21133904) * Yin[19];

/* Generating code for PE 8 in layer 3 */
Xsum[30]  = (float)(-1.1289997) + (float)(-1.7301096) * Yin[0] +
    (float)(-1.2706949) * Yin[1] + (float)(-1.5117877) * Yin[2] +
    (float)(-1.3128918) * Yin[3] + (float)(-1.4757698) * Yin[4] +
    (float)(0.65220791) * Yin[5] + (float)(1.7428278) * Yin[6] +
    (float)(0.61475885) * Yin[7] + (float)(1.2227138) * Yin[8] +
    (float)(1.0847834) * Yin[9];
Xsum[30] += (float)(-1.5691651) * Yin[10] + (float)(-0.85616481) * Yin[11]
     + (float)(-1.7383595) * Yin[12] + (float)(-1.1450688) * Yin[13] +
    (float)(-0.40118957) * Yin[14] + (float)(-0.58298486) * Yin[15] +
    (float)(0.99605548) * Yin[16] + (float)(0.77839392) * Yin[17] +
    (float)(0.60149568) * Yin[18] + (float)(1.1602714) * Yin[19];

/* Generating code for PE 9 in layer 3 */
Xsum[31]  = (float)(0.096551582) + (float)(-0.050478779) * Yin[0] +
    (float)(0.057656057) * Yin[1] + (float)(-0.03250061) * Yin[2] +
    (float)(-0.036905576) * Yin[3] + (float)(-0.2051387) * Yin[4] +
    (float)(-0.41296607) * Yin[5] + (float)(-0.50270027) * Yin[6] +
    (float)(-0.41660067) * Yin[7] + (float)(-0.16561231) * Yin[8] +
    (float)(-0.30192643) * Yin[9];
Xsum[31] += (float)(-0.10400457) * Yin[10] + (float)(-0.51148754) * Yin[11]
     + (float)(-0.3807762) * Yin[12] + (float)(-0.069051974) * Yin[13] +
```

```
    (float)(0.022478757) * Yin[14] + (float)(-0.27212459) * Yin[15] +
    (float)(-0.072177403) * Yin[16] + (float)(-0.13238665) * Yin[17] +
    (float)(-0.18513454) * Yin[18] + (float)(-0.22440013) * Yin[19];

/* Generating code for PE 0 in layer 3 */
Xout[22] = 1.0 / (1.0 + exp( -Xsum[22] ));

/* Generating code for PE 1 in layer 3 */
Xout[23] = 1.0 / (1.0 + exp( -Xsum[23] ));

/* Generating code for PE 2 in layer 3 */
Xout[24] = 1.0 / (1.0 + exp( -Xsum[24] ));

/* Generating code for PE 3 in layer 3 */
Xout[25] = 1.0 / (1.0 + exp( -Xsum[25] ));

/* Generating code for PE 4 in layer 3 */
Xout[26] = 1.0 / (1.0 + exp( -Xsum[26] ));

/* Generating code for PE 5 in layer 3 */
Xout[27] = 1.0 / (1.0 + exp( -Xsum[27] ));

/* Generating code for PE 6 in layer 3 */
Xout[28] = 1.0 / (1.0 + exp( -Xsum[28] ));

/* Generating code for PE 7 in layer 3 */
Xout[29] = 1.0 / (1.0 + exp( -Xsum[29] ));

/* Generating code for PE 8 in layer 3 */
Xout[30] = 1.0 / (1.0 + exp( -Xsum[30] ));

/* Generating code for PE 9 in layer 3 */
Xout[31] = 1.0 / (1.0 + exp( -Xsum[31] ));

/* Generating code for PE 0 in layer 4 */
Xsum[32]  = (float)(-0.65057516) + (float)(1.0928892) * Xout[22] +
   (float)(0.33727017) * Xout[23] + (float)(0.38339254) * Xout[24] +
   (float)(-1.1274824) * Xout[25] + (float)(-0.40105018) * Xout[26] +
   (float)(0.34875277) * Xout[27] + (float)(-0.48891786) * Xout[28] +
   (float)(1.5872207) * Xout[29] + (float)(-2.3664045) * Xout[30] +
   (float)(-0.15500106) * Xout[31];
Yout[0] = 1.0 / (1.0 + exp( -Xsum[32] ));

/* Generating code for PE 1 in layer 4 */
Xsum[33]  = (float)(0.078558519) + (float)(-0.67272425) * Xout[22] +
   (float)(-0.98405063) * Xout[23] + (float)(0.24272983) * Xout[24] +
   (float)(1.1989859) * Xout[25] + (float)(1.5565525) * Xout[26] +
   (float)(-0.72113997) * Xout[27] + (float)(-0.97956073) * Xout[28] +
   (float)(-0.15540032) * Xout[29] + (float)(1.4880245) * Xout[30] +
   (float)(-0.64811188) * Xout[31];
```

```
Yout[1] = 1.0 / (1.0 + exp( -Xsum[33] ));

/* Generating code for PE 2 in layer 4 */
Xsum[34]  = (float)(-0.18949994) + (float)(0.82227021) * Xout[22] +
   (float)(-0.47706285) * Xout[23] + (float)(-0.37742797) * Xout[24] +
   (float)(-0.16749881) * Xout[25] + (float)(-0.31269461) * Xout[26] +
   (float)(-0.28799385) * Xout[27] + (float)(-0.069637783) * Xout[28] +
   (float)(0.62282759) * Xout[29] + (float)(-2.2626534) * Xout[30] +
   (float)(-0.10584904) * Xout[31];
Yout[2] = 1.0 / (1.0 + exp( -Xsum[34] ));

