MODELING AND ANALYSIS OF A WAVELET NETWORK BASED OPTICAL SENSOR FOR VIBRATION MONITORING

A THESIS IN MECHATRONICS

Presented to the faculty of the American University of Sharjah
School of Engineering
in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by
YASMINE AHMED EL-ASHI
B.S. 2006

Sharjah, UAE

We approve the thesis of Yasmine Ahmed El-Ashi

Date of signature

_____ _____

Dr. Rached Dhaouadi
Associate Professor, Electrical Engineering, AUS
Thesis Advisor

_____ _____

Dr. Taha Landolsi
Assistant Professor, Computer Engineering, AUS
Thesis Co-Advisor

_____ _____

Dr. Khaled Assaleh
Associate Professor, Electrical Engineering, AUS
Graduate Committee

_____ _____

Dr. Ameen El-Sinawi
Associate Professor, Mechanical Engineering, AUS
Graduate Committee

_____ _____

Dr. Rached Dhaouadi
Associate Professor,
Coordinator, Mechatronics Engineering Graduate Program

_____ _____

Dr. Hany El-Kadi
Associate Dean, College of Engineering

_____ _____

Dr. Yousef Al Assaf
Dean, College of Engineering

_____ _____

Mr. Kevin Lewis Mitchell
Director, Graduate & Undergraduate Programs & Research

MODELING AND ANALYSIS OF A WAVELET NETWORK BASED OPTICAL SENSOR FOR VIBRATION MONITORING

Yasmine Ahmed El-Ashi, Candidate for Master of Science in Mechatronics Engineering

American University of Sharjah, 2010

ABSTRACT

The main objective of this research is to present a wavelet network based optical lateral position sensor for vibration monitoring using a 2×2 photodetector array. In our approach, the power distribution of the light spot is measured taking into consideration the radial symmetry of the spot and its Gaussian intensity profile. The proposed system uses a He-Ne laser source whose Gaussian beam impinges on the photodetector array. The normalized optical power for each photocell is obtained theoretically by deriving the optical power equations as the beam scans the plane of photodetectors. The position detection is based on finding a relationship between the power distribution of the photodetector array and the position of the beam center. An experimental setup of the system is developed to validate the theoretical results. Furthermore, a wavelet network function approximation technique is used to estimate the x–y position of the beam center corresponding to the measured optical powers.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I thank Allah the Most Gracious, the Most Merciful, for giving me the will and power to complete my thesis research, and allowing me to pass through such an experience where not only do you acquire the academic skills of research, but you also learn other qualities such as patience and perseverance.

Next, I would like to express my sincere gratitude to my research advisors, Dr. Taha Landolsi and Dr. Rached Dhaouadi, to whom I owe a lot, for their patience, guidance and encouragement throughout the different stages of the research. I highly appreciate their vision, novel ideas, and their excitement at every big or small new finding made. I hope they accept my apologies for any inconveniences or disappointments that I might have caused.

I am also particularly grateful to my parents for their continuous support. Finally, it is with pleasure that I express my appreciation to all my friends and colleagues who supported me and aided me in both the good and hard times.

1

Introduction

There are numerous applications which require accurate, noncontact position measurements, such as vibration monitoring as well as vibration measurement systems with frequencies ranging from fractions of Hz to kHz and amplitudes varying between nanometers to meters [1].

Natural vibration is a manifestation of the oscillatory behavior in physical systems, as a result of a repetitive interchange of two types of energy, such as kinetic and potential energies among components in mechanical systems [2]. Additional factors responsible for mechanical vibrations in machines, components and systems, involve unbalanced inertia, bearing failure in rotating systems, poor kinematic design resulting in a non-rigid and non-isolating structure, component failure, and operation outside prescribed load ratings [3]. Such types of vibration are usually categorized as undesirable or harmful vibration, which also include structural motions generated due to earthquakes, noise generated by construction equipment, and dynamic interactions between vehicles and bridges or guide ways. The elimination or suppression of such

undesirable vibrations will result in a reduction in noise levels and improved work environment, maintenance of high performance standards and production efficiency, as well as prolonging the useful life of industrial machinery, thus cutting down the costs and frequency of maintenance. On the other hand, there are useful forms of vibration which include those generated by devices used in physical therapy and medical applications, vibrators used in industrial mixers, part feeders and sorters, and vibratory material removers such as drills and finishers. For instance, product alignment for industrial processing or grading can be carried out by means of vibratory conveyers or shakers [2].

Over the past 50 years, the speeds of operation of machinery have doubled, and consequently vibration loads generated due to rotational excitation would have quadrupled if proper actions of design and control were not considered. As vibration isolation and reduction techniques have become an integral part of machine design the need for accurate measurement and analysis of mechanical vibration has grown significantly [4]. To accomplish this, we should undergo a phase of monitoring and diagnostic testing of vibration which would require devices such as sensors and transducers, signal conditioning and modification hardware (such as filters, amplifiers, analog/digital conversion means), and actuators (such as vibration exciters or shakers) [2].

Vibrations have been mainly detected by contact and noncontact-type sensors for measuring displacement, velocity or acceleration [4]. Conventional vibration sensors such as potentiometers or linear variable differential transformers (LVDTs), piezoelectric accelerometers, and strain gauges are common in practice. Contact-type sensors such as potentiometers and strain gauges are used to sense displacement either by making physical contact with or attached to the object of interest. However, in some cases, physical contact may not be practical, in terms of impeding or altering the natural behavior of the device, or inability to install them in hard-to-reach places, or when the object is fragile and prone to damage [5]. Moreover, potentiometers have the following limitations: (1) High frequency or highly transient measurements are not feasible because of factors such as slider bounce, friction and inertia resistance, and induced voltages in the wiper arm and primary coil. (2) Resolution is limited by the number of turns in the coil and by the coil uniformity. (3) Wear out and heating

up with associated oxidation in the coil and slider contact cause accelerated degradation [2]. Another extensively used vibration sensor is the piezoelectric accelerometer, an electromechanical device where its output voltage is proportional to an applied vibratory acceleration. Although it is light in weight, exhibits a high frequency response (up to about 1 MHz), accurate and sensitive, piezoelectric transducers cover a relatively small area, and are difficult to electrically isolate making them unsuitable in applications surrounded by electrical and magnetic fields [6].

In situations where physical contact between the sensor and the device is inaccessible and undesirable, non-contact type sensors provide a better option. These sensors operate on capacitive, inductive, magnetic or optical principles. For instance, capacitive sensors have very high resolution (d <0.01 nm), however they are sensitive to changes in temperature, humidity and surface irregularities. Inductive sensors measure displacement by current induction when a ferrous or nonferrous metallic object passes through the electromagnetic field of a coil wound. Such sensors also have relatively high-resolution (nanometer) and good bandwidth (tens of kilohertz), with the added advantage of being immune to dirt, water, and lubricating oil. Capacitive and inductive sensors are generally expensive and require special signal processing circuitry for operation [5].

In the recent years, optical non-contact position sensors have received great attention owing to their immunity to electromagnetic interference, resistance to corrosion, chemical inertness, and light weight. Such sensors include Fabry-Perot interferometers, fiber Bragg grating (FBG) arrays, Michelson interferometers, and Mach-Zehnder interferometers (MZI) which are used for measuring mechanical vibrations at magnetic cores of power transformers [6]. Fabry-Perot interferometers are used extensively as versatile tools for fast and sensitive vibration analysis in harsh engineering environments. The sensitivity of these sensors is increased by increasing the length of the sensing area; however, such an approach causes the sensor to be affected by fluctuations. Not only do these sensors suffer from limitations in signal demodulation caused by phase ambiguity, but the external disturbance aforementioned can also be observed as a phase drift that is often compensated in order to measure the dynamic parameters of the system [7],[8]. Furthermore, Bragg gratings have a relatively poor resolution, and are difficult to be located and installed in the structure

[7]. The conventional Michelson interferometer-based laser vibrometer suffer from two main drawbacks; limited sensitivity to surface displacement detection and their intolerance to the presence of optical speckles in the light beams. An abrupt change in the speckles can lead to a sudden degradation in the optical power reaching the photodetector, and a misinterpretation of the diminished output [9].

Nevertheless, reflective optical proximity sensors offer comparable performance to their inductive and capacitive counterparts in terms of resolution and bandwidth. In an optical sensor, a source impinges light onto a target object, and subsequently reflects off the object's surface, which is then projected onto a detector. The sensed intensity of the light reflected onto the photodiode is related to an object's distance from the photodetector. Optical sensors are also relatively inexpensive, unlike the capacitive and inductive sensors [5].

One feature common to all of the previously mentioned non-contact sensors is that they are capable of measuring displacement in the direction of the optical axis of the system. To measure lateral displacement, that is perpendicular to the optical axis a position-sensitive detector (PSD) is usually used. Makynen, Kostamovaara, and Myllyla presented in [1] a lateral displacement sensing method based on the idea of imaging an illuminated cooperative target on a four-quadrant (4Q) PSD. This arrangement has the advantage of being capable of providing true lateral displacement instead of angular displacement in large working volume without calibration. This is possible due to the unique property of a target-focused 4Q detector, in which the size of the measurement span is determined solely by the size of the cooperative target, thus providing inherently accurate, constant scaling that is independent of the target distance. The 4Q detector consists of four photodiodes (quadrants) positioned symmetrically around the center of the sensor and separated by a narrow gap. The position information is derived from the signals received by the quadrants as the image spot moves over the detector surface.

Other analog position-sensitive detectors that have been proposed by Makynen, Kostamovaara, and Myllyla in [10] are the lateral-effect photodiode (LEP) detectors. An LEP is a large-area, single-element photodiode having uniform resistive layers with two wide edge contacts on both the anode and cathode. The current carriers generated in the illuminated region are divided between the electrodes in

proportion to the distance of the current paths between the illuminated region and the electrodes. The measurement field of the LEP is determined by the size of its active area and it detects the spot position irrespective of spot size or shape. The achievable SNR and the resolution of the 4Q detector is better than that of the LEP. However, LEP provides far better accuracy in a typical outdoor environment because atmospheric turbulence induced, spatially uncorrelated intensity fluctuations within the light spot which result from defocusing, derange the measurement resolution of the 4Q receiver [10].

Furthermore, one of the established non-contact lateral position sensing techniques involve the use of a charge-coupled device (CCD) camera. The CCD sensor records the light intensity in each pixel by means of charge coupling where the charges are transferred to a second bank of photosites before analog-to-digital conversion is made. In this case, position measurment is usually performed by calculating the center of gravity of the light distribution [11].

The quadcell array for lateral, two dimensional position measurement consists of square shaped and homogeneous photodetectors (PDs), clustered in a 2×2 configuration [12]. The lateral dimensions of standard discrete commercial PSDs extend up to several millimeters. Using a quadcell array of photodetectors involves several advantages, such as, large position measurement area, reduced number of direct output signals, acceptance of a wide range of spot intensity profiles and radii, negligible spatial fluctuation of the signal, immunity to coordinate crosstalk, and possible operation with modulated or pulsed light [12].

The transfer characteristics of the photodetector depend on the shape and intensity distribution of the beam spot. In [1], the authors used a perfectly linear transfer function by assuming a square light spot with uniform intensity distribution and with its edges parallel to the edges of the quadrants. However, in most practical conditions, the spot intensity profile exhibits radial symmetry and the resulting response is non-linear [12].

The purpose of this research is to present a wavelet network based, non-contact optical position sensor using a photodetector array, which measures the power distribution of the light spot, taking into consideration the nonlinearities involved, that is the radial symmetry of the spot and its Gaussian intensity profile.

Since only the optical power of each photodetector can be practically acquired, we aim to find a relationship between the power distribution of the photodetector array and the position of the spot center, through a theoretical and experimental model. In our approach, we account for the nonlinear transfer characteristics, by using a wavelet network as a function approximation technique to estimate the x–y position of the light spot center, that corresponds to the acquired optical powers. Therefore, in order to achieve a more accurate system model and to depict a better comparison between the theoretical results and experimental measurements, we take into consideration the circular shape and Gaussian intensity profile of the light spot in formulating the optical power equations. These equations will be further used, along with a developed algorithm, to simulate the theoretical model of the proposed position detection system. In addition, system imperfections such as the gap separations between the photodetectors, have been accounted for in the simulation and their effect on the optical power distribution is studied. A potential application of our proposed system will be on vibration monitoring, where the position information will be employed to obtain characteristics such as the amplitude, frequency and speed of vibration.

The rest of the thesis report is organized as follows; in Chapters 2 and 3, we give an overview of beam optics, wavelets and wavelet networks (WN) and their use in function approximation. Next, in Chapter 4, we describe the design of our proposed system, which involves the system architecture of the theoretical optical acquisition model of the position sensor. We also present the experimental setup and results used to validate the theoretical optical model. Finally, in Chapter 5, we discuss the results obtained after training and testing the WN.

2

Beam Optics

The optical signal emanating from the He-Ne laser source is commonly modeled as a Gaussian beam traveling in free-space whose intensity varies with the propagation distance z and the radius of the beam ρ, measured from its center. As the laser beam propagates, its power remains constant but its intensity decreases with an inverse-square law. This behavior is important to consider in the theoretical model of the proposed system as well as the design of the experimental setup because the power intercepted by the photodetector depends on the area of the detector active surface. In this Chapter, we will discuss the propagation of light in free-space that would lead us to the derivation of Gaussian beam optics, intensity and power characteristics.

2.1 The Wave Equation

Light propagates in the form of waves. In free space, light waves travel with speed c_o. A homogeneous transparent medium such as glass is characterized by a single constant, its refractive index ($n \geq 1$). In a medium of refractive index n, light waves

travel with a reduced speed.

$$c = \frac{c_o}{n}. \tag{2.1}$$

An optical wave is described mathematically by a real function of position $\mathbf{r} = (x, y, z)$ and time t, denoted by $u(\mathbf{r}, t)$ and known as the wavefunction. It satisfies the wave equation,

$$\nabla^2 u - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = 0, \tag{2.2}$$

where ∇^2 is the Laplacian operator, $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2$. Any function satisfying (2.2) represents a possible optical wave. Because the wave equation is linear, the principle of superposition applies, for instance if $u_1(\mathbf{r}, t)$ and $u_2(\mathbf{r}, t)$ represent optical waves, then $u(\mathbf{r}, t) = u_1(\mathbf{r}, t) + u_2(\mathbf{r}, t)$ also represents a possible optical wave [13].

When electric dipoles are forced to oscillate, they induce an electric field that oscillates at the same frequency. In addition, due to the motion of oscillating charges, a magnetic field oscillating at the same frequency is also induced. These simultaneous oscillating fields are the basis for all known modes of electromagnetic radiation. Thus, X-rays, UV radiation, visible light, and infrared and microwave radiation are all part of the same physical phenomenon. Although each radiating mode is significantly different from the others, all modes of electromagnetic radiation can be described by the same equations, since they all obey the same basic laws.

Oscillation alone is insufficient to account for electromagnetic radiation. The other important observation is that radiation propagates. It is emitted by a source and if uninterrupted, can propagate indefinitely in both time and space. Although there are certain media that can block radiation, we find it more astonishing that electromagnetic waves can propagate through free space; unlike electrical currents or sound, conductors are not necessary for the transmission of radiation. Although this property is unique to radiation, some of its characteristics is analogous to the propagation of acoustical waves or vibrations in solids. These waves, like the electromagnetic waves combine propagation with the oscillation of a physical parameter. Thus, by analogy the description of the propagation of electromagnetic radiation should involve equations similar to those describing the propagation of sound waves or the vibrational modes of solids. Furthermore, since we anticipate that electromagnetic

waves are the result of oscillatory motion of electric charges, we should be able to derive equations describing their propagation from Maxwell's equations [14].

First, to demonstrate the analogy between electromagnetic waves propagation and that of acoustic waves, we shall derive the wave equation for the case of just one spatial variable, for the physical system represented by a vibrating string. Consider a perfectly flexible homogeneous string stretched to a uniform tension τ between two points. Let $u(x,t)$ be the displacement of the string from its horizontal position. The

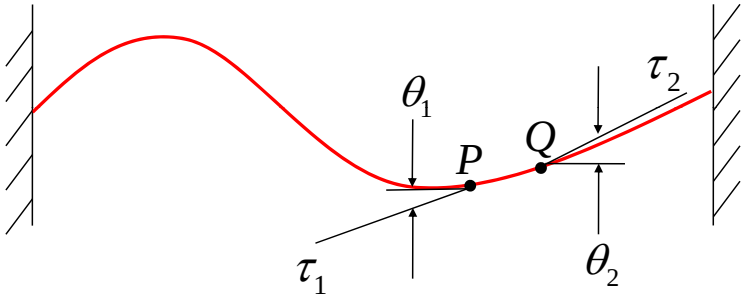

Figure 2.1: A vibrating string at an instant of time, the quantities shown are used in the derivation of the classical one-dimensional Wave equation [15].

quantities τ_1 and τ_2 are the tensions at the points P and Q on the string. Both τ_1 and τ_2 are tangential to the curve of the string. Assuming that there is only vertical motion of the string, the horizontal components of the tensions at the points along the string must be equal. Using the notation provided in Figure 2.1, we have the following relation,

$$\tau_1 \cos\theta_1 = \tau_2 \cos\theta_2 = \tau = \text{constant}. \tag{2.3}$$

There is a net vertical force that causes the vertical motion of the string, which we find to be,

$$F_{net} = \tau_2 \sin\theta_2 - \tau_1 \sin\theta_1. \tag{2.4}$$

By Newton's second law ($F = ma$), this net force is equal to the mass $\rho\Delta x$ along the segment PQ times the acceleration of the string, $\partial^2 u/\partial t^2$. Thus, we can state the following:

$$\tau_2 \sin\theta_2 - \tau_1 \sin\theta_1 = \rho\Delta x \frac{\partial^2 u}{\partial t^2}. \tag{2.5}$$

Dividing equation (2.5) by equation (2.3) gives:

$$\tan\theta_2 - \tan\theta_1 = \frac{\rho\Delta x}{\tau}\frac{\partial^2 u}{\partial t^2}. \tag{2.6}$$

Since $\tan\theta_1$ and $\tan\theta_2$ are the slopes of the curve of the string at x and $x+\Delta x$, respectively, they can be written as, $\tan\theta_1 = \frac{\partial u}{\partial x} = u_x(x)$ and $\tan\theta_2 = \frac{\partial u}{\partial x} = u_x(x+\Delta x)$, where u_x denotes the partial derivative of u with respect to x. Substituting the values for $\tan\theta_1$ and $\tan\theta_2$ into equation (2.6) yields:

$$u_x(x+\Delta x) - u_x(x) = \frac{\rho\Delta x}{\tau}\frac{\partial^2 u}{\partial t^2}. \tag{2.7}$$

Dividing equation (2.7), by Δx and setting the limit $\Delta x \to 0$,

$$\lim_{\Delta x\to 0}\frac{u_x(x+\Delta x) - u_x(x)}{\Delta x} = \frac{\rho}{\tau}\frac{\partial^2 u}{\partial t^2},$$

$$\frac{\partial u_x}{\partial x} = \frac{\rho}{\tau}\frac{\partial^2 u}{\partial t^2},$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{\rho}{\tau}\frac{\partial^2 u}{\partial t^2}. \tag{2.8}$$

And so equation (2.7) becomes,

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}, \tag{2.9}$$

in the limit $\Delta x \to 0$, where $v = (\tau/\rho)^{1/2}$ has units of speed [15]. Its extension to more spatial variables is given by:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2},$$

$$\nabla^2 u = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}. \tag{2.10}$$

Although, equations for the propagation of electromagnetic waves are likely to be similar to those for acoustic waves, there is an important distinction between

the two. Acoustic wave equations describe the propagation of a scalar quantity; electromagnetic wave equations describe the propagation of electric and magnetic fields, which are vectorial.

In order to derive the equations that describe the propagation of electromagnetic waves, we begin with Maxwell's equations:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} + [0]\,, \tag{2.11}$$

$$\nabla \times \mathbf{H} = -\left[\frac{\partial \mathbf{D}}{\partial t}\right] + \mathbf{j}, \tag{2.12}$$

$$\nabla \cdot \mathbf{D} = \rho, \tag{2.13}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{2.14}$$

Where, $\mathbf{E}, \mathbf{D}, \mathbf{H}, \mathbf{B}, \mathbf{j}$ are respectively the electric field, electric displacement, magnetic field, magnetic induction and current density vectors. To demonstrate the symmetry between the effects of electricity and magnetism, for each equation describing the effects of the electric field there is a counterpart describing effects of the magnetic field. Even the electric charges fit into the symmetric picture, when a term for electric charge or electric current is present in one equation, a zero term is present in its magnetic counterpart, representing the absence of magnetic monopoles. Thus, the zero term in brackets in equation (2.11) is the magnetic analog to the current density $\mathbf{j}$ in equation (2.12). Similarly, the charge density term ρ in equation (2.13) is replaced by a zero in equation (2.14).

Maxwell's equations form the basis for the development of the equations that describe the propagation of electromagnetic waves. Historically, the electromagnetic wave equations were derived by Maxwell merely to describe the propagation of oscillating electric or magnetic fields in space. Neither Maxwell nor his peers recognized the relation between the propagation of electromagnetic fields and the propagation of light. Optics and the propagation of electromagnetic waves were at that time considered to be separate and unrelated fields of physics. Only after showing that the propagation velocity of electromagnetic waves was identical to the already measured speed of light, Maxwell suggested that his results might be more general than expected and hence applicable to the studies of optics.

Inspection of Maxwell's equations reveals that when the magnets or electric charges are static, the electric field vector in equation (2.13), does not contain any terms of the magnetic field, and conversely in equation (2.14) is independent of the electric field. When in motion however the magnets or electric charges induce fields that are interdependent. This is apparent in both equations (2.11) and (2.12), where $\mathbf{E}$ depends on the time derivative of $\mathbf{B}$, and where $\mathbf{H}$ varies with the magnitude or direction of the current flow or with the time derivative of $\mathbf{D}$. Nevertheless, an equation that describes the propagation of electric waves is expected to be independent of the terms that include the magnetic field, and vice versa. Since only two equations (2.11) and (2.12) describe the dynamic effect of these two fields, we will be using them as our initial point for deriving the equations describing the propagation of electric or magnetic waves. We first consider equation (2.11). The simplest way of eliminating the magnetic field term from this equation is by obtaining the curl of both sides:

$$\nabla \times \nabla \times \mathbf{E} = -\frac{\partial}{\partial t}\left(\nabla \times \mathbf{B}\right) = -\mu \frac{\partial}{\partial t}\left(\nabla \times \mathbf{H}\right). \tag{2.15}$$

Assuming that the magnetic permeability μ is constant, it was placed outside the derivative operators, thereby leaving only the magnetic field to be operated on. However, the term $\nabla \times \mathbf{H}$ in equation (2.15) can be replaced by the right-hand side of equation (2.12), thereby eliminating the magnetic field term. The following equation,

$$\begin{aligned} \nabla \times \nabla \times \mathbf{E} &= -\mu \frac{\partial}{\partial t}\left(-\left[\frac{\partial \mathbf{D}}{\partial t}\right] + \mathbf{j}\right) && (2.16) \\ &= \mu \frac{\partial^2 \mathbf{D}}{\partial t^2} - \mu \frac{\partial \mathbf{j}}{\partial t}, && (2.17) \end{aligned}$$

is now in the desired form; it contains only terms of electric field or electric charge. Furthermore, it includes both time and space derivations of these quantities and so describes both the temporal and spatial variation of the electric field due to the motion of electric charges. Although this equation is complete in itself, it can be further simplified by using the vector identity, $\nabla \times \nabla \times \mathbf{A} = \nabla\left(\nabla \cdot \mathbf{A}\right) - \nabla^2 \mathbf{A}$, where $\mathbf{A}$ is an arbitrary vector and $\nabla^2 = \nabla \cdot \nabla$ is the Laplacian operator. Thus in the Cartesian coordinate system, operating on any vector $\mathbf{A} = A_x \hat{e}_x + A_y \hat{e}_y + A_z \hat{e}_z$ with

the Laplacian yields,

$$\begin{aligned}\nabla^2\mathbf{A} &= \left(\frac{\partial^2 A_x}{\partial x^2}+\frac{\partial^2 A_x}{\partial y^2}+\frac{\partial^2 A_x}{\partial z^2}\right)\hat{e}_x+\left(\frac{\partial^2 A_y}{\partial x^2}+\frac{\partial^2 A_y}{\partial y^2}+\frac{\partial^2 A_y}{\partial z^2}\right)\hat{e}_y \\ &+ \left(\frac{\partial^2 A_z}{\partial x^2}+\frac{\partial^2 A_z}{\partial y^2}+\frac{\partial^2 A_z}{\partial z^2}\right)\hat{e}_z.\end{aligned}$$

Thus, the left hand-side of equation (2.17) can be replaced by:

$$\nabla\times\nabla\times\mathbf{E}=\nabla\left(\nabla\cdot\mathbf{E}\right)-\nabla^2\mathbf{E}. \tag{2.18}$$

However, when the medium in which $\mathbf{E}$ propagates is homogeneous (i.e., when all the spatial derivatives of the electric permeability, vanish), and when the medium does not contain any free charges (i.e., $\rho=0$), the first of these terms is $\nabla\cdot\mathbf{E}=0$. With the above vector identities, equation (2.17) can be reduced to:

$$\nabla^2\mathbf{E}=\mu\frac{\partial^2\left(\varepsilon\mathbf{E}\right)}{\partial t^2}+\mu\frac{\partial\mathbf{j}}{\partial t}. \tag{2.19}$$

This is the wave equation that describes the propagation of an electric wave. It does not specify what caused the field or how the field can be annihilated, but it accurately predicts the magnitude and direction of $\mathbf{E}$ at any point in space or time. Since most optical elements consist of uniform media, the assumption that ε is constant is always justified. The second assumption, that is, that the density of unbalanced electric charges is $\rho=0$, is met in free space and in all electrically neutral media, whether dielectric or conducting. Therefore, by replacing $\mathbf{E}$ with the optical wave $u\left(\mathbf{r},t\right)$ and setting $\mu=\mu_o$ to the magnetic permeability in free space, and $\varepsilon=\varepsilon_o$ to the electric permeability in free space, and $\frac{\partial\mathbf{j}}{\partial t}=0$, we end up with the following wave equation for an optical wave:

$$\nabla^2 u\left(\mathbf{r},t\right)=\mu_o\varepsilon_o\frac{\partial^2 u\left(\mathbf{r},t\right)}{\partial t^2}. \tag{2.20}$$

Since, the speed of light $c=\frac{1}{\sqrt{\mu_o\varepsilon_o}}$, we can finally state the wave equation for an optical signal propagating in free space, as follows [13]:

$$\nabla^2 u\left(\mathbf{r},t\right)=\frac{1}{c^2}\frac{\partial^2 u\left(\mathbf{r},t\right)}{\partial t^2}. \tag{2.21}$$

2.2 Monochromatic waves

A mononchromatic wave is represented by a wavefunction with harmonic time dependence,

$$u(\mathbf{r},t) = a(\mathbf{r})\cos\left[2\pi ft + \varphi(\mathbf{r})\right]. \tag{2.22}$$

Where, $a(\mathbf{r})$ = amplitude, $\varphi(\mathbf{r})$ = phase, f = frequency (cycles/s or Hz) and $\omega = 2\pi f$ = angular frequency (radians/s). Both the amplitude and the phase are generally position dependent, but the wavefunction is a harmonic function of time with frequency f at all positions [13].

2.2.1 Complex wavefunction

It is convenient to represent the real wavefunction $u(\mathbf{r},t)$ in equation (2.22) in terms of a complex function:

$$U(\mathbf{r},t) = a(\mathbf{r})\exp\left[j\varphi(\mathbf{r})\right]\exp\left(j2\pi ft\right), \tag{2.23}$$

such that,

$$u(\mathbf{r},t) = \mathrm{Re}\left\{U(\mathbf{r},t)\right\} = \frac{1}{2}\left[U(\mathbf{r},t) + U^*(\mathbf{r},t)\right]. \tag{2.24}$$

The function $U(\mathbf{r},t)$ also known as the complex wavefunction, completely describes the wave, and the wavefunction $u(\mathbf{r},t)$ is simply its real part. Similar to the wavefunction $u(\mathbf{r},t)$ the complex wavefunction $U(\mathbf{r},t)$ must also satisfy the wave equation:

$$\nabla^2 U - \frac{1}{c^2}\frac{\partial^2 U}{\partial t^2} = 0. \tag{2.25}$$

2.2.2 Complex Amplitude

Equation (2.25) can be written in the following form:

$$U(\mathbf{r},t) = U(\mathbf{r})\exp\left(j2\pi ft\right). \tag{2.26}$$

Where the time independent factor $U(\mathbf{r}) = a(\mathbf{r})\exp\left[j\varphi(\mathbf{r})\right]$is referred to as the complex amplitude. The wavefunction $u(\mathbf{r},t)$ is therefore related to the complex ampli-

tude by:

$$\begin{aligned} u(\mathbf{r},t) &= \mathrm{Re}\{U(\mathbf{r},t)\} = \mathrm{Re}\{U(\mathbf{r})\exp(j2\pi ft)\} \\ &= \frac{1}{2}\left[U(\mathbf{r})\exp(j2\pi ft) + U^*(\mathbf{r})\exp(-j2\pi ft)\right] \end{aligned} \tag{2.27}$$

At a given position $\mathbf{r}$, the complex amplitude $U(\mathbf{r})$ is a complex variable as shown

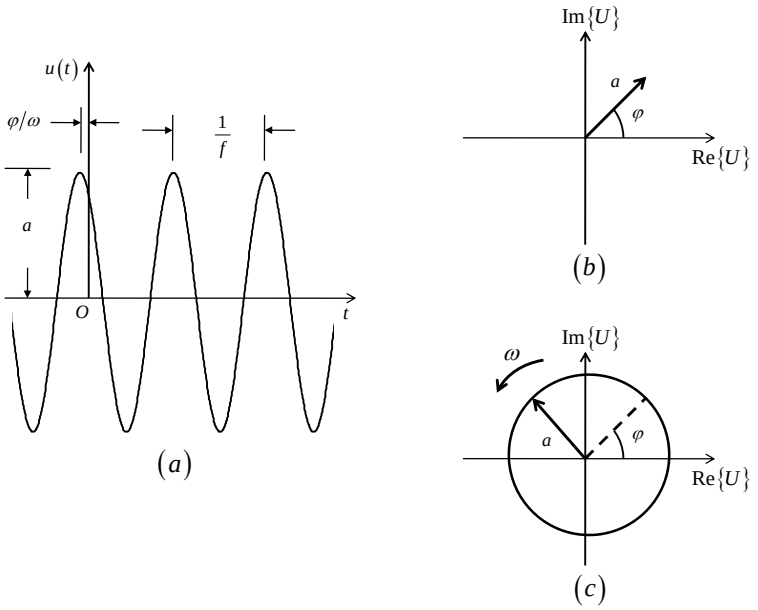

Figure 2.2: Representation of a monochromatic wave at a fixed position $\mathbf{r}$: (a) the wavefunction $u(t)$ is a harmonic function of time; (b) the complex amplitude $U = a\exp(j\varphi)$ is a fixed phasor; (c) the complex wavefunction $U(t) = U\exp(j2\pi ft)$ is a phasor rotating with angular velocity $\omega = 2\pi f$ radians/s [13].

in Figure 2.2(b), whose magnitude $|U(\mathbf{r})| = a(\mathbf{r})$ is the amplitude of the wave and whose $\arg\{U(\mathbf{r})\} = \varphi(\mathbf{r})$ is the phase. The complex wavefunction is represented graphically by a phasor rotating with angular velocity $\omega = 2\pi f$ radians/s (Figure 2.2(c)). Its initial value at $t = 0$ is the complex amplitude $U(\mathbf{r})$ [13].

2.2.3 The Helmholtz Equation

If we substitute $U(\mathbf{r},t) = U(\mathbf{r})\exp(j2\pi ft)$ into equation (2.25), we get:

$$\nabla^2 U(\mathbf{r})\,\mathrm{e}^{j2\pi ft} - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\left[U(\mathbf{r})\,\mathrm{e}^{j2\pi ft}\right] = 0. \tag{2.28}$$

Where, the value of the second derivative $\frac{\partial^2}{\partial t^2}\left[U(\mathbf{r})\,\mathrm{e}^{j2\pi ft}\right] = -(2\pi f)^2 U(\mathbf{r})\,\mathrm{e}^{j2\pi ft}$ can be substituted in the previous equation to arrive at:

$$\begin{aligned}\nabla^2 U(\mathbf{r})\,\mathrm{e}^{j2\pi ft} + \frac{(2\pi f)^2}{c^2} U(\mathbf{r})\,\mathrm{e}^{j2\pi ft} &= 0\\ \left(\nabla^2 + k^2\right) U(\mathbf{r})\,\mathrm{e}^{j2\pi ft} &= 0.\end{aligned}$$

Thus we can now state the Helmholtz equation as follows:

$$\left(\nabla^2 + k^2\right) U(\mathbf{r}) = 0, \tag{2.29}$$

where

$$k = \frac{2\pi f}{c} = \frac{\omega}{c}$$

is referred to as the wavenumber [13].

2.2.4 Intensity, Power and Energy

The optical intensity $I(\mathbf{r}, t)$, defined as the optical power per unit area (units of watts/cm2), is proportional to the average of the squared wavefunction,

$$I(\mathbf{r}, t) = 2\left\langle u^2(\mathbf{r}, t)\right\rangle. \tag{2.30}$$

The operation $\langle\cdot\rangle$ denotes averaging over a time interval of one optical cycle. Using equation (2.30) along with equation (2.22), we can determine the optical intensity. Where,

$$\begin{aligned}2u^2(\mathbf{r}, t) &= 2a^2(\mathbf{r})\cos^2\left[2\pi ft + \varphi(\mathbf{r})\right]\\ &= |U(\mathbf{r})|^2\left(2\cos^2\left[2\pi ft + \varphi(\mathbf{r})\right]\right).\end{aligned}$$

Using the trigonometric identity, $2\cos^2\theta = 1 + \cos(2\theta)$, we have the following representation for $2u^2(\mathbf{r}, t)$:

$$2u^2(\mathbf{r}, t) = |U(\mathbf{r})|^2\left\{1 + \cos\left(2\left[2\pi ft + \varphi(\mathbf{r})\right]\right)\right\}, \tag{2.31}$$

is averaged over an optical period, $1/f$,

$$\begin{aligned} I(\mathbf{r},t) &= 2\left\langle u^2(\mathbf{r},t)\right\rangle \\ &= \frac{1}{1/f}\int_0^{1/f} |U(\mathbf{r})|^2 \left\{1+\cos\left(2\left[2\pi f t+\varphi(\mathbf{r})\right]\right)\right\} dt \\ &= \frac{|U(\mathbf{r})|^2}{1/f}\left(\int_0^{1/f} 1dt+\int_0^{1/f}\cos\left(2\left[2\pi f t+\varphi(\mathbf{r})\right]\right)dt\right) \\ &= \frac{|U(\mathbf{r})|^2}{1/f}\left[t+\frac{1}{4\pi f}\sin\left(2\left[2\pi f t+\varphi(\mathbf{r})\right]\right)\right]_0^{1/f} = \frac{|U(\mathbf{r})|^2}{1/f}\left[\frac{1}{f}+0\right] \\ &= |U(\mathbf{r})|^2. \end{aligned}$$

Therefore the optical intensity $I(\mathbf{r},t) = |U(\mathbf{r})|^2$ of a monochromatic wave is the absolute square of its complex amplitude. And interestingly as we have just shown the intensity of a monochromatic wave does not vary with time [13]. The optical power $P(t)$ units of watts) flowing into an area A normal to the direction of propagation of light is the integrated intensity,

$$P(t) = \int_A I(\mathbf{r},t)\, dA. \tag{2.32}$$

The optical energy (units of joules) collected in a given time interval is the time integral of the optical power over the time interval,

$$E(t) = \int_{t_1}^{t_2} P(t)\, dt. \tag{2.33}$$

2.3 Wavefronts

The wavefronts are the surfaces of equal phase, $\varphi(\mathbf{r}) =$ constant. The constants are often taken to be multiples of 2π, $\varphi(\mathbf{r}) = 2\pi q$, where q is an integer. The wavefront normal at position $\mathbf{r}$ is parallel to the gradient vector $\nabla\varphi(\mathbf{r})$ (a vector with components $\partial\varphi/\partial x$, $\partial\varphi/\partial y$, and $\partial\varphi/\partial z$ in a Cartesian coordinate system). It represents the direction at which the rate of change of the phase is maximum [13].

2.3.1 The Plane Wave

One of the simplest solutions of the Helmholtz equation in a homogeneous medium is the plane wave. Using $(\nabla^2 + k^2) U(\mathbf{r}) = 0$, we have,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} + k^2 U = 0. \tag{2.34}$$

Let $U(\mathbf{r}) = f(x)\, g(y)\, h(z)$, substitute this expression into equation (2.34) and divide by $U(\mathbf{r})$:

$$\frac{g(y)\, h(z)\, f''(x)}{f(x)\, g(y)\, h(z)} + \frac{f(x)\, h(z)\, g''(y)}{f(x)\, g(y)\, h(z)} + \frac{f(x)\, g(y)\, h''(z)}{f(x)\, g(y)\, h(z)} + \frac{k^2 f(x)\, g(y)\, h(z)}{f(x)\, g(y)\, h(z)} = 0,$$

$$\frac{f''(x)}{f(x)} + \frac{g''(y)}{g(y)} + \frac{h''(z)}{h(z)} + k^2 = 0. \tag{2.35}$$

Let $f''/f = -k_x^2$, $g''/g = -k_y^2$, and $h''/h = -k_z^2$, therefore we can state the following relations:

$$k_x^2 + k_y^2 + k_z^2 = k, \tag{2.36}$$

$$\frac{d^2 f(x)}{dx} + k_x^2 f(x) = 0, \tag{2.37}$$

$$\frac{d^2 g(y)}{dy} + k_y^2 g(y) = 0, \tag{2.38}$$

$$\frac{d^2 h(z)}{dz} + k_z^2 h(z) = 0. \tag{2.39}$$

When solving for the differential equations (2.37), (2.38), and (2.39), we have:

$$f(x) = f^+ e^{-jk_x x} + f^- e^{jk_x x}, \tag{2.40}$$

$$g(y) = g^+ e^{-jk_y y} + g^- e^{jk_y y}, \tag{2.41}$$

$$h(z) = h^+ e^{-jk_z z} + h^- e^{jk_z z}. \tag{2.42}$$

The terms with negative exponentials indicate a wave traveling in the positive x, y or z direction, while the terms with positive exponentials result in waves traveling in the negative direction. For our present discussion we will select a wave traveling in

the positive direction, for each coordinate:

$$\begin{aligned} U(\mathbf{r}) &= f(x)\, g(y)\, h(z) \\ &= \left(f^{+}e^{-jk_x x}\right)\left(g^{+}e^{-jk_y y}\right)\left(h^{+}e^{-jk_z z}\right) \\ &= \left(f^{+}g^{+}h^{+}\right)\exp\left[-j\left(k_x x + k_y y + k_z z\right)\right] \\ &= A\exp\left[-j\left(k_x x + k_y y + k_z z\right)\right]. \end{aligned}$$

Let us define a wavenumber vector $\vec{k} = k_x\hat{e}_x + k_y\hat{e}_y + k_z\hat{e}_z = k_o\hat{n}$, where $k_o = |\vec{k}| = \sqrt{k_x^2 + k_y^2 + k_z^2}$ and $\hat{n}$ is a unit vector in the direction of propagation. In addition, we will define a position vector $\vec{r} = x\hat{e}_x + y\hat{e}_y + z\hat{e}_z$, such that the dot product $\vec{k}\cdot\vec{r} = k_x x + k_y y + k_z z$. Therefore, the plane wave $U(\mathbf{r})$ can be stated as follows:

$$U(\mathbf{r}) = A\exp\left(-j\vec{k}\cdot\vec{r}\right), \tag{2.43}$$

where A is a complex constant called the complex envelope, and the phase $\arg\{U(r)\} = \arg\{A\} - \vec{k}\cdot\vec{r}$. If the plane wave is propagating along the positive z–axis $U(\mathbf{r}) = A\exp(-jkz)$ only and assuming $\arg\{A\} = 0$, the corresponding wavefunction will be:

$$u(\mathbf{r}, t) = |A|\cos(2\pi f t - kz). \tag{2.44}$$

To maintain a fixed point on the wave ($2\pi f t - kz = \text{constant}$), one must move in the positive z direction as time increases, as if following a fixed point on the wave. The velocity of the wave in this sense is called the phase velocity v_p, because it is the velocity at which a fixed phase point on the wave travels.

$$\begin{aligned} \frac{d}{dt}\left(\arg\{U(\mathbf{r})\}\right) &= \frac{d}{dt}(\text{constant}) = 0 \\ \frac{d}{dt}(2\pi f t - kz) &= 0 \\ 2\pi f - k\frac{dz}{dt} &= 0 \\ 2\pi f - k v_p &= 0 \\ v_p = \frac{2\pi f}{k} &= \frac{\omega}{k}. \end{aligned}$$

Furthermore, the wavelength λ, is defined as the distance between two successive maxima (or minima or any other reference points) on the wave, at a fixed instant of time. Thus we can deduce the following:

$$[\omega t - kz] - [\omega t - k(z + \lambda)] = 2\pi,$$

$$\begin{aligned} k\lambda &= 2\pi \\ \lambda &= \frac{2\pi}{k}, \end{aligned}$$

Since

$$k = \frac{\omega}{v_p}$$

the wavelength λ can also be stated as:

$$\begin{aligned} \lambda &= \frac{2\pi}{k} = \frac{2\pi v_p}{\omega} = \frac{2\pi v_p}{2\pi f} = \frac{v_p}{f} \\ \lambda &= \frac{v_p}{f}. \end{aligned}$$

2.3.2 Paraxial Waves

A paraxial wave is a plane wave $U(\mathbf{r}) = A(\mathbf{r}) \exp(-jkz)$, with $k = \frac{2\pi}{\lambda}$ and wavelength λ, modulated by a complex envelope $A(\mathbf{r})$ that is a slowly varying function of position. The envelope is assumed to be approximately constant within a neighborhood of size λ, so that the wave locally underlies plane wave nature. Since the change of the phase $\arg\{A(x, y, z)\}$ is small within the distance of a wavelength, the planar wavefronts, $kz = 2\pi q$, of the carrier plane wave bend only slightly, so that their normals are paraxial rays [13]. For the paraxial wave to satisfy the Helmholtz equation, the complex envelope $A(\mathbf{r})$ must satisfy another partial differential equation obtained by substituting $U(\mathbf{r}) = A(\mathbf{r}) \exp(-jkz)$ into equation (2.29). The assumption that $A(\mathbf{r})$ varies slowly with respect to z signifies that within a distance $\Delta z = \lambda$, the change ΔA is much smaller than A itself, i.e., $\Delta A << A$. Since

$$\Delta A = (\partial A/\partial z)\, \Delta z = (\partial A/\partial z)\, \lambda$$

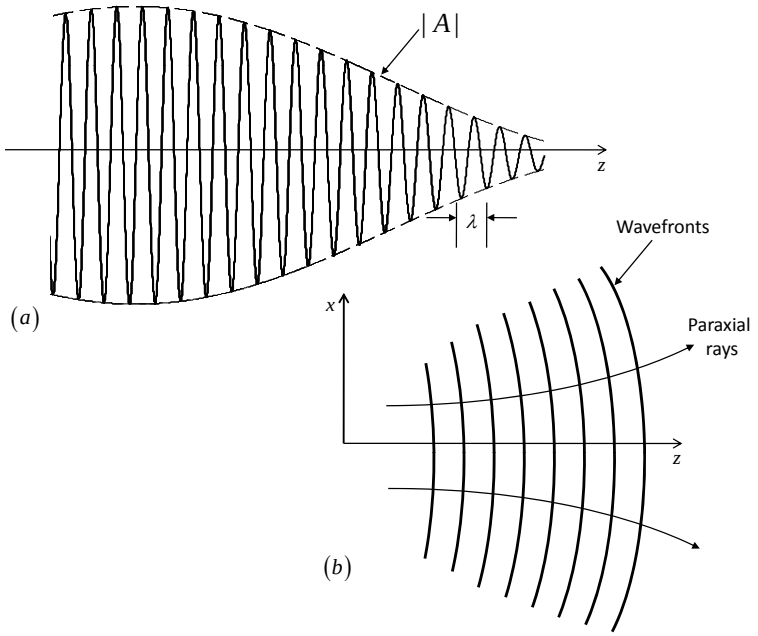

Figure 2.3: (a) The magnitude of a paraxial wave as a function of the axial distance z. (b) The wavefronts and wavefront normals of a paraxial wave [13].

it follows that, $\frac{\partial A}{\partial z}\lambda << A$ which implies $\frac{\partial A}{\partial z} << \frac{A}{\lambda} = \frac{Ak}{2\pi}$. And therefore, $\frac{\partial A}{\partial z} << kA$. Similarly, the derivative $\partial A/\partial z$ varies slowly within the distance λ, so that

$$\partial^2 A/\partial z^2 << k\partial A/\partial z$$

and therefore,

$$\frac{\partial^2 A}{\partial z^2} << k^2 A$$

.