/* Generating code for PE 3 in layer 4 */
Xsum[35]  = (float)(0.07574404) + (float)(-0.067089461) * Xout[22] +
   (float)(-0.016239651) * Xout[23] + (float)(-0.12841272) * Xout[24] +
   (float)(0.47594422) * Xout[25] + (float)(-0.1223451) * Xout[26] +
   (float)(0.023014756) * Xout[27] + (float)(0.32377771) * Xout[28] +
   (float)(-0.6264416) * Xout[29] + (float)(0.050953642) * Xout[30] +
   (float)(-0.23612306) * Xout[31];
Yout[3] = 1.0 / (1.0 + exp( -Xsum[35] ));

printf("\n\nNetwork outputs are:    ");
for(i=0;i<4;i++)
  printf("%f  ", Yout[i]);

}

call_mm2000_axis1()
  {
  printf("\nAdjusting fibres...");
  mmsend("1TB\r",mm_add);
  mmread(response,mm_add);
  printf("\nCurrent MotionMaster error state is: %s\n", response);
  strcmp(response,"E00 NO ERROR");
  if(!strcmp)
    {
    printf("\nMotionMaster error. Aborting...");
    Yout[1]=0.5;
    }
  mmsend("1PR1000\r",mm_add);
  }

}
```

Appendix 5: Published works

A NEURAL NETWORK APPROACH TO THE PHASE UNWRAPPING PROBLEM IN FRINGE ANALYSIS

D. J. TIPPER*, D. R. BURTON and M. J. LALOR
Coherent and Electro-optics Research Group
Liverpool John Moores University, School of Engineering and Technology Management, Byrom Street, Liverpool, L3 3AF

This paper presents a novel approach to the phase unwrapping problem by employing a back-propagation neural network to detect the presence of phase wraps in an image. The philosophy behind the approach is to keep the analysis simple by using a small network consisting only of six input, six hidden and six output neurons. Each input neuron is assigned to one pixel and this input "window" is convolved with an image to analyse only six pixels at a time. The unwrapped phase distribution is reconstructed from this series of analyses. It is shown that after training for approximately two hours, the network can successfully unwrap a one-dimensional phase distribution in 0.5 seconds and that this method could prove to be the basis for a robust two dimensional phase unwrapper.

INTRODUCTION

The aim of this work is to investigate the use of neural network techniques to assist in the phase unwrapping of fringe images. The phase unwrapping problem occurs due to the mathematical processes involved in the calculation of the phase values of an image. To produce a fringe image, two mutually coherent light beams are made to cross. At the point where the beams cross, interference fringes are produced whose intensity profile is sinusoidal. The intensity of illumination at any given point, I(x,y), is expressed as:

$$I(x, y) = a(x, y) + b(x, y) \cos [\phi(x, y)]$$

Where

$a(x, y)$ = additive noise – offset term
$b(x, y)$ = multiplicative noise - amplitude term
$\phi(x, y) = \phi_c + \phi_m$
ϕ_c = carrier phase
ϕ_m = modulation phase

If a surface is illuminated with a cosinusoidal fringe pattern, variations in intensity across that surface are observed. If the value of the modulation phase is known, it

is possible to reconstruct a map of the surface [1]. To find the modulation phase, it is desirable to eliminate the amplitude and offset terms. To make the analysis of such fringe patterns possible, they are subject to either phase stepping [2] or Fast Fourier Transform (FFT) [3] techniques.

Phase stepping involves measuring the intensity values of several fringe patterns where the phase value of each one is shifted by a known amount. Typically, four images are used whose phase values differ by intervals of $\pi/2$.

From equation (1), the intensity values are given by

$$\begin{aligned} I(0)\ (x, y) &= a(x, y) + b(x, y) \cos\ [\phi(x, y)] \\ I(\pi/2)\ (x, y) &= a(x, y) + b(x, y) \cos\ [\phi(x, y) + \pi/2] \\ I(\pi)\ (x, y) &= a(x, y) + b(x, y) \cos\ [\phi(x, y) + \pi] \\ I(3\pi/2)\ (x, y) &= a(x, y) + b(x, y) \cos\ [\phi(x, y) + 3\pi/2] \end{aligned}$$