Next, we will substitute $U(\mathbf{r}) = A(\mathbf{r})\exp(-jkz)$ into the Helmholtz equation, and assume $\partial^2 A/\partial z^2$ to be negligible in comparison with $k\partial A/\partial z$ or $k^2 A$:

$$\begin{aligned}\left(\nabla^2 + k^2\right) U(\mathbf{r}) &= 0 \\ \left(\nabla^2 + k^2\right) A(\mathbf{r})\exp(-jkz) &= 0,\end{aligned}$$

$$\exp(-jkz)\frac{\partial^2 A}{\partial x^2} + \exp(-jkz)\frac{\partial^2 A}{\partial y^2} + \frac{\partial^2}{\partial z^2}\left[A\exp(-jkz)\right] + k^2 A\exp(-jkz) = 0. \quad (2.45)$$

The term $\frac{\partial^2}{\partial z^2}\left[A\exp\left(-jkz\right)\right]$ is evaluated accordingly,

$$\frac{\partial}{\partial z}\left[A\exp\left(-jkz\right)\right] = \frac{\partial A}{\partial z}\exp\left(-jkz\right) - jkA\exp\left(-jkz\right).$$

Therefore,

$$\begin{aligned}\frac{\partial^2}{\partial z^2}\left[A\exp\left(-jkz\right)\right] &= \frac{\partial^2 A}{\partial z^2}\exp\left(-jkz\right) - jk\frac{\partial A}{\partial z}\exp\left(-jkz\right)\\ &- jk\left[\frac{\partial A}{\partial z}\exp\left(-jkz\right) - jkA\exp\left(-jkz\right)\right]\\ &= \frac{\partial^2 A}{\partial z^2}\exp\left(-jkz\right) - 2jk\frac{\partial A}{\partial z}\exp\left(-jkz\right) - k^2 A\exp\left(-jkz\right).\end{aligned}$$

Substituting, the expression for $\frac{\partial^2}{\partial z^2}\left[A\exp\left(-jkz\right)\right]$ back into equation (2.45) we get:

$$\left[\frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} + \frac{\partial^2 A}{\partial z^2} - 2jk\frac{\partial A}{\partial z} - k^2 A + k^2 A\right]\exp\left(-jkz\right) = 0.$$

Since we assumed $\partial^2 A/\partial z^2$ to be relatively very small, we finally obtain the following Paraxial Helmholtz equation:

$$\nabla_T^2 A - 2jk\frac{\partial A}{\partial z} = 0, \tag{2.46}$$

where $\nabla_T^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$ is the transverse Laplacian operator. An important solution of the Paraxial Helmholtz equation that exhibits the characteristics of an optical beam is a wave known as the Gaussian beam. In principle, the beam power is concentrated within a small cylinder surrounding the beam axis. The intensity distribution in any transverse plane is a circularly symmetric Gaussian function centered about the beam axis. The width of this function is minimum at the beam waist and grows gradually in both directions. In the next discussion, an expression for the complex amplitude of the Gaussian beam is derived, as well as a description of its physical properties such as intensity, power and beam radius will be provided [13].

2.3.3 The Gaussian Beam

One simple solution to the paraxial Helmholtz equation provides the paraboloidal wave for which,

$$A(\mathbf{r}) = \frac{A_1}{z} \exp\left(-jk\frac{\rho^2}{2z}\right), \tag{2.47}$$

where, $\rho^2 = x^2 + y^2$ and A_1 is a constant. The paraboloidal wave is the paraxial approximation of the spherical wave $U(\mathbf{r}) = (A_1/r)\exp(-jkr)$ when x and y are much smaller than z. Another solution of the paraxial Helmholtz equation provides the Gaussian beam. It is obtained from the paraboloidal wave by use of a simple transformation. Since the complex envelope of the paraboloidal wave is a solution of the paraxial Helmholtz equation, a shifted version of it, with $z - \xi$ replacing z where ξ is a constant,

$$A(\mathbf{r}) = \frac{A_1}{q(z)} \exp\left[-jk\frac{\rho^2}{2q(z)}\right], \tag{2.48}$$

where $q(z) = z - \xi$. This provides a paraboloidal wave centered about the point $z = \xi$ instead of $z = 0$. When ξ is complex, equation (2.48) remains a solution of equation (2.46), but it acquires dramatically different properties. In particular, when ξ is purely imaginary, for instance $\xi = -jz_0$ where z_0 is real, equation (2.48) gives rise to the complex envelope of the Gaussian beam, $A(\mathbf{r}) = (A_1/q(z))\exp[-jk\rho^2/2q(z)]$, with $q(z) = z + jz_0$. In this case, the parameter z_0 is known as the Rayleigh range. To separate the envelope and the phase of this complex envelope, let:

$$\frac{1}{q(z)} = \frac{1}{z + jz_0} \cdot \frac{z - jz_0}{z - jz_0} = \frac{z - jz_0}{z^2 + z_0^2}. \tag{2.49}$$

Thus we can write $1/q(z)$ as:

$$\frac{1}{q(z)} = \frac{1}{R(z)} - j\frac{\lambda}{\pi W^2(z)}, \tag{2.50}$$

where,

$$\frac{1}{R(z)} = \frac{z}{z^2 + z_0^2}$$

and

$$\frac{\lambda}{\pi W^2(z)} = \frac{z_0}{z^2 + z_0^2}$$

. Thus, $R(z)$ can be expressed as,

$$R(z) = \frac{z^2 + z_0^2}{z} = z\left[1 + \left(\frac{z_0}{z}\right)^2\right].$$

While $W(z)$ can also be represented as a function of z and z_0 in the following manner:

$$\begin{aligned} W^2(z) &= \frac{\lambda}{\pi}\left(\frac{z^2 + z_0^2}{z_0}\right) = \frac{\lambda}{\pi} z_0 \left(\frac{z^2}{z_0^2} + 1\right) \\ W(z) &= \left(\frac{\lambda}{\pi} z_0\right)^{1/2} \left(\frac{z^2}{z_0^2} + 1\right)^{1/2} = W_0 \left[1 + \left(\frac{z}{z_0}\right)^2\right]^{1/2}. \end{aligned}$$

Before proceeding further into our derivation, let us define the following:

$$\begin{aligned} |q(z)| &= \left[z^2 + z_0^2\right]^{1/2} = z_0 \left[\left(\frac{z}{z_0}\right)^2 + 1\right]^{1/2} \\ \arg\{q(z)\} &= \tan^{-1}\left(\frac{z_0}{z}\right). \end{aligned}$$

Substituting equation (2.50) into equation (2.48) and using $U(\mathbf{r}) = A(\mathbf{r}) \exp(-jkz)$, we can deduce the following:

$$\begin{aligned} U(\mathbf{r}) &= \frac{A_1}{q(z)} \exp\left(-jk\frac{\rho^2}{2q(z)}\right) \exp(-jkz) \\ &= \frac{A_1}{|q(z)| \exp(j \arg\{q(z)\})} \exp\left(-\frac{jk\rho^2}{2}\left[\frac{1}{R(z)} - j\frac{\lambda}{\pi W^2(z)}\right]\right) \exp(-jkz) \\ &= \frac{A_1}{|q(z)| \exp(j \arg\{q(z)\})} \exp\left(-\frac{jk\rho^2}{2R(z)} - \frac{k\lambda\rho^2}{2\pi W^2(z)}\right) \exp(-jkz). \end{aligned}$$

Since $k = \frac{2\pi}{\lambda}$ we have, $\frac{k\lambda}{2\pi} = 1$, and the above expression can be written as:

$$U(\mathbf{r}) = \frac{A_1}{|q(z)|} \exp(-j \arg\{q(z)\}) \exp\left(-\frac{\rho^2}{W^2(z)}\right) \exp\left(-jkz - \frac{jk\rho^2}{2R(z)}\right). \quad (2.51)$$

Substituting the expression for $|q(z)|$, and $-\arg\{q(z)\} = -\frac{\pi}{2} + \zeta(z)$ into equation (2.51), then multiplying and dividing by W_0, we have the following:

$$\begin{aligned} U(\mathbf{r}) &= \frac{A_1 \cdot W_0}{z_0\left[\left(\frac{z}{z_0}\right)^2 + 1\right]^{1/2} \cdot W_0} \exp\left(j\left[-\frac{\pi}{2} + \zeta(z)\right]\right) \exp\left(-\frac{\rho^2}{W^2(z)}\right) \exp\left(-jkz - \frac{jk\rho^2}{2R(z)}\right) \\ &= \frac{A_1}{jz_0} \cdot \frac{W_0}{W(z)} \exp\left(-\frac{\rho^2}{W^2(z)}\right) \exp\left(-jkz - \frac{jk\rho^2}{2R(z)} + j\zeta(z)\right). \end{aligned}$$

Therefore, the expression for the complex amplitude $U(\mathbf{r})$ of the Gaussian beam can be stated as:

$$U(\mathbf{r}) = A_0 \frac{W_0}{W(z)} \exp\left(-\frac{\rho^2}{W^2(z)}\right) \exp\left(-jkz - jk\frac{\rho^2}{2R(z)} + j\zeta(z)\right). \tag{2.52}$$

A new constant $A_0 = A_1/jz_0$ has been defined for convenience. In addition, the beam parameters can be stated as follows:

$$W(z) = W_0\left[1 + \left(\frac{z}{z_0}\right)^2\right]^{1/2}, \tag{2.53}$$

$$R(z) = z\left[1 + \left(\frac{z_0}{z}\right)^2\right], \tag{2.54}$$

$$\zeta(z) = \tan^{-1}\left(\frac{z}{z_0}\right) \tag{2.55}$$

$$W_0 = \left(\frac{\lambda z_0}{\pi}\right)^{1/2}. \tag{2.56}$$

Equations (2.52)–(2.56) will be further used to determine the intensity and power properties of the Gaussian beam.

2.3.3.1 **Intensity:**

The optical intensity $I(\mathbf{r}) = |U(\mathbf{r})|^2$ is a function of the axial and radial distances z and $\rho = (x^2 + y^2)^{1/2}$

$$I(\rho, z) = I_0 \left[\frac{W_0}{W(z)}\right]^2 \exp\left[-\frac{2\rho^2}{W^2(z)}\right], \tag{2.57}$$

where $I_0 = |A_0|^2$. At each value of z the intensity is a Gaussian function of the radial distance ρ. Due to this, the wave is called a Gaussian beam. The Gaussian function has its peak at $\rho = 0$ (on axis) and decays monotonically as ρ increases. The beam

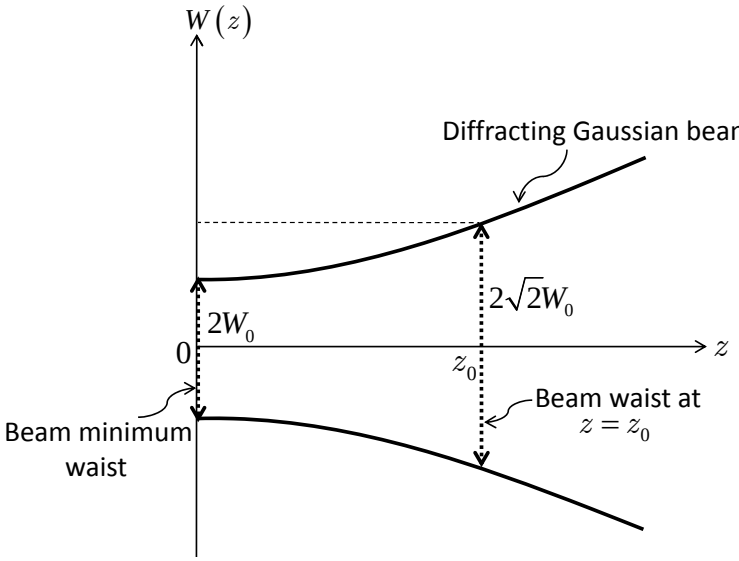

Figure 2.4: Gaussian beam model for the laser source used in the proposed system.

width $W(z)$ of the Gaussian distribution increases with the axial distance. On the beam axis ($\rho = 0$) the intensity,

$$I(0, z) = I_0 \left[\frac{W_0}{W(z)}\right]^2 = \frac{I_0}{1 + (z/z_0)^2}, \tag{2.58}$$

has its maximum value I_0 at $z = 0$ and drops gradually with increasing z, reaching half its peak value at $z = \pm z_0$ as shown in Figure 2.4. When $|z| >> z_0$, $I(0, z) \approx I_0 z_0^2 / z^2$, so that the intensity decreases with the distance in accordance with an inverse-square law. The overall peak intensity $I(0, 0) = I_0$ occurs at the beam center ($z = 0, \rho = 0$).

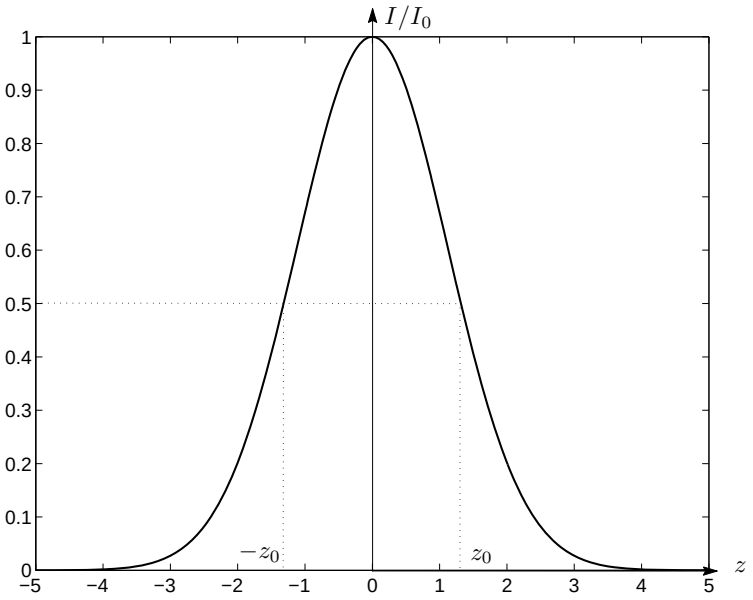

Figure 2.5: The normalized Gaussian intensity profile.

2.3.3.2 **Power:**

The total optical power carried by the beam is the integral of the optical intensity over a transverse plane (say at a distance z),

$$P = \int_A I(\rho, z)\, dA, \tag{2.59}$$

where $dA = \rho d\theta d\rho$, therefore the above integral can be evaluated as follows:

$$\begin{aligned}
P &= \int_0^{\infty}\int_0^{2\pi} I(\rho, z)\, \rho d\rho d\theta \\
&= \int_0^{\infty} I(\rho, z)\, \rho d\rho \cdot \int_0^{2\pi} d\theta \\
&= \int_0^{\infty} I(\rho, z)\, 2\pi\rho d\rho \\
&= \int_0^{\infty} I_0 \left[\frac{W_0}{W(z)}\right]^2 \exp\left[-\frac{2\rho^2}{W^2(z)}\right] 2\pi\rho d\rho \\
&= I_0 \left[\frac{W_0}{W(z)}\right]^2 2\pi \int_0^{\infty} \exp\left[-\frac{2\rho^2}{W^2(z)}\right] \rho d\rho.
\end{aligned}$$

By using a change of variables, $u = \rho^2$ where $du/2 = \rho d\rho$, and inserting them into the above double integral, we attain to the following:

$$\begin{aligned} P &= I_0 \left[\frac{W_0}{W(z)}\right]^2 \frac{2\pi}{2} \int_0^\infty \exp\left[-\frac{2u}{W^2(z)}\right] du \\ &= I_0 \left[\frac{W_0}{W(z)}\right]^2 \frac{2\pi}{2} \cdot \frac{-W^2(z)}{2} \left(\exp\left[-\frac{2u}{W^2(z)}\right]\right)_0^\infty \\ &= -I_0 \left[\frac{W_0}{W(z)}\right]^2 \frac{\pi}{2} W^2(z)(0-1) \\ &= I_0 \left[\frac{W_0}{W(z)}\right]^2 \frac{\pi}{2} W^2(z) = \frac{I_0}{2}\left(\pi W_0^2\right). \end{aligned}$$

Therefore the total optical power can be stated as:

$$P_T = \frac{1}{2} I_0 \left(\pi W_0^2\right), \tag{2.60}$$

where the result is independent of z. Thus the beam power is one-half the peak intensity times the beam area. Since beams are often described by their power P, it is useful to express I_0 in terms of P using equation (2.60) and to rewrite equation (2.57) in the form:

$$I(\rho, z) = \frac{2P_T}{\pi W^2(z)} \exp\left[-\frac{2\rho^2}{W^2(z)}\right]. \tag{2.61}$$

The ratio of the power carried within a circle of radius ρ_0 in the transverse plane at position z to the total power is:

$$\frac{1}{P_T} \int_0^{\rho_0} I(\rho, z)\, 2\pi\rho d\rho = 1 - \exp\left[-\frac{2\rho_0^2}{W^2(z)}\right]. \tag{2.62}$$

The power contained within a circle of radius $\rho_0 = W(z)$ is approximately 86% of the total power. About 99% of the power is contained within a circle of radius $1.5W(z)$. Since the radius of the circular spot is $\rho_0 = 0.5$ cm, then to achieve 99% of the total power W_0 was set to 1/3 cm in the theoretical model. Therefore, the minimum beam waist $2W_0$ is equal to 2/3 cm.

3

Functional Approximation with Wavelet Networks

In order to estimate the lateral position of the light spot center corresponding to the power distribution of the photodetector array, a wavelet network will be used as a function approximation technique. In this chapter, we will introduce the concept of function approximation, and neural networks. Next, we will discuss the relation between the wavelet frames and wavelet networks, as well as the wavelet network structure, and learning procedure that will be adopted in our research.

3.1 Function Approximation

According to T. Poggio in [16], the problem of learning a mapping between an input and an output space is similar to the problem of synthesizing an associative memory that retrieves the appropriate output when presented with the input and generalizes

when presented with new inputs. A classical framework for this problem is approximation theory which deals with the problem of approximating or interpolating a continuous, multivariate function $f(\mathbf{x})$ by an approximating function $F(\mathbf{w}, \mathbf{x})$ having a fixed number of parameters $\mathbf{w}$ belonging to some set $\mathbf{P}$. In this case, $\mathbf{x}$ and $\mathbf{w}$ are real vectors where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and $\mathbf{w} = [w_1, w_2, \ldots, w_m]$.

For a choice of a specific F, the problem is to find the set of parameters $\mathbf{w}$ that provides the best possible approximation of f on the given input/output data set. This can be categorized as the learning stage of our approximation problem. Therefore, it is important to select an approximating function F that can represent f as well as possible. It would be pointless to try to learn, if the chosen approximation function $F(\mathbf{w}, \mathbf{x})$ could only give a very poor representation of $f(\mathbf{x})$, even when using optimal parameter values. Thus, we need to distinguish three main problems invloved in function approximation; (1) the problem of which approximation to use, that is which approximating functions $F(\mathbf{w}, \mathbf{x})$ would effectively represent the function $f(\mathbf{x})$; (2) the problem of which algorithm to use for finding the optimal values of the parameters $\mathbf{w}$ for a given choice of F; (3) the problem of an efficient implementation of the algorithm either through hardware or software or both [16].

Most approximation schemes can be mapped into a certain network that can be dubbed as a neural network. In general, networks can be regarded as a graphic notation for a large class of algorithms. In our discussion, a network is a function represented by the composition of a number of basic functions.