The intensity equations can be solved simultaneously to give

$$\tan\phi(x, y) = \{I(\pi/2) - I(3\pi/2)\}/\{I(0) - I(\pi)\}$$

The phase value at any point x, y is, therefore, given by

$$\phi\ (x, y) = \arctan\{I(\pi/2) - I(3\pi/2)\}/\{I(0) - I(\pi)\}$$

The FFT was developed as a fringe analysis tool by Takeda *et al.* [4] and involves the use of a single, modulated phase map. Using this approach an FFT algorithm is applied to the image and the result is filtered and inverse transformed to return an image with real and imaginary parts.

When the intensity profile of a fringe pattern is described by

$$I(x, y) = a(x, y) + b(x, y)\cos[\phi(x, y)]$$

the equation can be re-written to allow for tilting of wavefronts on non-uniform surface as

$$I(x, y) = a(x, y) + c(x, y) \exp\ (2\pi i\ .\ f_0.x) + c^*(x, y)\ \exp\ (-2\pi i.\ f_0.\ x)$$

where

$$c(x, y) = [b(x, y)/2]\exp\{i\phi(x, y)\}$$

and $c^*(x, y)$ is the complex conjugate of $c(x, y)$

If the Fourier transform of the equation is taken, the equation becomes

$$I(f, y) = A(f, y) + C(f - f_0, y) + C^*(f + f_0, y)$$

where the capital letters represent the Fourier spectra and f represents the spatial frequency in the x direction. The Fourier spectra are separated by a carrier frequency f_0. $C(f - f_0, y)$ can be isolated by means of a filter and the inverse FFT can be taken. giving

$$f(x, y) = \arctan \{Im[c(x, y)]\}/\{Re[c(x, y)]\}$$

where Im = imaginary component of c(x, y)
Re = real component of c(x, y)

The Phase Unwrapping Problem

Both the phase stepping and FFT methods give the phase values for an image as an arctangent function. The mathematical nature of this function causes the final phase values to be returned wrapped modulo 2π. This phenomenon is explained in Figure 1. To accurately reproduce a map of the object's surface, the phase values must be reconstructed, or *unwrapped*. When real wrapped phase distributions contain noise, it becomes difficult for the unwrapper to distinguish between noise and genuine phase wraps. This is especially the case when using classical algorithmic unwrappers.

To date, many algorithms for the unwrapping of phase values have been proposed, the earliest being "Schafer's Algorithm" [5]. This involves comparison of adjacent pixel values. A large number of decisions must be made and any errors can be propagated throughout the entire image. Numerous algorithms have expanded on the basis idea, the most robust utilising regional rather than global analysis [6,7,8]. However, no unwrapper is completely noise-immune.

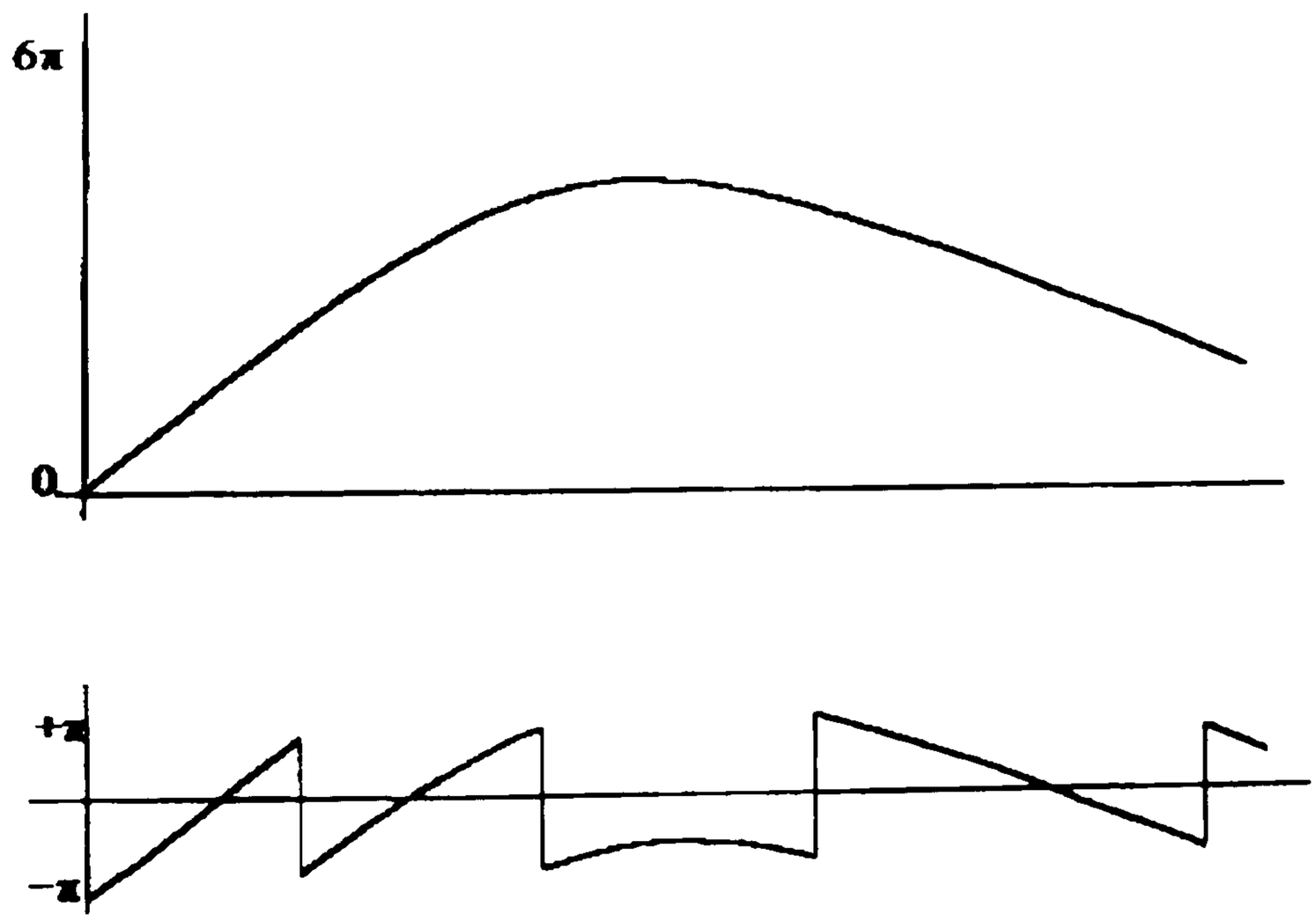


Figure 1 A 1-D phase distribution and how it appears when wrapped modulo 2π

NEURAL COMPUTING

Background

Throughout history, many theories have been proposed to explain the operation of the human brain. It is only since the nineteenth century, however, that any real insight into its operation has been gained. The fundamental element of the nervous system is the *neuron*. The average brain contains approximately five *billion* neurons, all of which have five specific functions: they receive signals from neighbouring neurons, integrate these signals, give rise to nerve pulses, conduct these pulses and transmit them to other neurons. It is from an idea for a mathematical model of the neuron first postulated by McCulloch and Pitts [9] in the 1940s that the concept of the neural network originates. The neural network is a computing paradigm which differs from conventional computing as it "learns through experience", rather than utilising an algorithmic approach. Each processing element of a neural network mimics the operation of a biological neuron by behaving as a simple thresholding device. When a large number of neurons are connected together, a large number of simple calculations can be combined to achieve a more complex result. A network consists of a layer input neurons, a layer of output neurons and one or more layers of "hidden" neurons, so called because their input and output values are not known by the user. Three types of learning are associated with neural networks.: supervised, unsupervised and reinforcement. Supervised learning involves presenting the network with an input stimulus and an expected response, whereas unsupervised learning only makes use of an input stimulus and leaves the network to calculate its own output. Reinforcement learning falls between these two methods as the network is only told whether a response to a given stimulus is "good" or "bad". During training, the connection strengths, or "weights" between layers are adapted so that selected output neurons produce a value, or "fire", in response to a given stimulus. The network is effectively taught to recognise a series of input values. The principles of the neural network are discussed in more detail in other works [10,11,12].

Neural Computing as an Aid to Phase Unwrapping

Although digital computers are fast when complex arithmetic calculations are required, they are notoriously inadequate for such tasks as pattern recognition when compared to the human brain. Numerous works have dealt with the use of neural networks for pattern recognition, e.g. [13,14,15]. If the need to detect wraps in a phase distribution can be considered a problem of recognition, this leads to an interesting question: Can a neural network be "taught" to recognise a phase wrap? The idea of using a neural network for the processing of phase data has been explored by Takeda [16], who used a Hopfield recurrent neural network [17]. Little information is available regarding the implementation of this method, but the system uses a network which consists of a large number of neurons with correspondingly large numbers of training data and high training times. The approach described in this paper is simpler in that instead of using a large network to process an entire image,

a much smaller network is used to process sections of the phase map, effectively building up an image from a large number of smaller sets of inputs.

EXPERIMENTAL WORK

All experimentation was carried out using the neural network package "NeuralWorks Professional II Plus" [18]. This was implemented on a standard 486 DX2/66 personal computer. A 6x1 "window" of pixels was convolved with the original image, each pixel being assigned to one input neuron. The network consisted of six input, six hidden and six output neurons. This configuration was the minimum number of input and output neurons, an output neuron being trained to fire when a wrap was present at the corresponding input neuron. This configuration was the smallest possible before the system began behaving like a conventional point-to-point unwrapping algorithm. A diagram of the network is shown in Figure 2.

The network was trained using both real and simulated data, which were grouped into training "sets". These were ASCII files which contained the phase values of a number of different fringe patterns. The simulated data were noise free and showed a maximum of one wrap per training set. It is unlikely that, in a genuine wrapped phase distribution, two wraps will be encountered whose separation is less than six pixels. The real data consisted of phase values calculated by the FFT method from fringe patterns projected onto a flat, white surface. Each training set consisted of one hundred groups of data, or "training vectors", which were randomly presented to the network twenty thousand times. Experiments were conducted using both supervised and unsupervised learning. Kohonen [19] and Hopfield networks were used for unsupervised learning. To ascertain the viability of using supervised learning, a backpropagation network was used