To measure the quality of the approximation, one introduces a distance function ρ to determine the distance $\rho[f(\mathbf{x}), F(\mathbf{w}, \mathbf{x})]$ of an approximation $F(\mathbf{w}, \mathbf{x})$ from $f(\mathbf{x})$. The distance is usually induced by a norm, such as the standard L^2 norm. The approximation problem can then be stated formally as:

DEFINITION 2.1 *If $f(\mathbf{x})$ is a continuous function and $F(\mathbf{w}, \mathbf{x})$ is an approximation function that depends continuously on $\mathbf{w} \in \mathbf{P}$ and $\mathbf{x}$, the approximation problem is to determine the parameters $\mathbf{w}^*$ such that,*

$$\rho[F(\mathbf{w}^*, \mathbf{x}), f(\mathbf{x})] \leq \rho[F(\mathbf{w}, \mathbf{x}), f(\mathbf{x})], \tag{3.1}$$

for all $\mathbf{w}$ in the set $\mathbf{P}$.

A solution of this problem, if it exists, is said to be a best approximation. The existence of a best approximation depends ultimately on the class of functions

to whom $F(\mathbf{w}, \mathbf{x})$ belongs [16].

Recently, the wavelet theory has received substantial interest in the fields of numerical analysis and signal processing [17], [18]. Wavelets are a family of basis functions which exhibit interesting properties such as orthogonality, compact support, and localization in time and frequency [19]. Owing to wavelet theory, very efficient and fast algorithms have been developed for analyzing, approximating, and estimating functions or signals. However, the implementation of such algorithms is only adequate for problems of a relatively small input dimension. This is due to the excessive cost of constructing and storing wavelet basis of large dimension. Artificial neural networks are considered more promising candidates for handling problems of larger dimension and their complexity is less sensitive to the input dimension [20].

Neural networks have been established as general function approximation tools for fitting nonlinear models from input/output data and are widely used in applications which involve system modeling and identification [16]. However, the practical implementation of neural networks suffers from the lack of efficient constructive methods, both for determining the parameters of neurons and selection of the network structure. At a different rate, the recently introduced wavelet decomposition is emerging as a powerful tool for approximation [21], [22]. Due to the similar structure of wavelet decomposition and one-hidden-layer neural network, Q. Zhang and A. Benveniste in [23], [20], [24] proposed to combine both wavelets, and neural networks. This new type of network by the name of wavelet network (WN), is presented in [23], as a class of feed-forward networks composed of wavelets which act as activation functions replacing the traditional sigmoidal functions. The basic idea is to use more powerful computing units obtained by cascading wavelet transform as an alternative to neurons. The WN merges the good localization properties of wavelets with the approximation abilities of neural networks. In addition, the wavelet network learning is performed by the standard back-propagation type algorithm as in the conventional feed-forward neural network [19].

3.2 Neural networks

In this section a brief overview of neural networks and their structure will be provided. Let $\boldsymbol{\Theta}$ be a set containing pairs of sampled inputs and the corresponding outputs

generated by an unknown map, $f : \mathbb{R}^m \to \mathbb{R}^n$, m, $n < \infty$, such that:

$$\boldsymbol{\Theta} = \{(x^p, y^p) : y^p = f(x^p); x^p \in \mathbb{R}^m, y^p \in \mathbb{R}^n, i = 1, \ldots, N_p, N_p < \infty\}.$$

We call $\boldsymbol{\Theta}$ the training set. The task of functional approximation is to use the data provided in $\boldsymbol{\Theta}$ to learn or approximate the map f. Numerous existing schemes to perfom such a task are based on parametrically fitting a particular functional form to the given data. Simple examples of such schemes are those which attempt to fit linear models or polynomials of fixed degree to the data in $\boldsymbol{\Theta}$. More recently, feedforward neural networks have been used to learn the map f [25].

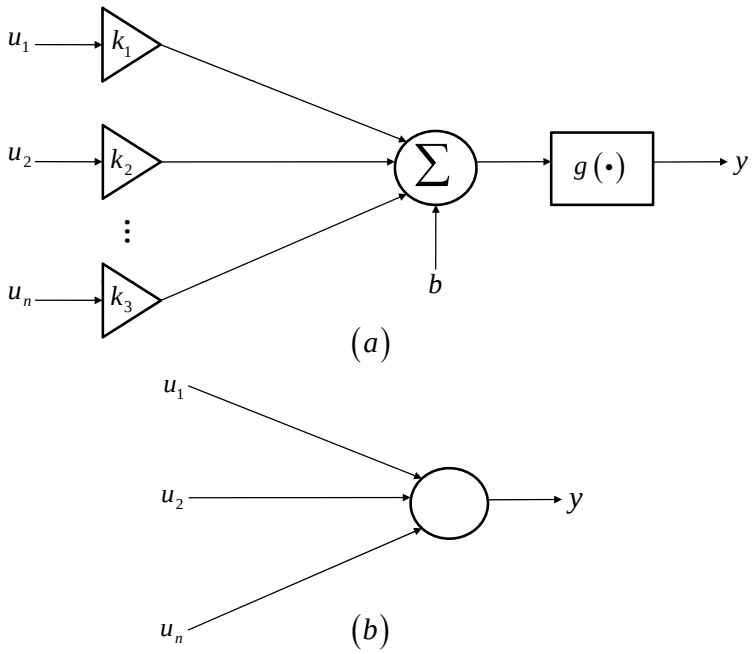

Figure 3.1: (a) Single neuron model. (b) Simplified schematic of single neuron [25].

The basic component in a feedforward neural network is the single neuron model as shown in Figure 3.1(a). Where $u_1, \ldots, u_n$ are the inputs to the neuron, $k_1, \ldots, k_n$ are multiplicative weights applied to the inputs, b is a biasing input, $g : \mathbb{R} \to \mathbb{R}$, and y is the output of the neuron. Thus, we have:

$$y = g\left(\sum_{i=1}^{n} k_i u_i + b\right)$$

The neuron of Figure 3.1(a) is often depicted as illustrated in Figure 3.1(b), where

the input weights, bias, summation and function g are implicit. Traditinally, the activation function g has been chosen to be the well known sigmoidal function. This choice of g was initially based upon the observed firing rate response of biological neurons. A feedforward neural network is constructed by interconnecting a number of neurons (such as the one shown in Figure 3.1) so as to form a network in which all connections are made in the forward direction, that is from input to output without feedback loops, as shown in Figure 3.2. Neural networks of this form are usually

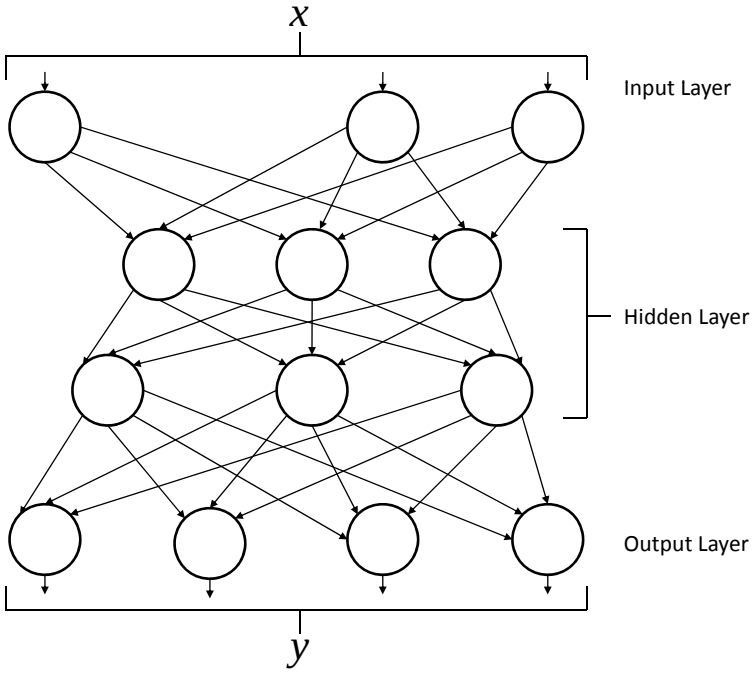

Figure 3.2: Feedforward neural network [25].

composed of an input layer, a number of hidden layers, and an output layer. The input layer consists of neurons which accept external inputs to the network. Inputs and outputs of the hidden layers are internal to the network, and hence the term hidden. Outputs of neurons in the output layer are the external outputs of the network. Once the structure of the feedforward network has been decided, that is the number of hidden layers and the number of nodes in each hidden layer has been set, a mapping is learned by varying the connection weights w_{ij}'s and the biases b_j's so as to obtain the desired input-output response for the network. One method often used to vary the weights and biases is known as the backpropagation algorithm in which

the weights and biases are modified so as to minimize a cost function of the form,

$$E = \sum_{(x^p, y^p) \in \Theta} \| O^p - y^p \|^2,$$

where O^p is the output vector at the output layer of the network when x^p is applied at the input. In this case, w_{ij} denotes the weight applied to the output O_j of the jth neuron when connecting it to the input of the ith neuron, and b_j is the bias input to the jth neuron. Backpropagation employs gradient descent to minimize E. That is, the weights and biases are varied in accordance with the rules,

$$\Delta w_{ij} = -\varepsilon \frac{\partial E}{\partial w_{ij}},$$

and,

$$\Delta b_j = -\varepsilon \frac{\partial E}{\partial b_j}.$$

Feedforward neural networks are known to have empirically demonstrated ability to approximate complicated maps very well using the technique just described.

3.3 Wavelet Transforms

In this section we shortly state some basic concepts about wavelet transforms, which involve continuous wavelet transform and wavelet bases and frames, that will be useful in understanding the construction and development of wavelet networks.

3.3.1 The Continuous Wavelet Transform (CWT)

Historically the continuous wavelet transform was the first studied wavelet transform. To introduce the wavelet transform we assume that a wavelet function $\psi(x)$ is given that satisfies the following two requirements:

(1) $\psi(x)$ is continuous and has exponential decay, that is $\psi(x) \leqslant Me^{-C|x|}$ for some constants C and M.

(2) The integral of ψ is zero, that is $\int_{-\infty}^{\infty} \psi(x)\, dx = 0$ [26].

An example of a suitable wavelet function is $\psi(x) = xe^{-x^2}$, whose graph is given in Figure 3.3. In the following discussion, we assume that $\psi(x)$ equals zero

outside some fixed interval $-A \leqslant x \leqslant A$, which is a stronger condition than the first condition just given. We are now ready to state the definition of the wavelet transform.

DEFINITION 2.2 *Given a wavelet ψ satisfying the two requirements just given, the wavelet transform of a function $f \in L^2(\mathbb{R})$ is a function $w : \mathbb{R}^2 \mapsto \mathbb{R}$ given by*

$$w(d,m) = \frac{1}{\sqrt{|d|}} \int_{-\infty}^{\infty} f(x)\, \psi\left(\frac{x-m}{d}\right) dx. \tag{3.2}$$

From the preceding definition, it is not clear how to define the wavelet transform at $d = 0$. However, the change of variables $y = (x - m)/d$ converts the wavelet transfrom into the following:

$$w(d,m) = \sqrt{|d|} \int_{-\infty}^{\infty} f(yd+m)\, \psi(y)\, dy. \tag{3.3}$$

From this representation clearly, $w(d,m) = 0$ when $d = 0$.

As d becomes small, the graph of

$$\psi_{d,m}(x) = \frac{1}{\sqrt{|d|}} \psi\left(\frac{x-m}{d}\right),$$

becomes tall and skinny, as illustrated in the graphs of $\psi_{1,0}$ and $\psi_{1/2,0}$ with $\psi(x) = xe^{-x^2}$ given in Figures 3.3 and 3.4, respectively. Therefore, the frequency of $\psi_{d,m}$ increases as d decreases. In addition, if most of the support of ψ, that is the nonzero part of the graph of ψ, is located near the origin, then most of the support of $\psi_{d,m}$ will be located near $x = m$. So $w(d,m)$ measures the frequency component of f that vibrates with frequency proportional to $1/d$ near the point $x = m$.

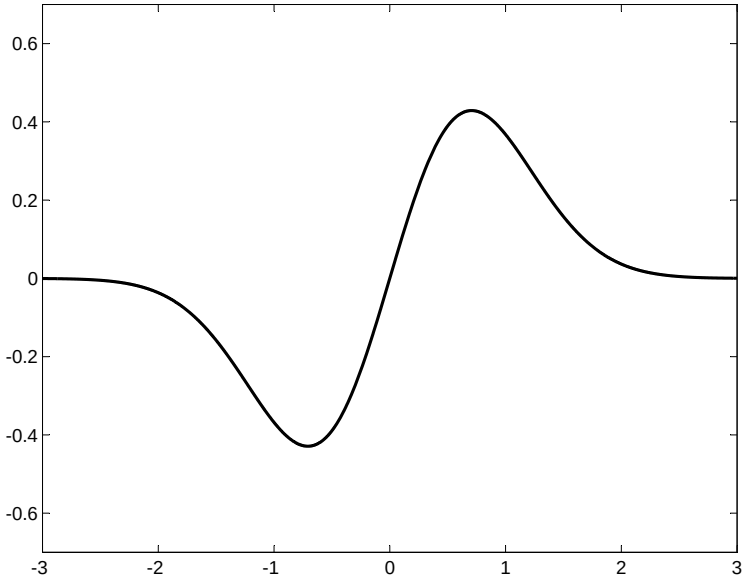

Figure 3.3: Graph of $\psi_{1,0}(x) = \psi(x) = xe^{-x^2}$ [26].

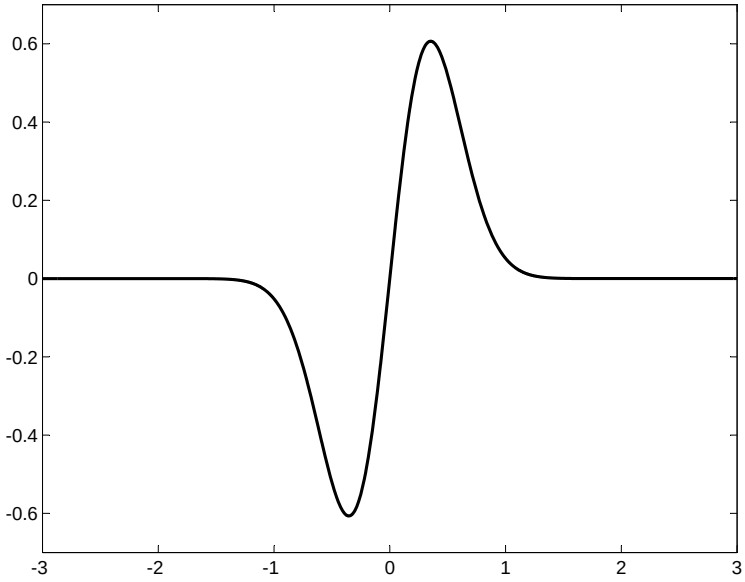

Figure 3.4: Graph of $\psi_{1/2,0}$ [26].

3.3.2 Inverse Wavelet Transform

The inversion formula of the wavelet transform is given in the following theorem:

Theorem 2.1 *Suppose ψ is a continuous wavelet satisfying the following conditions; ψ has exponential decay at infinity and $\int_{-\infty}^{\infty} \psi(x)\,dx = 0$. Then for any function $f \in L^2(\mathbb{R})$, the following inversion formula holds*:

$$f(x) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |d|^{-1/2} \frac{w(d,m)}{d^2} \psi\left(\frac{x-m}{d}\right) dm\, dd, \tag{3.4}$$

where, the Fourier transform $\hat{\psi}(\omega)$ of ψ satisfies the following condition in [26], [23]:

$$C_\psi = 2\pi \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty. \tag{3.5}$$

Therefore, according to the continuous wavelet decomposition theory, we are able to decompose any function $f(x) \in L^2(\mathbb{R})$ using a family of functions obtained by dilating and translating a single mother wavelet $\psi(x)$. In addition, the preceding wavelet inversion theorem states that a function f can be decomposed as a weighted sum (or integral) of its frequency components, as measured by $w(d,m)$. The wavelet inversion theorem involves two parameters, namely, the translation m and dialation d, since the wavelet transform gives a measures of the frequency (using the parameter d) of f near the point $x = m$ [26].

3.3.3 Wavelet bases and frames

The continuous wavelet transform and its inverse transform are not directly implementable on digital computers. In practice, they have to be discretized [20]. When the inverse wavelet transform in equation (3.4) is discretized into:

$$f(x) = \sum_{j=1}^{N} w_j \frac{1}{\sqrt{|d_j|}} \varphi_j(x). \tag{3.6}$$

some conditions are required so that this discrete version of the reconstruction of f holds. The wavelet $\varphi_j(x)$ is derived from its mother wavelet $\psi(x)$ by the following relation:

$$\varphi_j(x) = \psi\left(\frac{x-m_j}{d_j}\right) = \psi(z_j), \tag{3.7}$$

where m_j and d_j are the discretized translation and dilation factors.

Let Ω_c be a denumerable family of functions generated by ψ of the following form:

$$\Omega_c = \left\{ \frac{1}{\sqrt{d_j}} \psi \left(\frac{x - m_j}{d_j} \right) : m_j \in \mathbb{R}, d_j \in \mathbb{R}^+, j \in \mathbb{Z} \right\}, \tag{3.8}$$

which constitute an orthonormal basis of some functional space such as $L^2(\mathbb{R})$. Usually a regular lattice $\{(d_0^a, tm_0) : a \in \mathbb{Z}, t \in \mathbb{Z}\}$ is used for the discretization where the scalar parameters d_0 and m_0 define the step sizes of dilation and translation discretizations.

Wavelet bases have numerous applications in signal processing and numerical analysis because they offer very efficient algorithms and provide more useful information than Fourier transform. However, it is not always possible to build orthonormal wavelet bases with any wavelet function ψ.

Though there are some well developed techniques for constructing the wavelet function ψ and its associated orthonormal basis, the wavelet function ψ has to satisfy strong restrictions. These restrictions lead to conflicts between regularity and compactness of the wavelet function, both being desired properties. Furthermore, if one gives up the idea that the discrete family in equation (3.8) should be a basis of some considered functional space and requires only that equation (3.8) constitutes a frame, then one gains more freedom on the choice of ψ [20].

Wavelet frames are redundant basis, constructed by simple operations of translation and dilation of the mother Wavelet, which must satisfy conditions less stringent than their orthonormal counterparts. The frame condition can be stated as follows; there exist two constants $c_{\min} > 0$ and $c_{\max} < \infty$ such that for all $f \in L^2(\mathbb{R})$, the following inequalities hold:

$$c_{\min} \parallel f \parallel^2 \leq \sum_{\varphi_j \in \Omega_c} |\langle \varphi_j, f \rangle|^2 \leq c_{\max} \parallel f \parallel^2. \tag{3.9}$$

In this sum, $\parallel f \parallel$ denotes the norm of function and $\langle \cdot, \cdot \rangle$ is the inner product in $L^2(\mathbb{R})$ and the sum ranges over all the elements of the family Ω_c [23], [27]. As mentioned earlier using wavelet frames rather than orthonormal basis provides more freedom and flexibility in the choice of the wavelet function ψ, however the tradeoff here is that the reconstruction of the coefficients w_j in equation 3.6 becomes nontrivial [20].

3.4 Wavelet Networks (WN)

In this section we discuss the connection between wavelet networks and the discrete inverse wavelet transform. In addition, we give an overview of the structure of the wavelet network and the process of learning for function approximation.

3.4.1 Adaptive Discretization

Although the wavelet bases and frames have been developed with efficient numerical algorithms, their applications have been limited to problems of relatively small input dimension. The main reason behind this is that wavelet bases and frames are usually constructed with regularly dilated and translated wavelets, independent of the available measured information or training data. In practice, the construction and storage of such wavelet basis or frame of large input dimension is of prohibitive cost. Therefore, it is expected that the wavelet estimator will be more efficient if the wavelet basis is constructed according to the training data. This inevitably yields the idea of adaptive discretization of the continuous wavelet transform [20].

To elaborate more on the preceding point, in order for us to obtain a discrete reconstruction as shown in equation (3.6), instead of using a fixed lattice of (d_j, m_j), we can adaptively determine the values of (d_j, m_j), according to the function f or the sampled input/output data in the set $\mathbf{\Theta}$. By following such a methodology, all the parameters w_j, d_j, and m_j in equation (3.6) are to be adapted. Thus, equation (3.6) is very similar to a one hidden layer feedforward neural network. Such adaptive discrete inverse wavelet transform is called wavelet network.

From this perspective, equation (3.6) can be constructed using techniques of neural networks. Usually, neural networks used in function approximation are first randomly initialized and then trained by a backpropagation procedure. The random initialization makes such learning procedures very inefficient. In contrast, wavelet networks can be initialized with regular wavelet lattice as will be shown later in Chapter 5. It is to be remarked that the regular lattices of wavelet frames are special cases of adaptive discretizations of the continuous inverse wavelet transform. Consequently, the discrete reconstruction formula in equation (3.6) must hold for some properly adapted (d_j, m_j). Furthermore, if the function f has some particular

property of regularity, better results can be obtained, due to the flexibility of the adaptive wavelet family [20].

3.4.2 Wavelet Network Structure

Zhang, and Benveniste introduced the general wavelet network structure, based on the so-called $\left(1+\frac{1}{2}\right)$-layer neural network [23]. In the next discussion we use the modified version of this network presented by Zhang in [28]. The main difference between the two approaches is that in [28] a linear term $\mathbf{a_k} = [a_{k1}, a_{k2}, \ldots, a_{kNi}]$ is introduced to help the learning of the linear relation between the input and the output signals. The WN architecture used is shown in Figure 3.5, and the equation that defines the network is given by:

$$\hat{y}_k(x) = \sum_{j=1}^{N_w} c_{kj}\Phi_j(x) + \sum_{i=1}^{N_i} a_{ki}x_i + b_k, \tag{3.10}$$

where i is the index for input nodes, j is the index for hidden nodes, k is the index for output nodes, N_i is the number of input nodes, N_w is the number of hidden nodes, and N_o is the number of output nodes. As shown in Figure 3.5, the wavelet network is composed of an input layer with N_i inputs, a hidden layer having N_w wavelet neurons or wavelons, and an output layer having a linear output neuron. The coefficients of the linear part of the network which consist of the components of the vector $\mathbf{a_k}$ and the bias term b_k are called direct connections [27]. In addition, x_i refers to the ith input to the network, $\hat{y}_k(x)$ is the kth output of the network, and $\Phi_j(x)$ represents the multidimensional activation function at the jth hidden wavelon. For modeling multi-variable processes, multidimensional wavelets $\Phi_j : \mathbb{R}^{N_i} \mapsto \mathbb{R}$ must be defined. These are constructed as the product of N_i scalar wavelets:

$$\Phi_j(x) = \prod_{i=1}^{Ni} \psi(z_{ji}) = \prod_{i=1}^{Ni} \varphi_{ji}. \tag{3.11}$$

Here, $z_{ji} = (x_i - m_{ji})/d_{ji}$, and the array $\mathbf{x}$ is given by $\mathbf{x} = [x_1, x_2, \ldots, x_i, \ldots, x_{Ni}]^{\mathrm{T}}$. Families of multidimensional wavelets generated according to this scheme have been shown to be frames of $L^2\left(\mathbb{R}^{Ni}\right)$ [23], [24]. In this work, we have selected $\psi(z_{ji}) = -z_{ji}e^{-z_{ji}^2/2}$ as our scalar mother wavelet, which satisfies condition (3.5), and in the

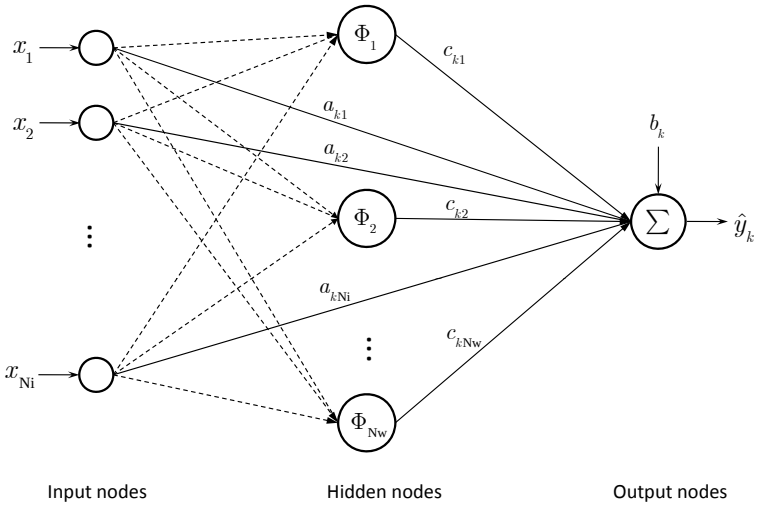

Figure 3.5: Function approximation using wavelet networks.

multidimensional case, direct products of such scalar wavelets have been taken.

3.4.3 WN Learning

The learning algorithm for adjusting the parameters of the WN is based on a sample of input/output pairs, $\{\mathbf{x}, y_k(\mathbf{x})\}$, where $y_k(\mathbf{x})$ is the function to be approximated. The WN training depends on minimizing the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{p=1}^{Np}\sum_{k=1}^{No}(e_k^p)^2 = \frac{1}{2}\sum_{p=1}^{Np}\sum_{k=1}^{No}(y_k^p - \hat{y}_k^p)^2, \tag{3.12}$$

where $e_k^p = y_k^p - \hat{y}_k^p$ is the error between the kth target output, y_k^p, and the corresponding wavelet network output, $\hat{y}_k^p$, for training pattern p, while N_p is the total number of elements in the training set. All the parameters of the wavelet network to be adjusted are collected in a vector $\theta = [b_k, a_{ki}, c_{kj}, m_{ji}, d_{ji}]^{\mathrm{T}}$. The minimization is performed based on the gradient descent algorithm. The partial derivative of the cost function with respect to θ is:

$$\frac{\partial J}{\partial \theta} = -\sum_{p=1}^{Np}\sum_{k=1}^{No} e_k^p \frac{\partial \hat{y}_k^p}{\partial \theta}. \tag{3.13}$$

The components of the vector $\partial\hat{y}_k^p/\partial\theta$ for the conventional WN can be derived as follows:

$$\frac{\partial\hat{y}_k^p}{\partial b_n} = \delta_{nk} \tag{3.14}$$

$$\frac{\partial\hat{y}_k^p}{\partial a_{ni}} = \delta_{nk} x_i^p \tag{3.15}$$

$$\frac{\partial\hat{y}_k^p}{\partial c_{nj}} = \delta_{nk} \Phi_j^p \tag{3.16}$$

$$\frac{\partial\hat{y}_k^p}{\partial m_{ji}} = c_{kj} \frac{\partial\Phi_j^p}{\partial z_{ji}^p} \frac{\partial z_{ji}^p}{\partial m_{ji}} \tag{3.17}$$

$$\frac{\partial\hat{y}_k^p}{\partial d_{ji}} = c_{kj} \frac{\partial\Phi_j^p}{\partial z_{ji}^p} \frac{\partial z_{ji}^p}{\partial d_{ji}}. \tag{3.18}$$

Here, $z_{ji}^p = (x_i^p - m_{ji})/d_{ji}$, for $(i = 1, \ldots, N_i)$, $(j = 1, \ldots, N_w)$, $(k = 1, \ldots, N_o)$, and $(p = 1, \ldots, N_p)$. Notice that we used Kronecker's symbol delta defined as: $\delta_{nk} = 1$ for $n = k$ and $\delta_{nk} = 0$ for $n \neq k$.

The network parameter vector θ is updated every epoch by using $\theta_{\text{new}} = \theta_{\text{old}} + \Delta\theta\,(l)$ where:

$$\Delta\theta\,(l) = -\frac{1}{Np}\left(\mu\frac{\partial J}{\partial\theta}\right) + \gamma\Delta\theta\,(l-1)\,. \tag{3.19}$$

Here, μ is the learning rate and $\gamma = 1 - \mu$ is the momentum coefficient, where both are set in the interval $(0, 1)$. The figure of merit used to assess the approximation results is the mean squared error (MSE), which can be stated as follows:

$$\text{MSE} = \frac{1}{Np}\sum_{p=1}^{Np}\sum_{k=1}^{No}(y_k^p - \hat{y}_k^p)^2 = \frac{2}{Np}J\,(\theta)\,. \tag{3.20}$$

4

Optical System Modeling and Design

4.1 System Architecture

The hardware architecture of our system, as shown in Figure 4.1, consists of a laser source mounted on two actuators for azimuthal and vertical motion steering. The laser source is adjusted to point at a mirror placed on a vibrating platform. The Gaussian beam emitted from the source is reflected by the mirror onto a photodetector array, which captures the light intensity distribution of the laser spot. Different photocurrent outputs generated from each photodiode (PD) depend on the relative beam area intercepted by the different PDs. Thus, the optical power distribution of the photodetector array depends on the 2 dimensional position of the spot center.

The design problem at hand involves three major parts as demonstrated in Figure 4.1. The first part, the optical acquisition system, is concerned with acquiring the optical power distribution $P = [P_1 \cdots P_i \cdots P_n]^{\mathrm{T}}$ of the array of n photodetector cells. Next, the optical power information is fed into a lateral position detection system. In this stage a WN is used as a function approximation technique to yield an

estimate $\hat{x}(P,t)$ and $\hat{y}(P,t)$ of both $x(P,t)$ and $y(P,t)$, the Cartesian coordinates of the center of the light spot that correspond to the acquired optical power distribution. The final stage uses the estimated position of the light spot for vibration monitoring. In this research, we focus on the first two stages of the proposed system.

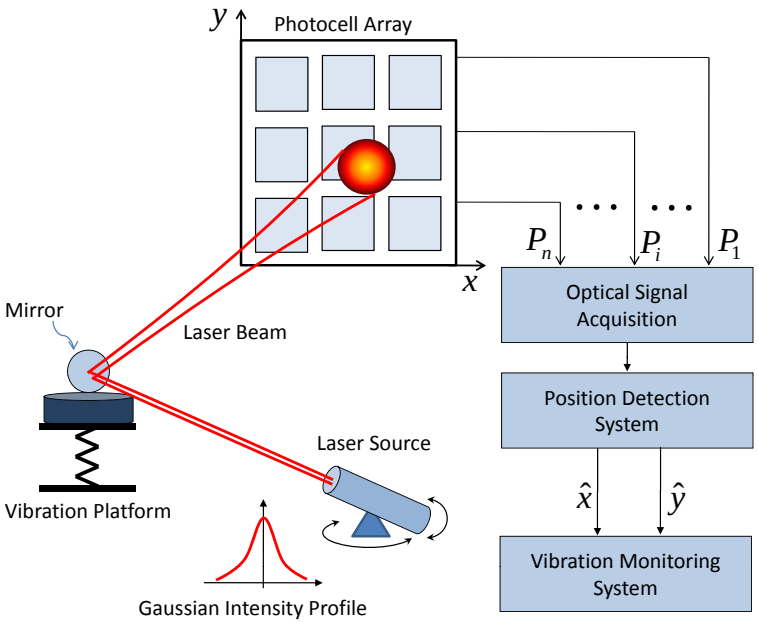

Figure 4.1: Hardware architecture of the proposed position detection system.

4.2 Theoretical Optical Acquisition Model

The theoretical model evaluates the optical power distribution as the center of the beam is scans the plane of photocell array. Since, optical power is directly related to the incident area of overlap between the photodetector and the beam $\left(P_i = \int I dA\right)$, closed form equations for the area of intersection for one photodetector cell have been found for further analytical purposes.

Next, we derive the equations for the optical power covered by the required area of overlap between the detector active surface and the beam spot, given the beam's Gaussian intensity profile. After evaluating the optical power as the beam is scanned over the active surface of one photodetector cell, the theoretical optical acquisition system model is extended further to account for an array of four photodiodes with horizontal and vertical spatial gap separations. The optical power for each photodiode is calculated as the beam moves about the plane covered by the quadcell.

The laser beam incident onto a 1 cm×1 cm square photodetector cell is modeled as a circular spot with a radius, $\rho_0 = 0.5$ cm. We developed a code to compute the interception area between the circular beam spot and the photodetector as the beam center is moved anywhere outside and inside the boundaries of the square cell. The position of the beam center is determined with respect to the origin of the Cartesian coordinate system, located at the center of the photodetector.

To calculate the area through simulation, certain supporting parameters such as the number of points of intersection between the circular spot and the square cell N_{int}, as well as the number of corners N_c of the square that lie within the circle, were found.

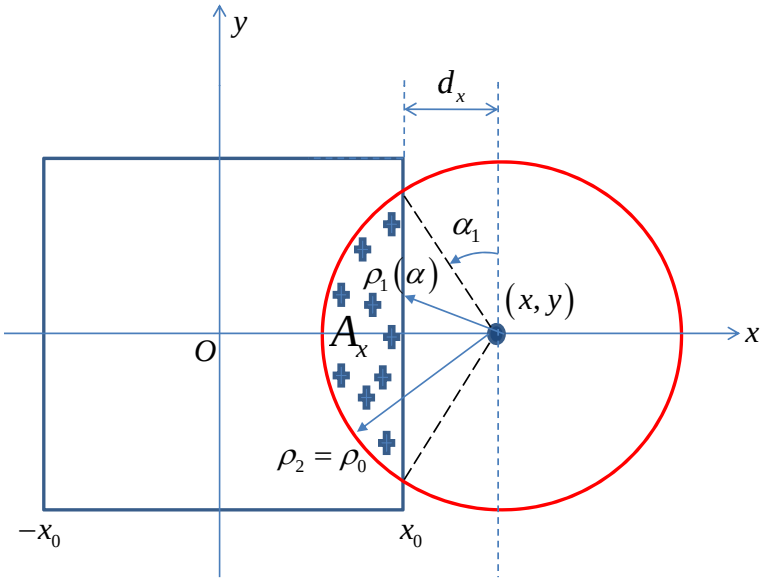

Figure 4.2: Parameters definition for area A_x.

Next, we derived the mathematical formulation necessary to obtain the closed form equations for the overlap area. This will be divided into three categories; (1) Finding the area of intersection while moving the beam along the x direction only, (2) Finding the area of intersection while the beam is moved along the y direction only, and finally (3) Finding the area of intersection as the beam is moved about the entire x–y plane, along any random path.

As shown in Figure 4.2, (x, y) indicate the coordinates of the center of the circular spot with respect to the center of the square photodetector cell, and (x_0, y_0) represent the coordinates of the top right-most corner of the photocell, which in this

case $x_0 = 0.5$ cm and $y_0 = 0.5$ cm. Let us first define the following parameters:

$$d_x = x - x_0, \tag{4.1}$$

which is the horizontal distance between the spot center and the side at which the circular spot intersects the square. In addition, we have:

$$\sin\alpha_1 = \sin\alpha_2 = \frac{x - x_0}{\rho_0}, \tag{4.2}$$

where, α_1 is the angle between the vertical axis crossing through the center of the beam and the first intersection between the circular spot and the square photocell. While, α_2 is the angle between the vertical axis crossing through the beam center and the second point of intersection.

Given that $\alpha_2 = \pi - \alpha_1$, $\rho_1(\alpha) = (x - x_0)/\sin\alpha$, and $\rho_2(\alpha) = \rho_0$, we can calculate the intersection area A_x by evaluating the following double integral:

$$\begin{aligned}
A_x &= \int_{\alpha_1}^{\alpha_2}\int_{\rho_1(\alpha)}^{\rho_2(\alpha)} \rho d\rho d\alpha \\
&= \int_{\alpha_1}^{\alpha_2} \left[\frac{\rho^2}{2}\right]_{\rho_1(\alpha)}^{\rho_2(\alpha)} d\alpha = \int_{\alpha_1}^{\alpha_2} \frac{{\rho_2}^2}{2} - \frac{{\rho_1}^2}{2} d\alpha \\
&= \frac{{\rho_0}^2}{2}\int_{\alpha_1}^{\alpha_2} 1 - \frac{(x - x_0)^2}{{\rho_0}^2\sin^2\alpha} d\alpha \\
&= \frac{{\rho_0}^2}{2}\int_{\alpha_1}^{\alpha_2} 1 - \frac{\sin^2\alpha_1}{\sin^2\alpha} d\alpha \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\alpha_1) - \frac{{\rho_0}^2}{2}\sin^2\alpha_1 \int_{\alpha_1}^{\alpha_2} \frac{1}{\sin^2\alpha} d\alpha \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\alpha_1) + \frac{{\rho_0}^2}{2}\sin^2\alpha_1 \left[\cot\alpha\right]_{\alpha_1}^{\alpha_2} \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\alpha_1) + \frac{{\rho_0}^2}{2}\sin^2\alpha_1 \left[\cot\alpha_2 - \cot\alpha_1\right].
\end{aligned} \tag{4.3}$$

The term $\cot\alpha_2 - \cot\alpha_1$ in the preceding integral can be reduced to the following:

$$\begin{aligned}\cot\alpha_2 - \cot\alpha_1 &= \frac{\cos\alpha_2}{\sin\alpha_2} - \frac{\cos\alpha_1}{\sin\alpha_1} \\ &= \frac{\sin\alpha_1\cos\alpha_2 - \sin\alpha_2\cos\alpha_1}{\sin\alpha_1\sin\alpha_2} \\ &= \frac{\sin(\alpha_1 - \alpha_2)}{\sin\alpha_1\sin\alpha_2}.\end{aligned}$$

Since, $\alpha_2 = \pi - \alpha_1$, hence $\alpha_1 - \alpha_2 = \alpha_1 - (\pi - \alpha_1) = 2\alpha_1 - \pi$, and therefore,

$$\begin{aligned}\cot\alpha_2 - \cot\alpha_1 &= \frac{\sin(\alpha_1 - \alpha_2)}{\sin\alpha_1\sin\alpha_2} \\ &= \frac{\sin(2\alpha_1 - \pi)}{\sin\alpha_1\sin\alpha_2} \\ &= \frac{-\sin(2\alpha_1)}{\sin\alpha_1\sin\alpha_2} \\ &= \frac{-2\sin\alpha_1\cos\alpha_1}{\sin\alpha_1\sin\alpha_2} = \frac{-2\cos\alpha_1}{\sin\alpha_2}.\end{aligned}$$

Since $\sin\alpha_2 = \sin\alpha_1$, we can state that:

$$\cot\alpha_2 - \cot\alpha_1 = \frac{-2\cos\alpha_1}{\sin\alpha_1}. \tag{4.4}$$

Substituting equation (4.4) into equation (4.3), we have:

$$\begin{aligned}A_x &= \frac{{\rho_0}^2}{2}(\pi - 2\alpha_1) + \frac{{\rho_0}^2}{2}\sin^2\alpha_1\left[-\frac{2\cos\alpha_1}{\sin\alpha_1}\right] \\ &= \frac{{\rho_0}^2}{2}(\pi - 2\alpha_1) + \frac{{\rho_0}^2}{2}\sin\alpha_1\left[-2\cos\alpha_1\right] \\ &= \frac{{\rho_0}^2}{2}\left[\pi - 2\alpha_1 - \sin 2\alpha_1\right].\end{aligned}$$

Therefore, A_x can be stated as:

$$A_x = \frac{\rho_0^2}{2}\left[\pi - 2\alpha_1 - \sin 2\alpha_1\right]. \tag{4.5}$$

Next, to find the intersection area A_y as the beam moves along the y-axis only, as shown in Figure 4.4, the following parameters were defined:

$$d_y = y - y_0, \tag{4.6}$$

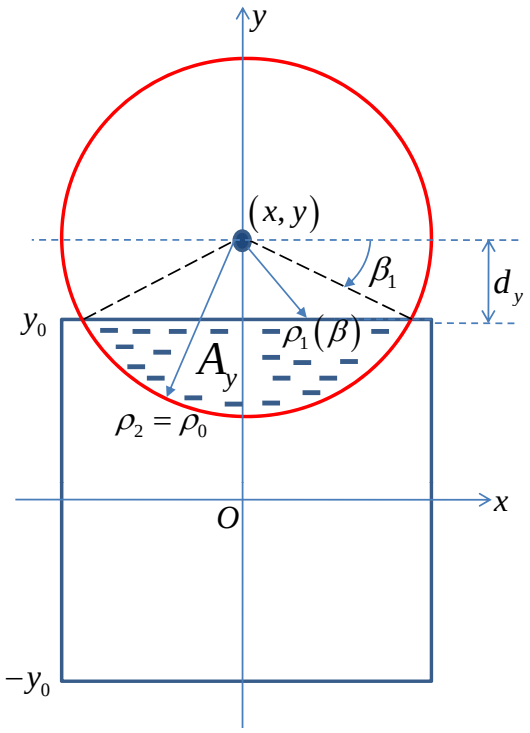

Figure 4.3: Parameters definition for area A_y.

which is the vertical distance between the spot center and the side at which the circular spot intersects the square. In addition, we have:

$$\sin\beta_1 = \sin\beta_2 = \frac{y - y_0}{\rho_0}, \tag{4.7}$$

where, β_1 is the angle between the horizontal axis crossing through the center of the beam and the first intersection between the circular spot and the square photocell. While, β_2 is the angle between the horizontal axis crossing through the beam center and the second point of intersection. Using $\beta_2 = \pi - \beta_1$, $\rho_1(\beta) = (y - y_0)/\sin\beta$, and

$\rho_2(\beta) = \rho_0$, the intersection area A_y, is evaluated as follows:

$$\begin{aligned}
A_y &= \int_{\beta_1}^{\beta_2}\int_{\rho_1(\beta)}^{\rho_2(\beta)} \rho d\rho d\beta \\
&= \int_{\beta_1}^{\beta_2} \left[\frac{\rho^2}{2}\right]_{\rho_1(\beta)}^{\rho_2(\beta)} d\beta = \int_{\beta_1}^{\beta_2} \frac{{\rho_2}^2}{2} - \frac{{\rho_1}^2}{2} d\beta \\
&= \frac{{\rho_0}^2}{2}\int_{\beta_1}^{\beta_2} 1 - \frac{(y-y_0)^2}{{\rho_0}^2 \sin^2\beta} d\beta \\
&= \frac{{\rho_0}^2}{2}\int_{\beta_1}^{\beta_2} 1 - \frac{\sin^2\beta_1}{\sin^2\beta} d\beta \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\beta_1) - \frac{{\rho_0}^2}{2}\sin^2\beta_1 \int_{\alpha_1}^{\alpha_2} \frac{1}{\sin^2\beta} d\beta \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\beta_1) + \frac{{\rho_0}^2}{2}\sin^2\beta_1 \left[\cot\beta\right]_{\beta_1}^{\beta_2} \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\beta_1) + \frac{{\rho_0}^2}{2}\sin^2\beta_1 \left[\cot\beta_2 - \cot\beta_1\right] \\
&= \frac{{\rho_0}^2}{2}(\pi - 2\beta_1) + \frac{{\rho_0}^2}{2}\sin\beta_1 \left[-2\cos\beta_1\right] \\
&= \frac{{\rho_0}^2}{2}\left[\pi - 2\beta_1 - \sin 2\beta_1\right].
\end{aligned}$$

Therefore, the area A_y can be stated as:

$$\begin{aligned}
A_y &= \int_{\beta_1}^{\beta_2}\int_{\rho_1(\beta)}^{\rho_2(\beta)} \rho\, d\rho\, d\beta \\
&= \frac{\rho_0^2}{2}\left[\pi - 2\beta_1 - \sin 2\beta_1\right].
\end{aligned} \tag{4.8}$$