which employed a normalised-cumulative-delta learning rule and sigmoid transfer function. This configuration was used as previous experience has shown it to be the most suitable for this type of application. At each presentation, the Root Mean Square (RMS) error between the desired and actual output was calculated by the package. When the RMS error reached its *convergence criterion*, which is an effective zero set by the user (in this case 0.01), the training was recognised as complete. To test the network, different data from the training data were presented to the network and the outputs from these data were compared with

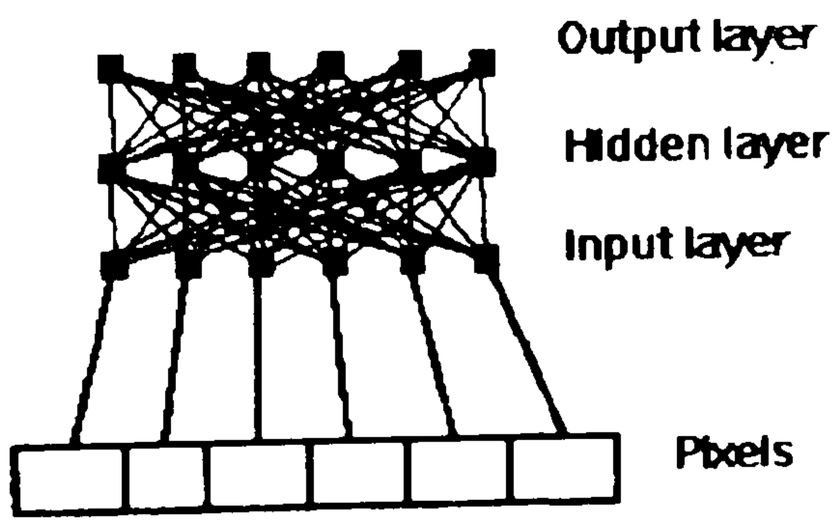


Figure 2 Configuration of the six-input phase unwrapping neural network.

the desired output. The test data were taken from fringe patterns projected onto the curved surface of a lens. When training and testing were complete, the NeuralWorks package converted the network into "C" code. The final output from the package was a C program which behaved with the same characteristics as the fully trained network. To perform the complete unwrapping operation, the C code from the package was incorporated into a program which read values of pixels, presented them to the network code and performed the necessary phase adjustments.

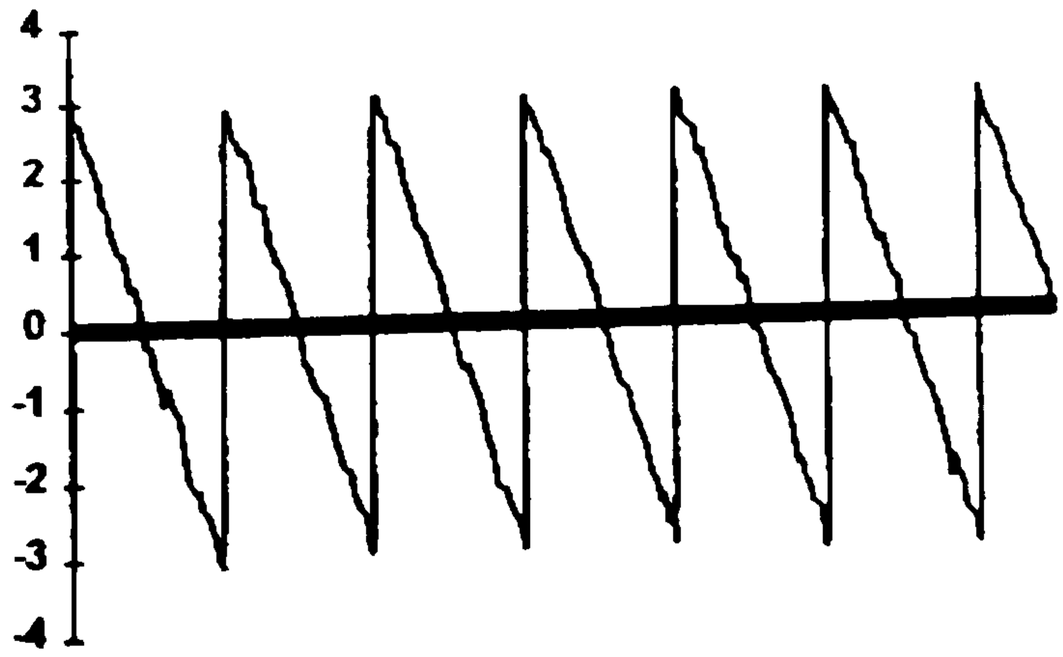
EXPERIMENTAL RESULTS

Unsupervised Learning

Training continued to 20,000 presentations when this method was used. Testing showed that wraps were correctly identified on approximately 50% of occasions. Due to this low success rate, a suitable unwrapping system could not be designed using this type of network.

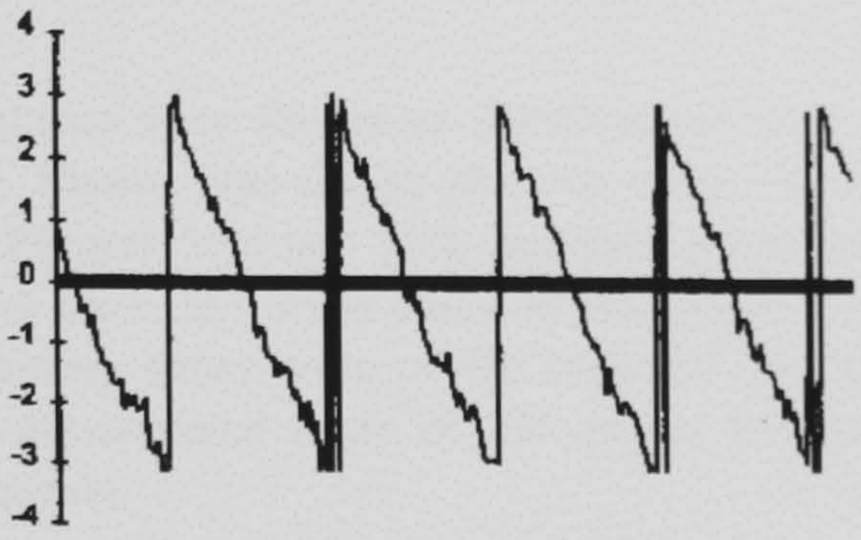
Supervised Learning

After approximately 8,500 random presentations of data, the network's convergence criterion was reached and training ceased. This was the case for both real and simulated data. Testing showed that phase wraps could be correctly identified on approximately 95% of occasions when training had taken place with real data and on approximately 80% of occasions with simulated data. Any inaccuracies involved the firing of an output neuron adjacent to the desired output neuron. Figure 3 shows three wrapped phase distributions. Figure 3(a) is noise-free and shows six wraps, Figure 3(b) contains four wraps and one single noise spike and Figure 3(c) shows five wraps and a number of noise spikes. Figure 4 shows the phase distributions unwrapped by Schafer's algorithm. In Figure 4(a), the distribution is unwrapped successfully, but Figures 4(b) and 4(c) show how the introduction of noise makes the final result unacceptable as the algorithm treats the noise spikes as wraps. In attempting to unwrap the noise, the errors are propagated throughout the final unwrapped phase



(a) Noise-free wrapped phase distribution.

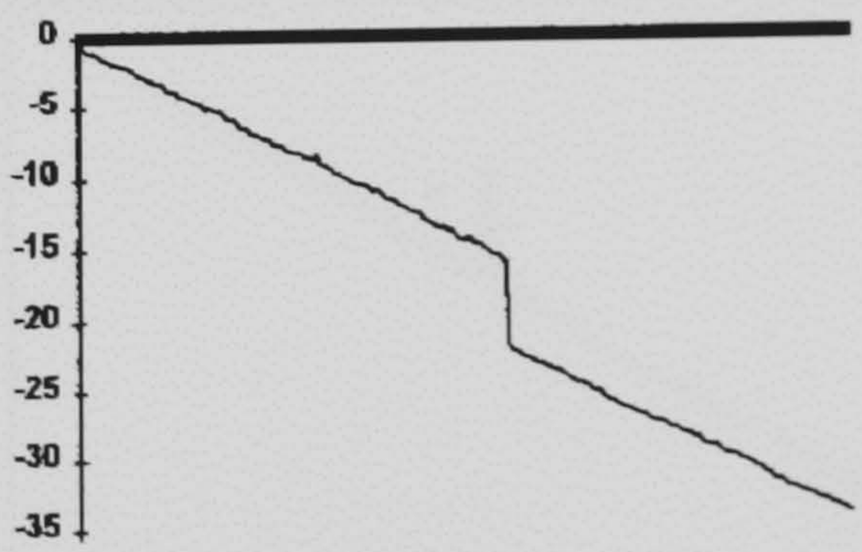
(b) Wrapped phase distribution; single noise spike.



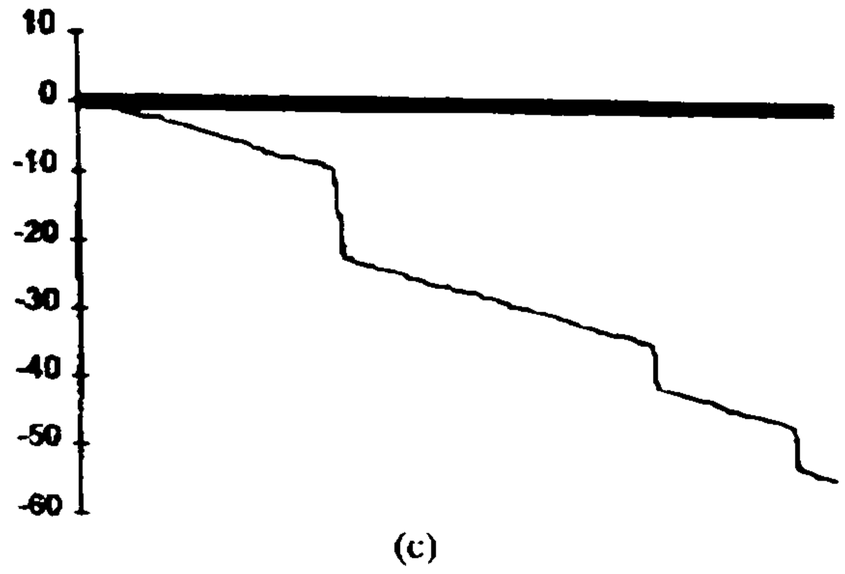
(c) Noisy wrapped phase distribution.

Figure 3 Wrapped phase distributions.

(a)



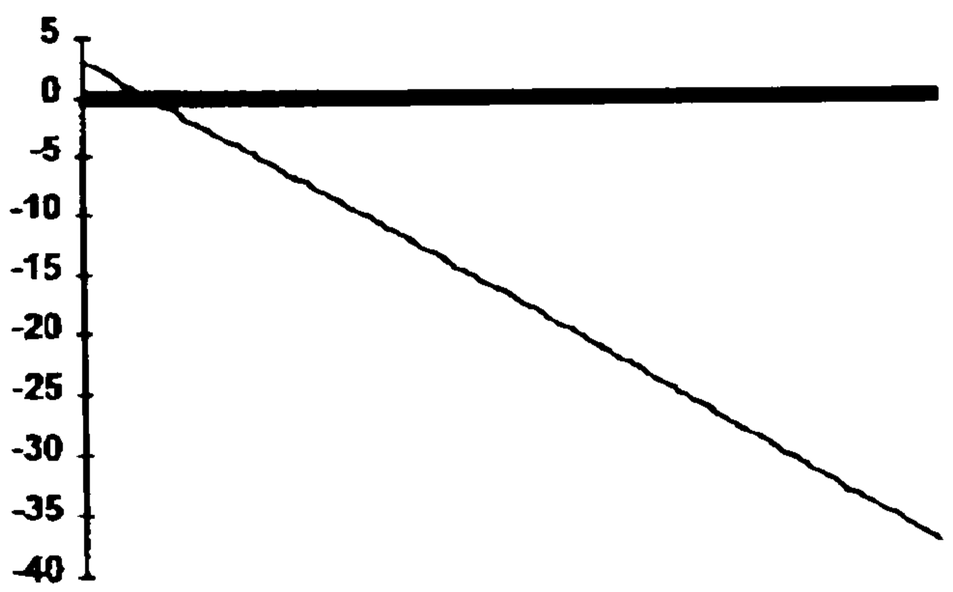
(b)



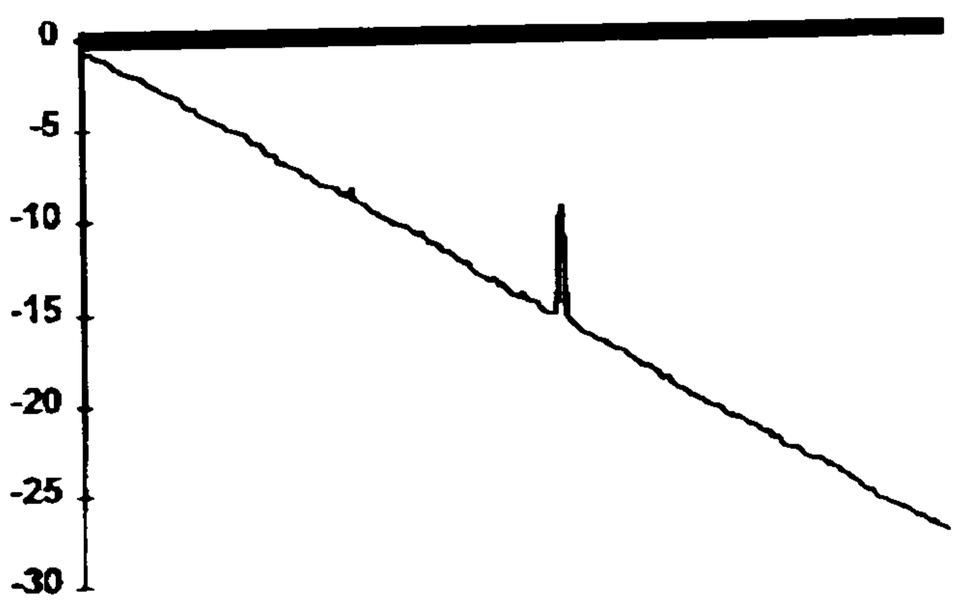
(c)

Figure 4 Phase distributions unwrapped using schafer's algorithm.

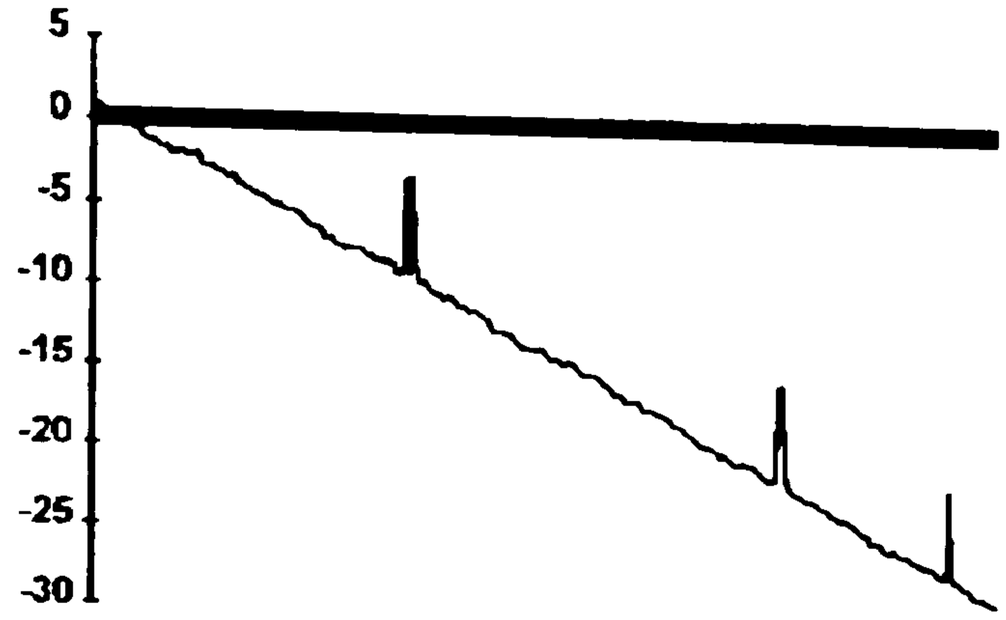
distribution. Figure 5 shows how the same distributions were unwrapped using the neural network method. Figure 5(a) shows that the same result is achieved with the noise-free distribution. Figures 5(b) and 5(c), however show the benefit of the neural network method. In each case, the noise spike is effectively ignored and unwrapping continues correctly. Training times were of the order of two hours and, with a fully trained network, a one-dimensional array of 256 pixels was successfully unwrapped in approximately 0.5 seconds.



(a)



(b)



(c)

Figure 5 Phase distributions unwrapped using the neural network method.

FURTHER WORK

An image is more than a series of lines. In a line, a pixel has only two immediate neighbours. In a real image, however, a pixel borders *four* others, meaning a network must be able to analyse twice the number of neighbouring pixels. Investigation into expansion of the wrap detector to a square "tile" is currently being carried out. This will need to take into account the relative vertical and horizontal positions of neighbouring pixels in a 2-D phase map. If a square "window" of pixels is convolved with the wrapped phase map, an analysis can be built up, a small portion at a time.

CONCLUSIONS

This paper has shown that an artificial neural network using supervised learning techniques can be successfully trained to identify the occurrence of wraps in a wrapped phase distribution. With a fully trained back-propagation network, one-dimensional wrapped phase distributions can be successfully unwrapped in approximately 0.5s. Best results were achieved when the networks were trained using data taken from real phase distributions. By combining the network with conventional code, one-dimensional phase distributions can be successfully unwrapped. Because of the success of the 1-D unwrapping experiments, it is felt that the potential exists to extend the concept further to unwrap complete two-dimensional wrapped phase distributions.

References

1. D. R. Burton and M. J. Lalor, "The precision measurement of engineering form by computer analysis of optically generated contours", Proc SPIE **1010**, Hamburg (1988).
2. P. Hariharan, "Basics of Interferometry", Academic Press (1992).
3. J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of Computation*, **9**, 297–301 (1965).
4. M. Takeda, H. Ina and H. Kobayashi, "Fourier transform method of fringe-pattern analysis for computer based topography and interferometry.", *J. Opt. Soc. Am.*, 72, 1 (1982).

5. R. W. Schafer and A. V. Oppenheim, "Digital Signal Processing", Prentice-Hall (1975).
6. J. Huntley, "Noise-immune phase unwrapping algorithm", *Applied Optics*, **28**, 15 (1989).
7. J. Gierloff, "Phase unwrapping by regions", *Proc. SPIE*, **8**, 2–9 (1987).
8. D. C. Ghiglia, G. A. Mastin and L. A. Romero, "Cellular automata method for phase unwrapping", *J. Opt Soc. Am.*, **4**, 1 (1987).
9. W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity.", *Bulletin of Mathematical Biophysics*, **5**, 115 (1943).
10. G. E. Hinton and J. A. Anderson, "Parallel Models of Associative Memory", Erlbaum 1981.
11. J. L. McClelland and D. E. Rumelhart, "Distributed memory and the representation of general and specific information" *J. Exp. Psy. (Gen).*, **114**, 159–188 (1985).
12. J. M. Zurada, "Introduction to Artificial Neural Systems", West, 1992.
13. J. Y. Shen, Y. X. Zhang and G. G. Mu, "Optical pattern recognition system based on a winner-take-all model of a neural network", *Optical Engineering*, **32**, 5, 1053 1056 (1993).
14. I. Guyon, P. Albrecht, Y. LeCun, J. Denker and W. Hubbard, "Design of a neural network character recognizer for a touch terminal", *Pattern Recognition*, **24**, 2, 105–119 (1991).
15. J. Loncelle, N. Derycke and F. F. Soulie, "Co-operation of GBP and LVQ networks for optical character recognition", *Int. Joint Conf. on Neural Networks*, 3, 694–699 (1992).
16. M. Takeda, "Phase unwrapping by neural network", Proc. FRINGE '93, Akademie Verlag, 1993.
17. J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proc. Nat. Acad. Sci. USA.*, **79**, 8 (1982).
18. NeuralWorks Professional II Plus Reference Guide. NeuralWare, Inc., Pittsburgh, USA, 1991.
19. T. Kohonen, "Associative Memory, a System Theoretical Approach.", Springer Verlag, 1977.

Fringe pattern optimisation using a backpropagation neural network.