To derive the area of intersection A_{xy} as the beam is moved along both the x and y axis of the photodetector plane, the parameters α_1, β_1, $d_x = x - x_0$, $d_y = y - y_0$, $\phi = \tan^{-1}(d_y/d_x)$, $\rho_x = (x - x_0)/\sin\alpha$, and $\rho_y = (y - y_0)/\sin\beta$ as shown in Figure 4.4 are first evaluated. The angle ϕ which is measured from the horizontal axis passing through the center of the beam spot to the corner of the photocell within the spot region, bisects the area A_{xy} into A_1 and A_2. To compute A_{xy}, the expressions for

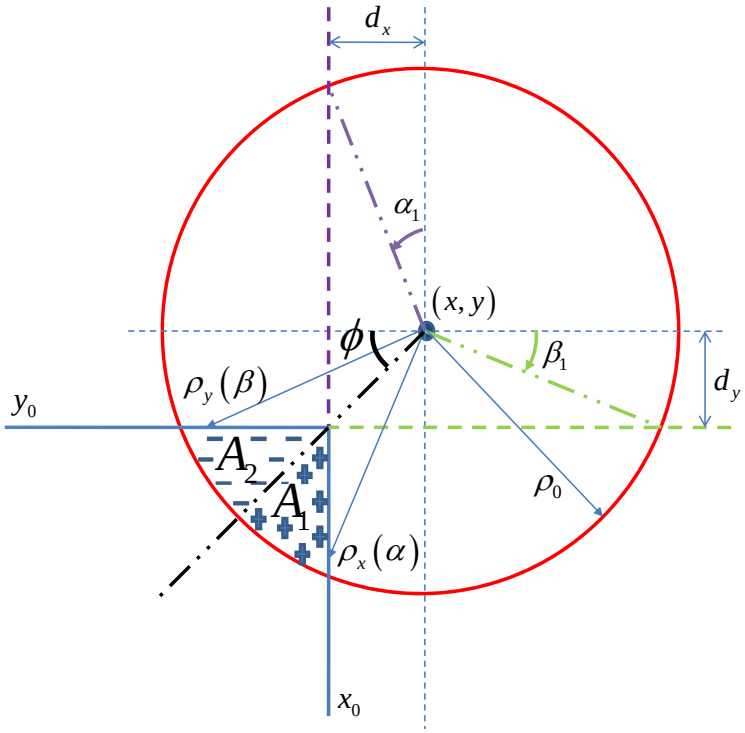

Figure 4.4: Parameters definition for area A_{xy}.

areas A_1 and A_2 are found, then the sum of both is taken.

$$
\begin{aligned}
A_1 &= \int_{\phi+\frac{\pi}{2}}^{\alpha_2}\int_{\rho_x}^{\rho_0} \rho d\rho d\alpha \\
&= \int_{\phi+\frac{\pi}{2}}^{\alpha_2}\left[\frac{\rho^2}{2}\right]_{\rho_x}^{\rho_0} d\alpha = \frac{1}{2}\int_{\phi+\frac{\pi}{2}}^{\alpha_2} {\rho_0}^2 - {\rho_x}^2 d\alpha \\
&= \frac{1}{2}\int_{\phi+\frac{\pi}{2}}^{\alpha_2} {\rho_0}^2 - \frac{(x-x_0)^2}{\sin^2\alpha} d\alpha = \frac{{\rho_0}^2}{2}\int_{\phi+\frac{\pi}{2}}^{\alpha_2} 1 - \frac{(x-x_0)^2}{{\rho_0}^2\sin^2\alpha} d\alpha \\
&= \frac{{\rho_0}^2}{2}\int_{\phi+\frac{\pi}{2}}^{\alpha_2} 1 - \frac{\sin^2\alpha_1}{\sin^2\alpha} d\alpha = \frac{{\rho_0}^2}{2}\left[\alpha_2 - \left(\phi+\frac{\pi}{2}\right)\right] - \frac{{\rho_0}^2\sin^2\alpha_1}{2}\int_{\phi+\frac{\pi}{2}}^{\alpha_2} \frac{1}{\sin^2\alpha} d\alpha \\
&= \frac{{\rho_0}^2}{2}\left[\frac{\pi}{2} - \alpha_1 - \phi\right] + \frac{{\rho_0}^2\sin^2\alpha_1}{2}\left[\cot\alpha\right]_{\phi+\frac{\pi}{2}}^{\alpha_2} \\
&= \frac{{\rho_0}^2}{2}\left[\frac{\pi}{2} - \alpha_1 - \phi\right] + \frac{{\rho_0}^2\sin^2\alpha_1}{2}\left[\frac{\cos\alpha}{\sin\alpha}\right]_{\phi+\frac{\pi}{2}}^{\alpha_2} \\
&= \frac{{\rho_0}^2}{2}\left[\frac{\pi}{2} - (\alpha_1+\phi)\right] + \frac{{\rho_0}^2\sin^2\alpha_1}{2}\left[-\frac{\cos\alpha_2}{\sin\alpha_2} + \frac{\sin(\phi+\pi/2)}{\cos(\phi+\pi/2)}\right].
\end{aligned}
\tag{4.9}
$$

Using the trignometric identities, $\sin(A \pm B) = \sin A\cos B \pm \cos A \sin B$ and $\cos(A \pm B) =$

$\cos A \cos B \mp \sin A \sin B$, we have the following relations:

$$\cos \alpha_2 = \cos (\pi - \alpha_1) = \cos \pi \cos \alpha_1 + \sin \pi \sin \alpha_1 = - \cos \alpha_1 \tag{4.10}$$

$$\sin \alpha_2 = \sin (\pi - \alpha_1) = \sin \pi \cos \alpha_1 - \cos \pi \sin \alpha_1 = \sin \alpha_1 \tag{4.11}$$

$$\cos (\phi + \pi/2) = \cos \phi \cos (\pi/2) - \sin \phi \sin (\pi/2) = - \sin \phi \tag{4.12}$$

$$\sin (\phi + \pi/2) = \sin \phi \cos (\pi/2) + \cos \phi \sin (\pi/2) = \cos \phi. \tag{4.13}$$

Substituting equations (4.10) to (4.13) into equation (4.9), we get the following:

$$\begin{aligned} A_1 &= \frac{{\rho_0}^2}{2} \left[\frac{\pi}{2} - (\alpha_1 + \phi) \right] + \frac{{\rho_0}^2 \sin^2 \alpha_1}{2} \left[- \frac{\cos \alpha_1}{\sin \alpha_1} + \frac{\sin \phi}{\cos \phi} \right] \\ &= \frac{{\rho_0}^2}{2} \left[\frac{\pi}{2} - (\alpha_1 + \phi) \right] + \frac{{\rho_0}^2 \sin^2 \alpha_1}{2} \left[- \frac{\cos (\alpha_1 + \phi)}{\sin \alpha_1 \cos \phi} \right] \\ &= \frac{{\rho_0}^2}{2} \left[\frac{\pi}{2} - (\alpha_1 + \phi) - \frac{\sin \alpha_1}{\cos \phi} \cos (\alpha_1 + \phi) \right]. \end{aligned}$$

Therefore, A_1 can be stated as:

$$A_1 = \frac{{\rho_0}^2}{2} \left[\frac{\pi}{2} - (\alpha_1 + \phi) - \frac{\sin \alpha_1}{\cos \phi} \cos (\alpha_1 + \phi) \right]. \tag{4.14}$$

Similarly, area A_2 is evaluated as follows:

$$\begin{aligned}
A_2 &= \int_{\pi-\phi}^{\beta_2}\int_{\rho_y}^{\rho_0} \rho d\rho d\beta \\
&= \int_{\pi-\phi}^{\beta_2} \left[\frac{\rho^2}{2}\right]_{\rho_y}^{\rho_o} d\beta = \frac{1}{2}\int_{\pi-\phi}^{\beta_2} {\rho_0}^2 - {\rho_y}^2 d\beta \\
&= \frac{1}{2}\int_{\pi-\phi}^{\beta_2} {\rho_0}^2 - \frac{(y-y_0)^2}{\sin^2\beta} d\beta = \frac{{\rho_0}^2}{2}\int_{\pi-\phi}^{\beta_2} 1 - \frac{(y-y_0)^2}{{\rho_0}^2\sin^2\beta} d\beta \\
&= \frac{{\rho_0}^2}{2}\int_{\pi-\phi}^{\beta_2} 1 - \frac{\sin^2\beta_1}{\sin^2\beta} d\beta = \frac{{\rho_0}^2}{2}\left[\beta_2 - (\pi-\phi)\right] - \frac{{\rho_0}^2\sin^2\beta_1}{2}\int_{\pi-\phi}^{\beta_2} \frac{1}{\sin^2\beta} d\beta \\
&= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1\right] + \frac{{\rho_0}^2\sin^2\beta_1}{2}\left[\cot\beta\right]_{\pi-\phi}^{\beta_2} \\
&= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1\right] + \frac{{\rho_0}^2\sin^2\beta_1}{2}\left[\frac{\cos\beta}{\sin\beta}\right]_{\pi-\phi}^{\beta_2} \\
&= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1\right] + \frac{{\rho_0}^2\sin^2\beta_1}{2}\left[-\frac{\cos\beta_2}{\sin\beta_2} + \frac{\cos(\pi-\phi)}{\sin(\pi-\phi)}\right], \qquad (4.15)
\end{aligned}$$

where,

$$\cos\beta_2 = \cos(\pi-\beta_1) = -\cos\beta_1 \qquad (4.16)$$

$$\sin\beta_2 = \sin(\pi-\beta_1) = \sin\beta_1 \qquad (4.17)$$

$$\cos(\pi-\phi) = \cos\pi\cos\phi + \sin\pi\sin\phi = -\cos\phi \qquad (4.18)$$

$$\sin(\pi-\phi) = \sin\pi\cos\phi - \cos\pi\sin\phi = \sin\phi \qquad (4.19)$$

Substituting equations (4.16) to (4.19) into equation (4.15) we get the following:

$$\begin{aligned}
A_2 &= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1\right] + \frac{{\rho_0}^2\sin^2\beta_1}{2}\left[-\frac{\cos\beta_1}{\sin\beta_1} + \frac{\cos\phi}{\sin\phi}\right] \\
&= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1\right] - \frac{{\rho_0}^2\sin^2\beta_1}{2}\left[\frac{\sin(\phi-\beta_1)}{\sin\beta_1\sin\phi}\right] \\
&= \frac{{\rho_0}^2}{2}\left[\phi - \beta_1 - \frac{\sin\beta_1}{\sin\phi}\sin(\phi-\beta_1)\right].
\end{aligned}$$

Therefore, A_2 can be stated as:

$$A_2 = \frac{{\rho_0}^2}{2}\left[\phi - \beta_1 - \frac{\sin\beta_1}{\sin\phi}\sin\left(\phi - \beta_1\right)\right]. \tag{4.20}$$

Hence, A_{xy} the total area of the shaded region can be written in the following closed form:

$$\begin{aligned} A_{xy} &= A_1 + A_2 \\ &= \int_{\phi+\frac{\pi}{2}}^{\alpha_2}\int_{\rho_x}^{\rho_0} \rho\, d\rho\, d\alpha + \int_{\pi-\phi}^{\beta_2}\int_{\rho_y}^{\rho_0} \rho\, d\rho\, d\beta \\ &= \frac{\rho_0^2}{2}[\frac{\pi}{2} - (\alpha_1+\beta_1) - \frac{\sin\alpha_1}{\cos\phi}\cos(\alpha_1+\phi) - \frac{\sin\beta_1}{\sin\phi}\sin(\phi-\beta_1)]. \end{aligned} \tag{4.21}$$

4.2.1 Modeling Optical Apodization

The optical power enclosed within a certain area of intersection, is derived for three different cases, the power as the beam moves along the x direction only, the y direction only, and both x–y directions. The power for horizontal motion can be evaluated using the following double integral:

$$P_x = \int_{\alpha_1}^{\alpha_2}\int_{\rho_1(\alpha)}^{\rho_2(\alpha)} I_0\left[\frac{W_0}{W(z)}\right]^2 \exp\left(\frac{-2\rho^2}{W^2(z)}\right)\rho d\rho d\alpha \tag{4.22}$$

Let,

$$I_1 = I_0\left[\frac{W_0}{W(z)}\right]^2 \tag{4.23}$$

$$\rho_2(\alpha) = \rho_0 \tag{4.24}$$

$$\rho_1(\alpha) = \frac{x - x_0}{\sin\alpha}. \tag{4.25}$$

Therefore, using the above definitions equation (4.22), can be written as:

$$P_x = I_1 \int_{\alpha_1}^{\alpha_2}\int_{\rho_1(\alpha)}^{\rho_2(\alpha)} \exp\left(\frac{-2\rho^2}{W^2(z)}\right)\rho d\rho d\alpha. \tag{4.26}$$

Let,

$$u = \frac{2}{W^2(z)}\rho^2, \tag{4.27}$$

where,

$$\begin{aligned} u_2 &= u\left(\rho_2 = \rho_0\right) = \frac{2}{W^2}{\rho_0}^2 \\ u_1 &= u\left(\rho_1 = \frac{x - x_0}{\sin\alpha}\right) = \frac{2}{W^2}\left(\frac{x - x_0}{\sin\alpha}\right)^2. \end{aligned}$$

Taking the derivative of u with respect to ρ we have:

$$du = \frac{4}{W^2(z)}\rho d\rho. \tag{4.28}$$

Substituting equations (4.27) and (4.28) into equation (4.26) and taking the limits u_2 and u_1, we have the following:

$$\begin{aligned} P_x &= I_1 \int_{\alpha_1}^{\alpha_2}\int_{u_1}^{u_2} \exp\left(-u\right)\left(\frac{W}{2}\right)^2 du d\alpha \\ &= I_1 \int_{\alpha_1}^{\alpha_2}\left(\frac{W}{2}\right)^2 \left[-\exp\left(-u\right)\right]_{u_1}^{u_2} d\alpha \\ &= I_1 \int_{\alpha_1}^{\alpha_2}\left(\frac{W}{2}\right)^2 \left[-\exp\left(-u_2\right) + \exp\left(-u_1\right)\right] d\alpha \\ &= I_1 \int_{\alpha_1}^{\alpha_2}\left(\frac{W}{2}\right)^2 \left[-\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) + \exp\left(-\frac{2}{W^2}\left(\frac{x - x_0}{\sin\alpha}\right)^2\right)\right] d\alpha \\ &= I_1\left(\frac{W}{2}\right)^2 \left\{\int_{\alpha_1}^{\alpha_2} -\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) d\alpha + \int_{\alpha_1}^{\alpha_2} \exp\left(-\frac{2}{W^2}\left(\frac{x - x_0}{\sin\alpha}\right)^2\right) d\alpha\right\} \\ &= \kappa_1 \left\{-\kappa_2\left(\alpha_2 - \alpha_1\right) + \int_{\alpha_1}^{\alpha_2} \exp\left(-\frac{\kappa_x}{\sin^2\alpha}\right) d\alpha\right\} \end{aligned}$$

Therefore, P_x can be stated as:

$$P_x = -\kappa_1\kappa_2(\alpha_2 - \alpha_1) + \kappa_1 \int_{\alpha_1}^{\alpha_2} \exp\left(-\frac{\kappa_x}{\sin^2\alpha}\right) d\alpha, \tag{4.29}$$

where,

$$\kappa_x = \frac{2}{W^2}(x - x_0)^2 \tag{4.30}$$

$$\kappa_1 = I_1\left(\frac{W}{2}\right)^2 \tag{4.31}$$

$$\kappa_2 = \exp\left(-\frac{2}{W^2}{\rho_0}^2\right). \tag{4.32}$$

Next, we investigated how the optical power intercepted by one photocell changes as the beam center moves along the x direction only. This can be divided into six different cases as follows:

if Case 1: $x \geq x_0$ **then**

$d_x = x - x_0$

$\alpha_1 = \sin^{-1}\left(\frac{d_x}{\rho_0}\right)$

and the power of the shaded region is $P = P_x$

else if Case 2: $x \leq x_0$ **then**

$d_x = -x_0 - x$

$\alpha_1 = \sin^{-1}\left(\frac{d_x}{\rho_0}\right)$

and the power of the shaded region is $P = P_x$

else if Case 3: $0 \leq x \leq x_0$ **then**

$d_x = -x_0 - x$

$\alpha_1 = \sin^{-1}\left(\frac{d_x}{\rho_0}\right)$

$P' = P_x$

and the power of the shaded region is $P = P_T - P_x$

else if Case 4: $-x_0 \leq x \leq 0$ **then**

$d_x = x + x_0$

$\alpha_1 = \sin^{-1}\left(\frac{d_x}{\rho_0}\right)$

$P' = P_x$

and the power of the shaded region is $P = P_T - P_x$

else if Case 5: $x = 0$ **then**

The power of the shaded region is $P = P_T$

else if Case 6: $x = x \geq x_0 + \rho_0$ **then**

The power is $P = 0$

Where, $P_T = 2\pi I_1 \left[\frac{W}{2}\right]^2 (1-\kappa_2)$, is the total power of the beam spot with radius 0.5 cm. The power while the beam is moved along the y direction only can be determined using the following:

$$P_y = \int_{\beta_1}^{\beta_2} \int_{\rho_1(\beta)}^{\rho_2(\beta)} I_0 \left[\frac{W_0}{W(z)}\right]^2 \exp\left(\frac{-2\rho^2}{W^2(z)}\right) \rho d\rho d\beta. \tag{4.33}$$

Provided that, $\rho_2(\beta) = \rho_0$, and $\rho_1(\beta) = (y - y_0)/\sin\beta$, a similiar procedure as P_x can be adopted to evaluate the optical power P_y as follows:

$$\begin{aligned}
P_y &= I_1 \int_{\beta_1}^{\beta_2} \int_{v_1}^{v_2} \exp(-u) \left(\frac{W}{2}\right)^2 du d\beta \\
&= I_1 \int_{\beta_1}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-u)\right]_{v_1}^{v_2} d\beta \\
&= I_1 \int_{\beta_1}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-v_2) + \exp(-v_1)\right] d\beta \\
&= I_1 \int_{\beta_1}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) + \exp\left(-\frac{2}{W^2}\left(\frac{y-y_0}{\sin\beta}\right)^2\right)\right] d\beta \\
&= I_1 \left(\frac{W}{2}\right)^2 \left\{ \int_{\beta_1}^{\beta_2} -\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) d\beta + \int_{\beta_1}^{\beta_2} \exp\left(-\frac{2}{W^2}\left(\frac{y-y_0}{\sin\beta}\right)^2\right) d\beta \right\} \\
&= \kappa_1 \left\{ -\kappa_2 (\beta_2 - \beta_1) + \int_{\beta_1}^{\beta_2} \exp\left(-\frac{\kappa_y}{\sin^2\beta}\right) d\beta \right\},
\end{aligned}$$

where,

$$\begin{aligned}
v_2 &= u(\rho_2 = \rho_0) = \frac{2}{W^2}{\rho_0}^2 \\
v_1 &= u\left(\rho_1 = \frac{y-y_0}{\sin\beta}\right) = \frac{2}{W^2}\left(\frac{y-y_0}{\sin\beta}\right)^2.
\end{aligned}$$

Therefore, the optical power P_y for vertical motion can be stated as:

$$P_y = -\kappa_1\kappa_2(\beta_2 - \beta_1) + \kappa_1 \int_{\beta_1}^{\beta_2} \exp\left(-\frac{\kappa_y}{\sin^2\beta}\right) d\beta, \tag{4.34}$$

where $\kappa_y = 2(y - y_0)^2/W^2$. Similarly, we investigated how the optical power intercepted by one photocell changes as the beam center moves along the y direction only. This can be divided into six different cases.

if Case 7: $y \geq y_0$ **then**

$d_y = y - y_0$

$\beta_1 = \sin^{-1}\left(\frac{d_y}{\rho_0}\right)$

and the power of the shaded region is $P = P_y$

else if Case 8: $y \leq y_0$ **then**

$d_y = -y_0 - y$

$\beta_1 = \sin^{-1}\left(\frac{d_y}{\rho_0}\right)$

and the power of the shaded region is $P = P_y$

else if Case 9: $0 \leq y \leq y_0$ **then**

$d_y = -y_0 - y$

$\beta_1 = \sin^{-1}\left(\frac{d_y}{\rho_0}\right)$

$P' = P_y$

and the power of the shaded region is $P = P_T - P_y$

else if Case 10: $-y_0 \leq y \leq 0$ **then**

$d_y = y + y_0$

$\beta_1 = \sin^{-1}\left(\frac{d_y}{\rho_0}\right)$

$P' = P_y$

and the power of the shaded region is $P = P_T - P_y$

else if Case 11: $y = 0$ **then**

The power of the shaded region is $P = P_T$

else if Case 12: $y = y \geq y_0 + \rho_0$ **then**

The power is $P = 0$

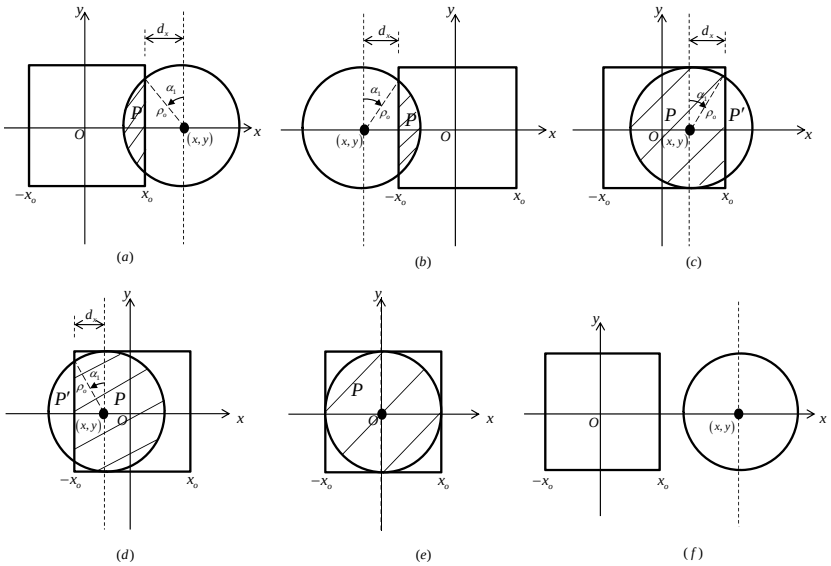

Figure 4.5: (a) Case 1: $x \geq x_0$, (b) Case 2: $x \leq x_0$, (c) Case 3: $0 \leq x \leq x_0$, (d) Case 4: $-x_0 \leq x \leq 0$, (e) Case 5: $x = 0$, (f) Case 6: $x = x \geq x_0 + \rho_0$.