D J Tipper, D R Burton & M J Lalor

Liverpool John Moores University, School of Engineering and Technology Management, Byrom Street, Liverpool, L3 3AF, UK.

Abstract. This paper describes a system for the optimisation of fringe patterns for surface measurement. The fringe patterns are generated using a twin-fibre adaptive interferometer. A backpropagation neural network is used to analyse values for the mean intensity, number of fringes, visibility and angle of fringe tilt of the image and make a decision as to whether the pattern is suitable for measurement according to these parameters. The output of the network is then fed back to the interferometer and the fibre position adjusted until an optimum fringe pattern is produced.

1. Introduction

1.1 Adaptive interferometry & Fringe Optimisation

Fringe optimisation can be defined as the selection of the best fringe pattern for any given surface. Consider a conventional fringe analysis system. Straight fringes are projected onto an object's surface, the image is viewed through a CCD camera and that image is analysed by a digital computer. If the object has a complex or irregular surface, the reflected fringes can be extremely difficult to measure accurately. The technology now exists, however, to create "adaptive" fringes. Using a twin-fibre adaptive interferometer or an LCD projection panel, it is possible to have control over both the spacing and the angle of tilt of the projected fringes. The use of this type of equipment would make it possible to configure a system in which the projected fringes can be adapted to suit the shape of the object to be measured. In an adaptive interferometry system, a set of fringes is projected onto a surface, the image is viewed through a CCD camera and before any measurement is carried out, the fringe pattern is *optimised* . The computer essentially decides whether the fringe pattern it sees will be suitable to measure the surface in question and if not the pattern is changed by the adaptive interferometer until it is suitable for measurement.

1.2 Artificial Neural Networks

The Artificial Neural Network (ANN) is a computing paradigm which has no algorithm as such, but "learns through experience". Originally developed from an idea for a mathematical model of a biological neuron [1], the ANN consists of a number of processing elements, or "neurons", each one acting as a simple thresholding device. When a large number of neurons are connected together, a large number of simple

calculations can be combined to produce a more complex overall result. Each network consists of a layer of input neurons and a layer of output neurons, which are linked by an intermediate layer of "hidden" neurons, so called because their input and output values are not seen by the user.

In order for an ANN to function correctly, it must first be "trained". There are three types of leaning normally associated with ANNs: supervised, unsupervised and reinforcement learning. With supervised learning, a set of input stimuli and corresponding output stimuli are presented to the network, so that it learns the correct response to each possible input. Unsupervised leaning involves presentation of only the input stimulus, so the ANN is left to calculate its own output. Reinforcement learning falls somewhere between these two categories as the network is only taught whether a particular input is "good" or "bad". The networks described in this paper make use of supervised learning. To train an ANN using supervised learning, a set of input values is applied to the input layer and corresponding output values to the output layer. The connection strengths, or "weights", between the layers are adapted so that selected output neurons will produce an output, or "fire", in response to a given stimulus. If training is successful, the network should be able to interpolate an output state from input data that is not a member of the training set.

The principles of the ANN are discussed in more detail in other works [2]

Much research has been carried out into the use of ANNs for pattern recognition problems, e.g.[3]. If fringe optimisation is considered as a pattern recognition problem, it follows that it may be possible to "teach" an ANN to recognise the best possible fringe pattern for measurement of a given surface.

2. Experimental Procedure

The problem originally encountered was how to define the quality of a fringe pattern. Parameters which could easily qualitatively define a fringe image were decided to be:

* Intensity
* Number of fringes
* Tilt
* Contrast/visibility

These parameters could be calculated from intensity values at key points in a given fringe image.

In order to provide suitable training data for the network, intensity values were taken from five regions in each image. Each image was divided into four equal regions. Intensity profiles were recorded for each of the four regions plus a fifth profile was recorded in the centre of the image where it was brightest. From these arrays of intensity values, it was possible to calculate mean intensity, number of fringes and visibility for each region. Visibility was calculated according to the equation

$$V = \frac{I_{max} - I_{min}}{I_{max} + I_{min}}$$

In order to determine the direction of tilt of the fringe pattern, five intensity profiles were also recorded as shown in figure 1.

{import a:\\ffig1.bmp}

Figure 1: Directions of intensity profiles for tilt calculation

The ANN used for this application was configured using the NeuralWorks Professional II Plus package running on a standard 486 DX2/66 stand-alone PC. A Backpropagation network using a sigmoid transfer function and Normalised-Cumulative-Delta learning rule was used whose architecture consisted of 20 input, 10 hidden and 4 output neurons. Each parameter was assigned five input neurons and one output neuron. The training set and test set each contained 120 training vectors, the training set being presented randomly to the network 50,000 times. When training was complete the test set was presented to the network and its results recorded.

When the network was fully trained and tested, it was converted into "C" code by the NeuralWorks package. The final output from the package was a C program which behaved with the same characteristics as the trained network. To perform the complete optimisation operation, a series of values were read from a fringe image, presented to the network code and the output incorporated into code to drive the interferometer.

3. Experimental Results

The results given by the network for the test set were compared with the desired results and number of matches were noted. This was converted to a percentage to give a value for the accuracy of the network. These values are given in table 1.

Table 1:

Parameter	% accuracy
Mean intensity	97
Fringe number	86
Tilt	97
Visibility	95
TOTAL	94

4. Conclusions

The experimental work has shown that backpropagation ANNs are suitable for determining the quality of fringe patterns for adaptive interferometry applications.

5. References

[1] McCulloch W S and Pitts W 1943 *Bulletin of Mathematical Biophysics* **5** 115

[2] Hinton G E and Anderson J A 1981 *Parallel Models of Associative Memory* (Erlbaum)
McClelland J L and Rumelhart D E 1985 *J. Exp. Psy.* (Gen) **114** 159-188
Zurada J M 1992 *Introduction to Artificial Neural Systems* (West)

[3] Shen J Y, Zhang Y X and Mu G G 1993 *Optical Engineering* **32**, 1053-1056
Guyon I, Albrecht P, LeCun Y, Denker J and Hubbard W 1991 *Pattern Recognition* **24** 105-119
Loncelle J, Derycke N and Soulie J J 1992 *Int. Joint Conf. on Neural Networks* **3** 694-699