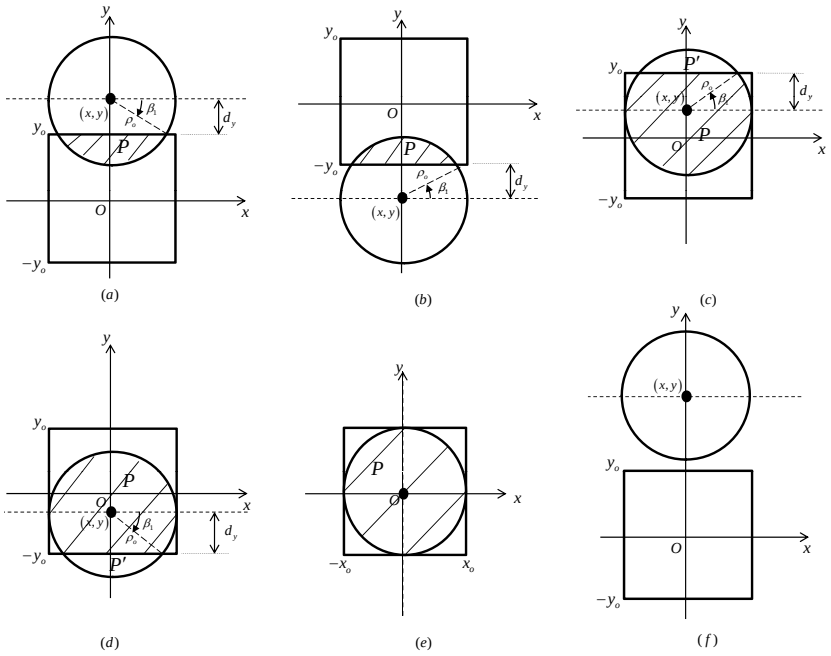

Figure 4.6: (a) Case 7: $y \geq y_0$, (b) Case 8: $y \leq y_0$, (c) Case 9: $0 \leq y \leq y_0$, (d) Case 10: $-y_0 \leq y \leq 0$, (e) Case 11: $y = 0$, (f) Case 12: $y = y \geq y_0 + \rho_0$.

Finally, as the beam is moved along both the x, and y directions over the entire plane of one photocell, the power can be stated as:

$$\begin{aligned} P_{xy} &= P_1 + P_2 && (4.35) \\ &= \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \int_{\rho_x}^{\rho_0} I(\rho, z)\, \rho\, d\rho\, d\alpha + \int_{\pi-\phi}^{\beta_2} \int_{\rho_y}^{\rho_0} I(\rho, z)\, \rho\, d\rho\, d\beta, && (4.36) \end{aligned}$$

where, $\rho_x = x - x_0/\sin\alpha$ and $\rho_y = y - y_0/\sin\beta$. To evaluate P_1 we proceed in the following manner:

$$\begin{aligned} P_1 &= \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \int_{\rho_x(\alpha)}^{\rho_0} I(\rho, z)\rho d\rho d\alpha \\ &= \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \int_{\rho_x(\alpha)}^{\rho_0} I_0 \left[\frac{W_0}{W(z)}\right]^2 \exp\left(\frac{-2\rho^2}{W^2(z)}\right) \rho d\rho d\alpha \\ &= I_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \int_{\rho_x(\alpha)}^{\rho_0} \exp\left(\frac{-2\rho^2}{W^2(z)}\right) \rho d\rho d\alpha. && (4.37) \end{aligned}$$

Substituting equations (4.27) and (4.28) into equation (4.37) and taking the

limits u_2 and u_1, we have the following:

$$\begin{aligned}
P_1 &= I_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \int_{u_1}^{u_2} \exp(-u) \left(\frac{W}{2}\right)^2 du d\alpha \\
&= I_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-u)\right]_{u_1}^{u_2} d\alpha \\
&= I_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-u_2) + \exp(-u_1)\right] d\alpha \\
&= I_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \left(\frac{W}{2}\right)^2 \left[-\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) + \exp\left(-\frac{2}{W^2}\left(\frac{x-x_0}{\sin\alpha}\right)^2\right)\right] d\alpha \\
&= I_1 \left(\frac{W}{2}\right)^2 \left\{ \int_{\phi+\frac{\pi}{2}}^{\alpha_2} -\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) d\alpha + \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \exp\left(-\frac{2}{W^2}\left(\frac{x-x_0}{\sin\alpha}\right)^2\right) d\alpha \right\} \\
&= \kappa_1 \left\{ -\kappa_2 \left(\alpha_2 - \left(\phi + \frac{\pi}{2}\right)\right) + \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \exp\left(-\frac{\kappa_x}{\sin^2\alpha}\right) d\alpha \right\}.
\end{aligned}$$

Therefore, P_1 can be stated as follows:

$$P_1 = -\kappa_1\kappa_2 \left(\alpha_2 - \left(\phi + \frac{\pi}{2}\right)\right) + \kappa_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \exp\left(-\frac{\kappa_x}{\sin^2\alpha}\right) d\alpha. \tag{4.38}$$

Next, P_2 is evaluated as follows:

$$\begin{aligned}
P_2 &= \int_{\pi-\phi}^{\beta_2} \int_{\rho_y(\beta)}^{\rho_0} I(\rho, z) \rho d\rho d\beta \\
&= \int_{\pi-\phi}^{\beta_2} \int_{\rho_y(\beta)}^{\rho_0} I_0 \left[\frac{W_0}{W(z)}\right]^2 \exp\left(\frac{-2\rho^2}{W^2(z)}\right) \rho d\rho d\beta \\
&= I_1 \int_{\pi-\phi}^{\beta_2} \int_{\rho_y(\beta)}^{\rho_0} \exp\left(\frac{-2\rho^2}{W^2(z)}\right) \rho d\rho d\beta.
\end{aligned} \tag{4.39}$$

Substituting equations (4.27) and (4.28) into equation (4.39) and taking the

limits v_2 and v_1, we have the following:

$$\begin{aligned}
P_2 &= I_1 \int_{\pi-\phi}^{\beta_2} \int_{v_1}^{v_2} \exp(-u) \left(\frac{W}{2}\right)^2 du d\beta \\
&= I_1 \int_{\pi-\phi}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-u)\right]_{v_1}^{v_2} d\beta \\
&= I_1 \int_{\pi-\phi}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp(-v_2) + \exp(-v_1)\right] d\beta \\
&= I_1 \int_{\pi-\phi}^{\beta_2} \left(\frac{W}{2}\right)^2 \left[-\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) + \exp\left(-\frac{2}{W^2}\left(\frac{y-y_0}{\sin\beta}\right)^2\right)\right] d\beta \\
&= I_1 \left(\frac{W}{2}\right)^2 \left\{ \int_{\pi-\phi}^{\beta_2} -\exp\left(-\frac{2}{W^2}{\rho_0}^2\right) d\beta + \int_{\pi-\phi}^{\beta_2} \exp\left(-\frac{2}{W^2}\left(\frac{y-y_0}{\sin\beta}\right)^2\right) d\beta \right\} \\
&= \kappa_1 \left\{ -\kappa_2 \left(\beta_2 - (\pi-\phi)\right) + \int_{\pi-\phi}^{\beta_2} \exp\left(-\frac{\kappa_y}{\sin^2\beta}\right) d\beta \right\}.
\end{aligned}$$

Therefore, P_2 can be stated as follows:

$$P_2 = -\kappa_1\kappa_2 \left(\beta_2 - (\pi-\phi)\right) + \kappa_1 \int_{\pi-\phi}^{\beta_2} \exp\left(-\frac{\kappa_y}{\sin^2\beta}\right) d\beta. \tag{4.40}$$

Substituting equations (4.38) and (4.40) into equation (4.36), P_{xy} can be written in the following form:

$$\begin{aligned}
P_{xy} &= -\kappa_1\kappa_2 \left[\tfrac{\pi}{2} - (\alpha_1+\beta_1)\right] + \kappa_1 \int_{\phi+\frac{\pi}{2}}^{\alpha_2} \exp\left(-\tfrac{\kappa_x}{\sin^2\alpha}\right) d\alpha \\
&\quad + \kappa_1 \int_{\pi-\phi}^{\beta_2} \exp\left(-\tfrac{\kappa_y}{\sin^2\beta}\right) d\beta.
\end{aligned} \tag{4.41}$$

The integrals in the preceding equations were evaluated numerically.

Using equations (4.29), (4.34), and (4.41) we developed an algorithm to determine the power P if the center of the circular beam spot is to be moved along any random path.

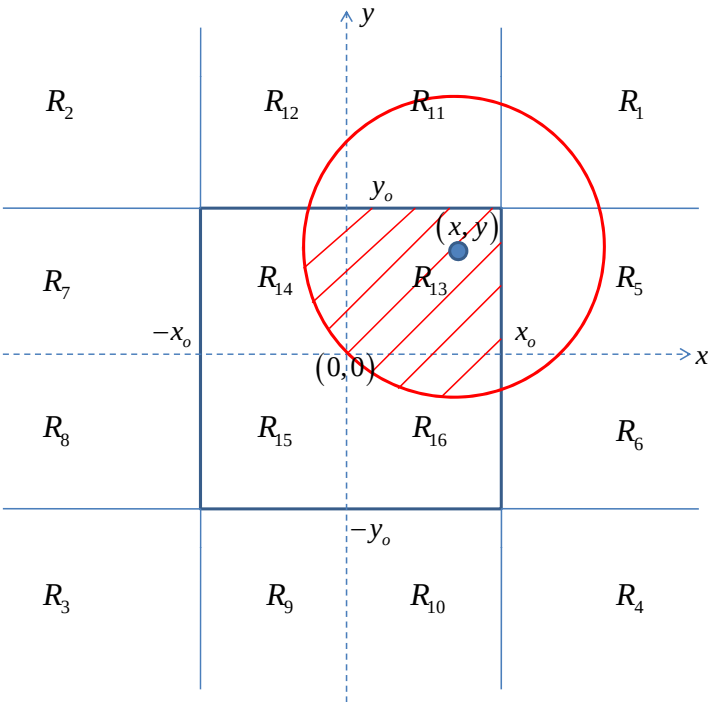

Figure 4.7: Example of beam center position as it scans the photocell's regions.

Region	x	y	d_x	d_y
R_1	$x > x_0$	$y > y_0$	$x - x_0$	$y - y_0$
R_2	$x < -x_0$	$y > y_0$	$-x - x_0$	$y - y_0$
R_3	$x < -x_0$	$y < -y_0$	$-x - x_0$	$-y - y_0$
R_4	$x > x_0$	$y < -y_0$	$x - x_0$	$-y - y_0$
R_5	$x \geq x_0$	$0 < y \leq y_0$	$x - x_0$	$y_0 - y$
R_6	$x \geq x_0$	$-y_0 \leq y < 0$	$x - x_0$	$y_0 + y$
R_7	$x \leq -x_0$	$0 < y \leq y_0$	$-x - x_0$	$y_0 - y$
R_8	$x \leq -x_0$	$-y_0 \leq y < 0$	$-x - x_0$	$y_0 + y$
R_9	$0 < x \leq x_0$	$y \leq -y_0$	$x_0 - x$	$-y - y_0$
R_{10}	$-x_0 \leq x < 0$	$y \leq -y_0$	$x_0 + x$	$-y - y_0$
R_{11}	$0 < x \leq x_0$	$y \geq y_0$	$x_0 - x$	$y - y_0$
R_{12}	$-x_0 \leq x < 0$	$y \geq y_0$	$x_0 + x$	$y - y_0$
R_{13}	$0 < x < x_0$	$0 < y < y_0$	$x_0 - x$	$y_0 - y$
R_{14}	$-x_0 < x < 0$	$0 < y < y_0$	$x_0 + x$	$y_0 - y$
R_{15}	$-x_0 < x < 0$	$-y_0 < y < 0$	$x_0 + x$	$y_0 + y$
R_{16}	$0 < x < x_0$	$-y_0 < y < 0$	$x_0 - x$	$y_0 + y$

Table 4.1: Table showing the range of each region.

To do this, the plane for one photocell has been divided into 16 different regions as shown in Figure 4.7. The power is computed depending on the region where the center of the beam is located.

if $(x, y) \in R_1, R_2, R_3$, or R_4 **then**
The power of the shaded region is $P = P_{xy}$
else if $(x, y) \in R_5, R_6, R_7$, or R_8 **then**
Check the following:
if $N_c = 1$ and $N_{int} = 2$ **then**
Check the following:
if $d_x \neq 0$ and $d_y \neq 0$ or $d_x \neq 0$ and $d_y = 0$ or $d_x = 0$ and $d_y \neq 0$ **then**
$P = P_x - P_{xy}$
else if $d_x = 0$ and $d_y = 0$ **then**
The power of the shaded region is $P = \frac{1}{4}P_T$.
else if $N_c = 0$ and $N_{int} = 2$ **then**
The power of the shaded region is $P = P_x$.
end if
else if $(x, y) \in R_9, R_{10}, R_{11}$, or R_{12} **then**
Check the following:
if $N_c = 1$ and $N_{int} = 2$ **then**
Check the following:
if $d_x \neq 0$ and $d_y \neq 0$ or $d_x \neq 0$ and $d_y = 0$ or $d_x = 0$ and $d_y \neq 0$ **then**
The power of the shaded region is $P = P_y - P_{xy}$.
else if $d_x = 0$ and $d_y = 0$ **then**
The power of the shaded region is $P = \frac{1}{4}P_T$.
end if
else if $N_c = 0$ and $N_{int} = 2$ **then**
The power of the shaded region is $P = P_y$.
end if
else if $(x, y) \in R_{13}, R_{14}, R_{15}$, or R_{16} **then**
Check the following:
if $N_c = 1$ and $N_{int} = 2$ **then**
The power of the shaded region is $P = P_T - (P_x + P_y - P_{xy})$.

else if $N_c = 0$ and $N_{int} = 4$ **then**

The power of the shaded region is $P = P_T - (P_x + P_y)$.

end if

end if

4.2.2 Modeling System Imperfections

The quadcell array of photodetectors has been modeled according to the orientation shown in Figure 4.8. Each photocell is represented as a 1 cm×1 cm square, with centers S_1, S_2, S_3, and S_4. The photocells are separated by a small horizontal distance ϵ and a vertical distance δ. In this case, the origin of the absolute Cartesian coordinate system is located at the center of the array. The coordinates of S_1, S_2, S_3 and S_4 with respect to the origin are:

$$S_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.5 + \varepsilon/2 \\ 0.5 + \delta/2 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -0.5 - \varepsilon/2 \\ 0.5 + \delta/2 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} -0.5 - \varepsilon/2 \\ -0.5 - \delta/2 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0.5 + \varepsilon/2 \\ -0.5 - \delta/2 \end{bmatrix}$$

The main objective of the optical acquisition model is to obtain the optical power distribution generated at each photodetector while the beam is scanned throughout the entire quadcell plane, thus an algorithm has been developed to calculate the optical power for each cell. The minimum resolution that can be generated using our algorithm is 0.01 cm for both the x and y positions.

The mathematical model mentioned in the previous discussion has been obtained while considering the origin of the x–y coordinate system situated at the center of one photocell. Therefore, to evaluate the optical power for the quad-cell using equations (4.29), (4.34), and (4.41), the origin of the absolute

coordinate system will be translated by a certain vector determined by the coordinates of the center of the cell. The portion of power captured by each cell is calculated. To find the power distribution for cell 1 in the array, the translational operation $(x, y) - (x_1, y_1)$ is first made. Then the earlier power calculations are carried out on photocell 1. A similar translation is applied to the second, third and fourth cell, where the power distribution is evaluated for each cell separately. Figures 4.9 to 4.12 show the normalized power distributions for each photocell, assuming ϵ and δ to be negligible.

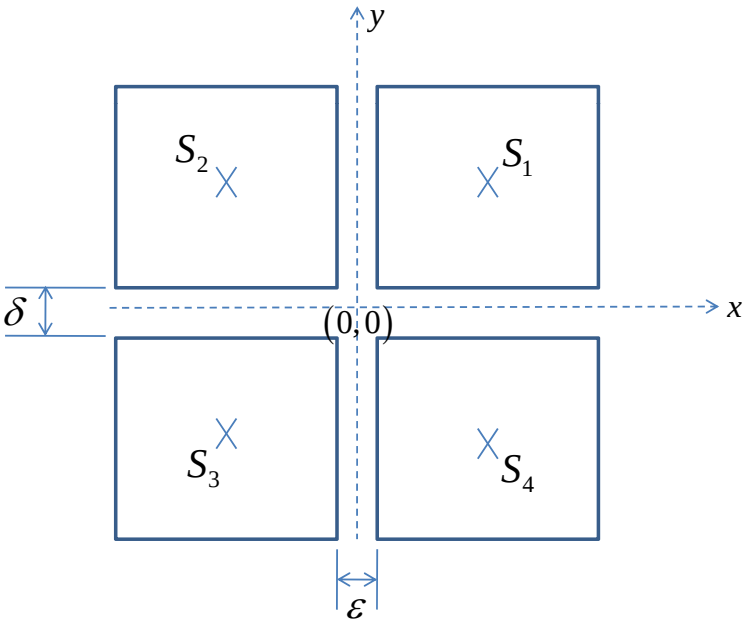

Figure 4.8: Quadcell array of photodetectors.

Furthermore, the center of the beam has been simulated to move on the plane of photodetectors along the trajectory $y = x$. A plot of the optical power distribution for the four cells against x_c (the x position of the beam center), following the given trajectory can be shown in Figure 4.14, where ϵ and δ are first set to zero. As illustrated, the beam occupies most of the active surface areas of photocells 1 and 3, and as it moves away from photocell 3 and into the vicinity of photocell 1, cells 2 and 4 start detecting a portion of the optical power. All four cells will ideally detect equal powers when the centroid of the beam is at the origin of the plane. Since the values for ϵ and δ are technically not equal to zero, their effect on the overall power distribution was investigated.

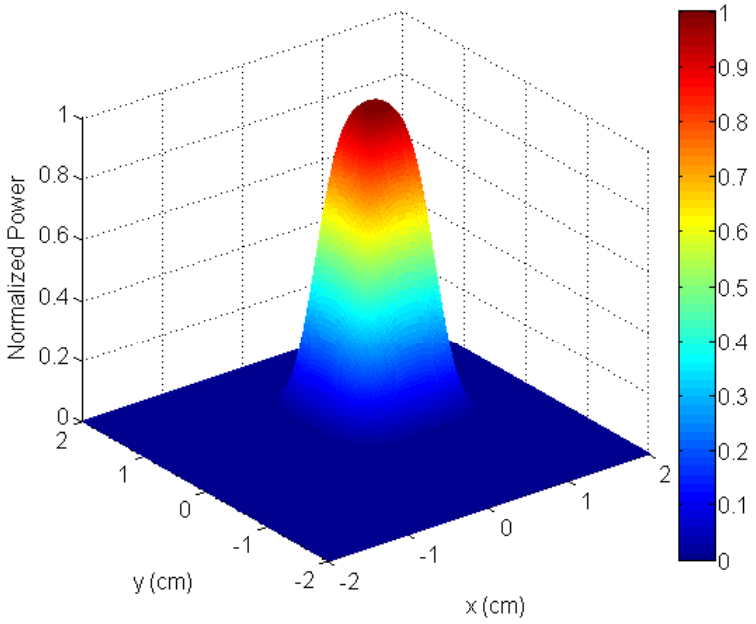

Figure 4.9: The normalized power obtained by photocell 1, as the beam center scans the quadcell plane.

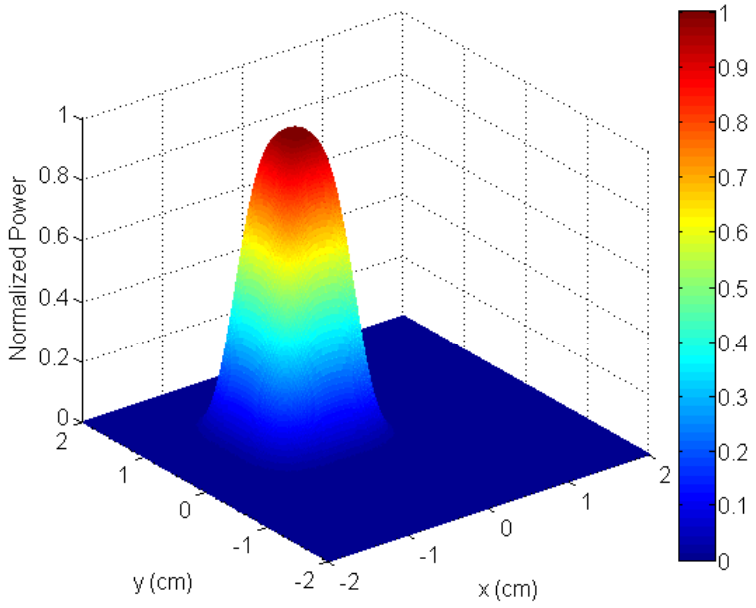

Figure 4.10: The normalized power obtained by photocell 2, as the beam center scans the quadcell plane.

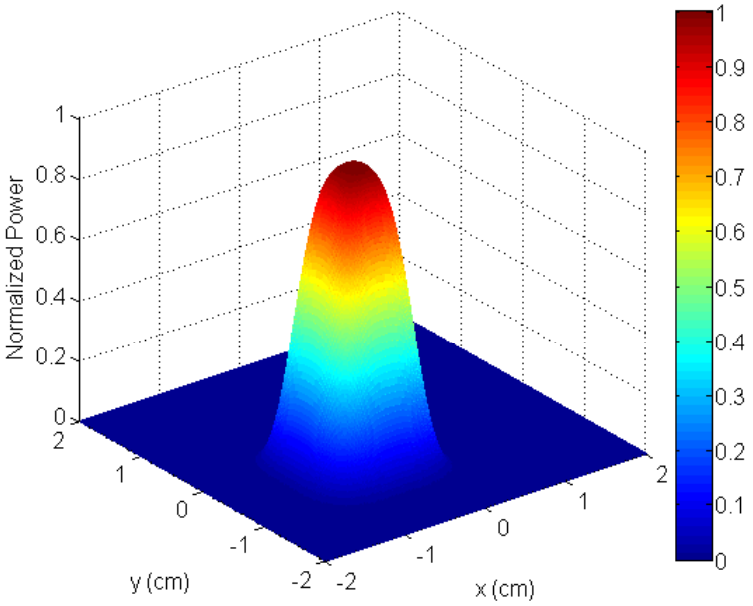

Figure 4.11: The normalized power obtained by photocell 3, as the beam center scans the quadcell plane.

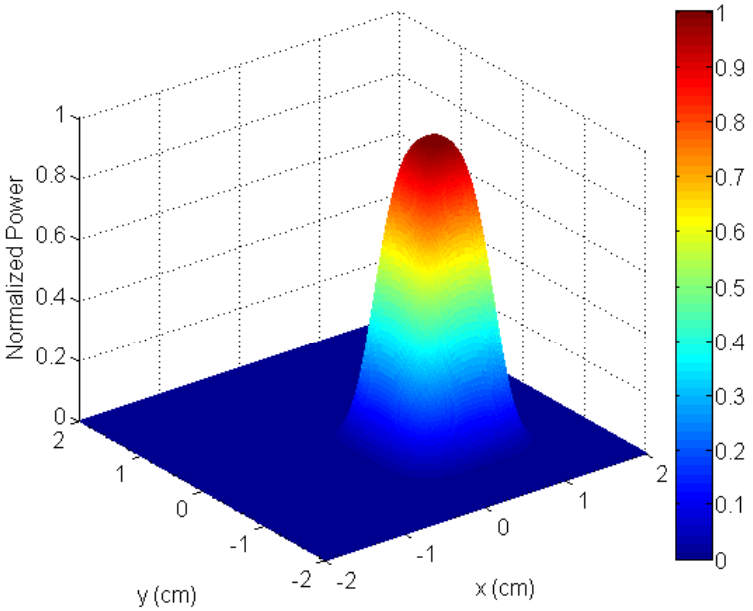

Figure 4.12: The normalized power obtained by photocell 4, as the beam center scans the quadcell plane.

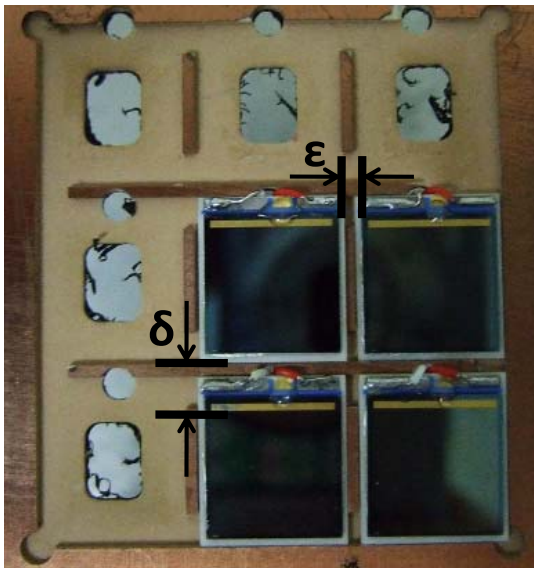

Figure 4.13: Quadcell arrangement for the experimental setup showing different ϵ, and δ.

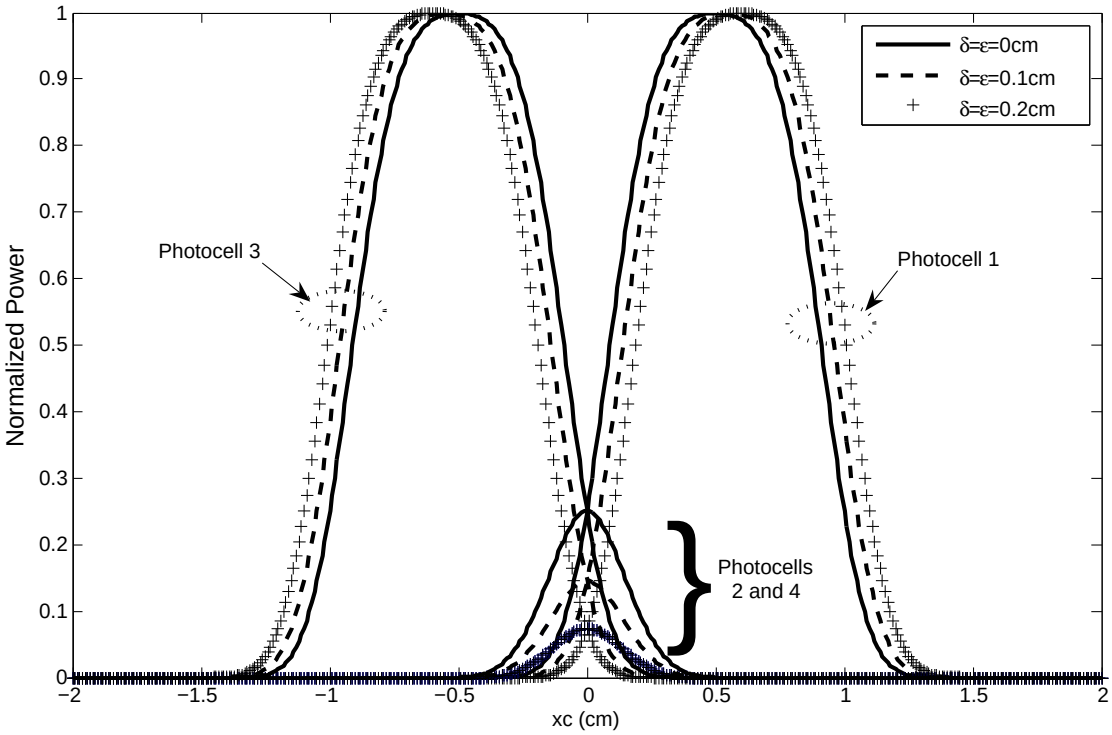

Figure 4.14: Variation of the optical power detected by all four photodetectors as the beam is moved along $y = x$ line for different values of ϵ, and δ.

Another simulation has been run, when $\epsilon = \delta = 0.1$ cm and 0.2 cm. As shown in Figure 4.14, the maximas of the normalized power waveforms for the first and third photocells shifted to their new center positions, at (0.55,0.55) and $(-0.55,-0.55)$ for $\epsilon = \delta = 0.1$ cm, and at (0.6,0.6) and $(-0.6,-0.6)$ for $\epsilon = \delta =$ 0.2 cm. In addition, the power at the origin drops exponentially as ϵ and δ become larger. This effect has been demonstrated in Figure 4.15. The value for ϵ was varied from 0 to 0.9 cm in steps of 0.1 cm. The normalized power at one of the photocells has a maximum value of 0.25 at $\epsilon = \delta = 0$ cm. With a spacing of 0.1 cm, the power at the origin of the plane dropped by 42.0% from its maximum value. For $\epsilon = \delta = 0.2$ cm, we have a 70.4% power drop.

In the previous discussion, ϵ and δ were equal, however the design of the experimental setup inevitably defies such an assumption. As shown in Figure 4.13, the photocells are placed on a metallic plate with substrates of size 1.1 cm$\times$1.3 cm. The spacing between them is 1 mm. Thorlabs FDS1010 photodiodes have been used, where the Si detector is mounted on a 0.45"$\times$0.52" (1.1 cm$\times$1.3 cm) ceramic wafer with an anode and a cathode. The active area for the FDS1010 is about 9.7 mm$\times$9.7 mm [33]. Taking into account the width of the inactive region of the photocell, would add an extra 0.2 cm to the vertical spacing between the photodiodes. Therefore, the experimental values for ϵ, and δ can be approximated to 0.1 cm and 0.3 cm. Using those values in the simulation, gave us a normalized power of 6.90e$-$02 at the origin of the quadcell plane. This is about 72.4% power drop when compared to the ideal case and 52.5% power drop relative to the case when $\epsilon = \delta = 0.1$ cm.

4.3 Experimental Study of the Position Detector

4.3.1 Experimental Setup

A laboratory prototype of the optical acquisition system has been implemented to verify the results obtained through simulation and to idenitify the performance of the WN with experimental testing data.

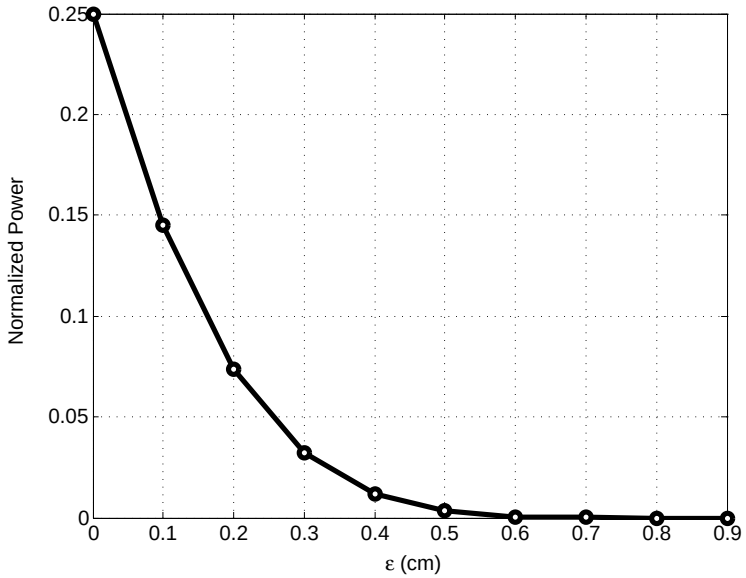

Figure 4.15: Plot of the normalized power for photocell 1 vs. ϵ, when the beam center is at the origin of the quadcell plane. ϵ and δ are assumed to be equal.

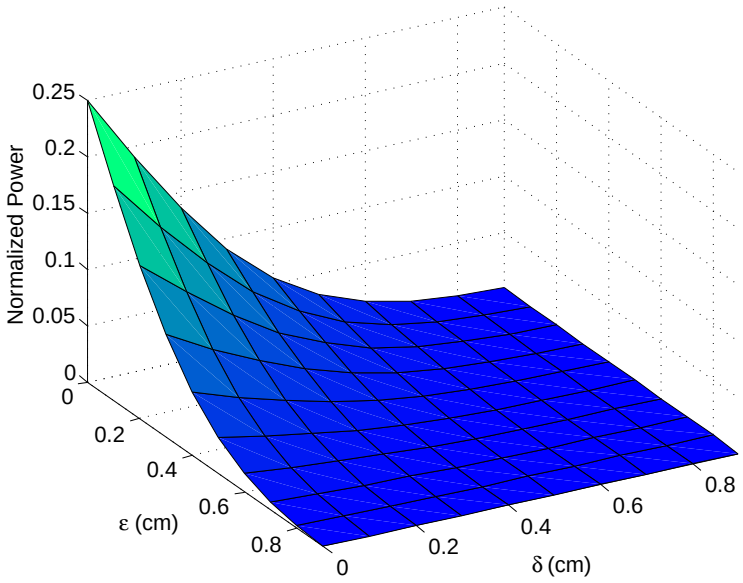

Figure 4.16: Plot of normalized power for photocell 1 vs. ϵ and δ, when the beam centroid is at the origin of the quadcell plane.

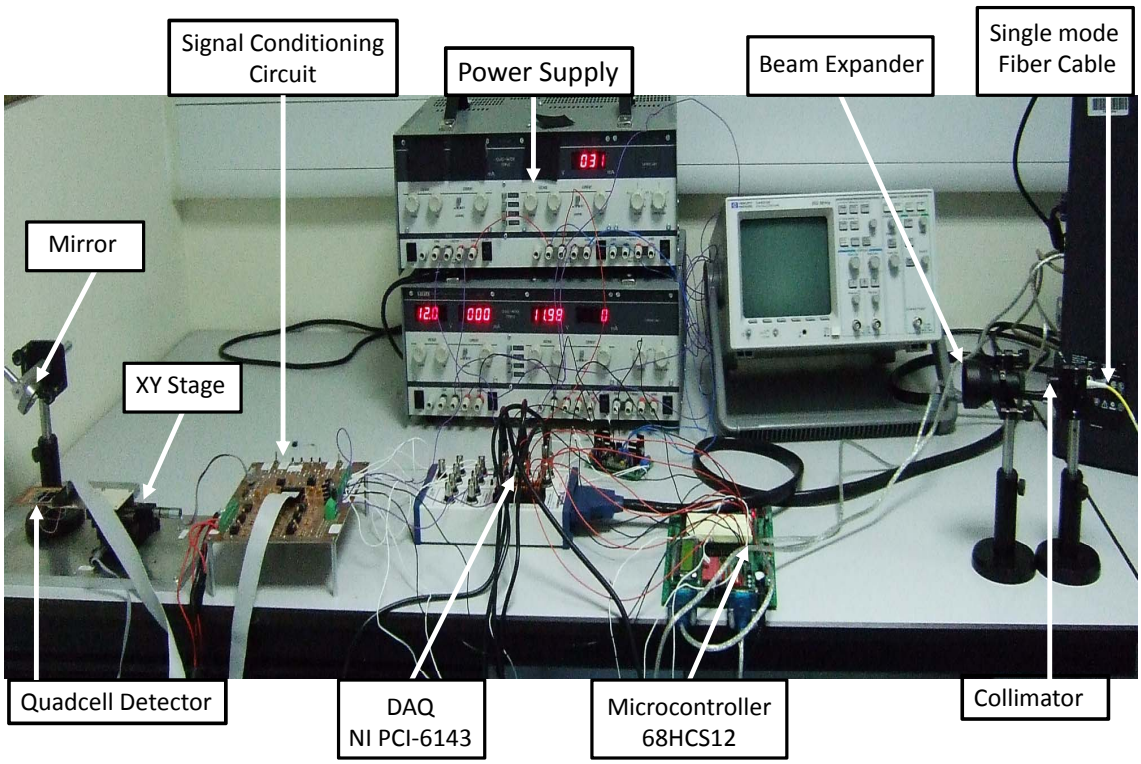

Figure 4.17: Experimental prototype of the optical power acquisition system.

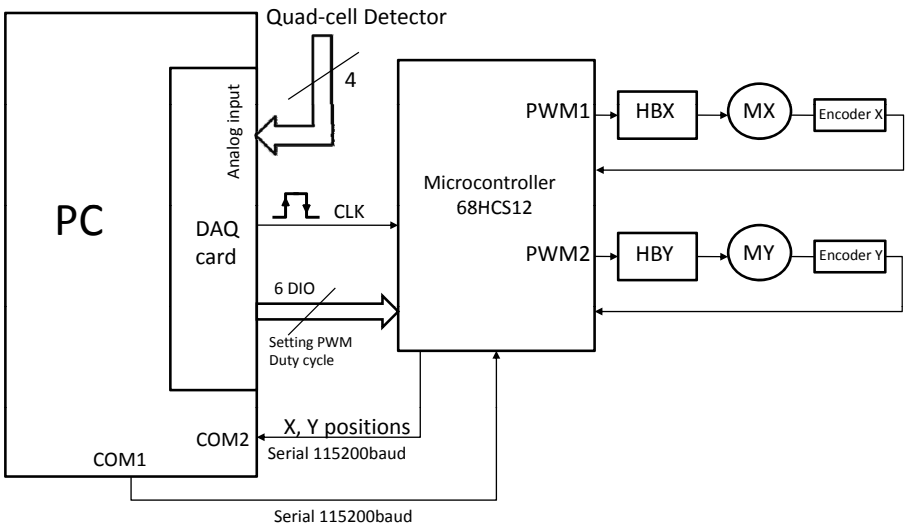

Figure 4.18: Block diagram for photo-voltage acquisiton and position measurement system (HBX: H-bridge for motor X; HBY: H-bridge for motor Y; MX: motor X; MY: motor Y; PWM1: pulse width modulated signal fed in to motor X; PWM2: pulse width modulated signal fed in to motor Y; COM1: communication port 1; COM2: communication port 2; CLK: clock to synchronize photo-voltage acquisition, position measurement; DIO: digital input/output channels; PC: personal computer).

The experimental setup in Figure 4.17 can be divided into three main parts; the optical setup and XY stage as shown in Figures 4.19 and 4.21 and the data acquisition system shown in Figure 4.18. The transmitter optics consists of a single mode optical fiber cable, where one of its ends is connected to a He-Ne laser source and the other end is coupled to a collimator. The optical fiber cable acts as a spatial filter at a wavelength of 633 nm, which removes higher modes so that only the fundamental mode is left. A collimator (F260FC-B, Thorlabs) is used with a 633 nm alignment wavelength, where the lens is manufactured to be one focal length away from the output end of the fiber. The laser beam is then projected into a Galilean Beam Expander (BE03M, Thorlabs) which is used to adjust the beam diameter to be the same size as one side of a photocell. On the other hand, the receiver optics is composed of a mirror which reflects the beam onto the quadcell detector. Each photodiode (FDS1010, Thorlabs) is connected to an RC noise filter with a cut-off frequency of about 10 kHz, and an amplification circuitry as shown in Figure 4.21 [33]. The output voltages of the photodiodes are fed into four simultaneously sampled analog input channels of a BNC shielded connecter block (NI BNC-2110, National Instruments) for the Data Acquisition Card (NI PCI-6143, National Instruments), which also provides a 16-bit resolution, and a sampling rate of up to 250 kS/s per channel.

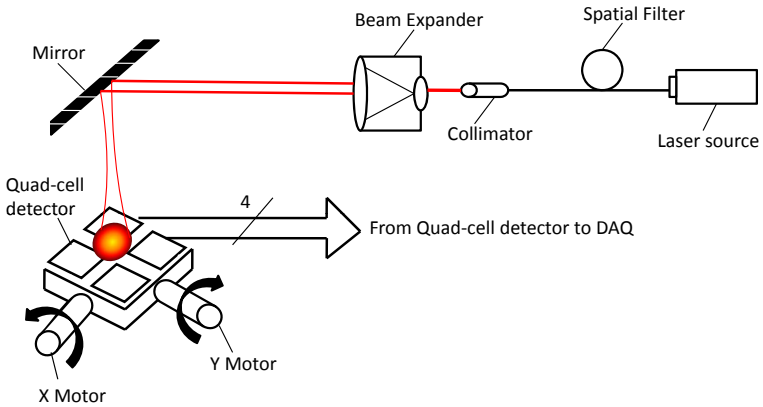

Figure 4.19: Optical setup of the system.

The quadcell array has been mounted on XY motorized linear translation stage (T25XY, Thorlabs) fitted with incremental encoders to measure the x and y positions of the center of the beam. The T25XY stage provides a travel range of about 2.5 cm and utilizes two 12 V DC servomotors with a 256:1 gear reduction

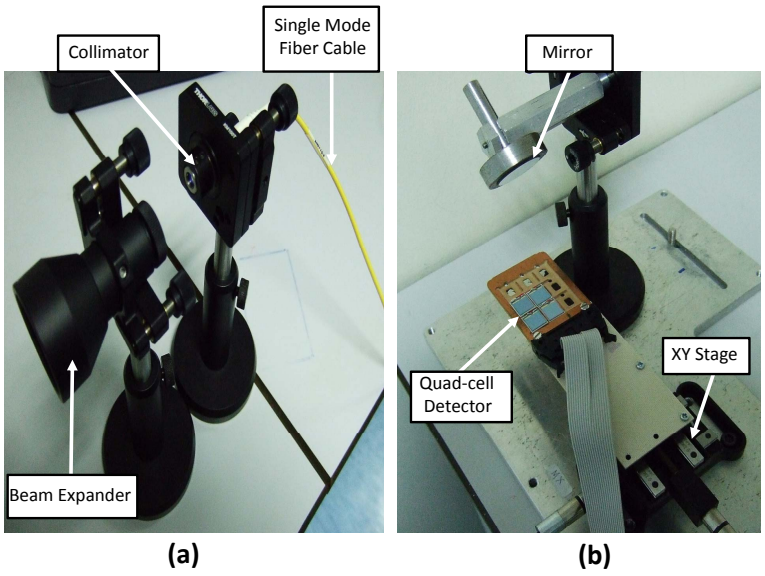

Figure 4.20: (a)Transmission optics.(b)Reception optics.

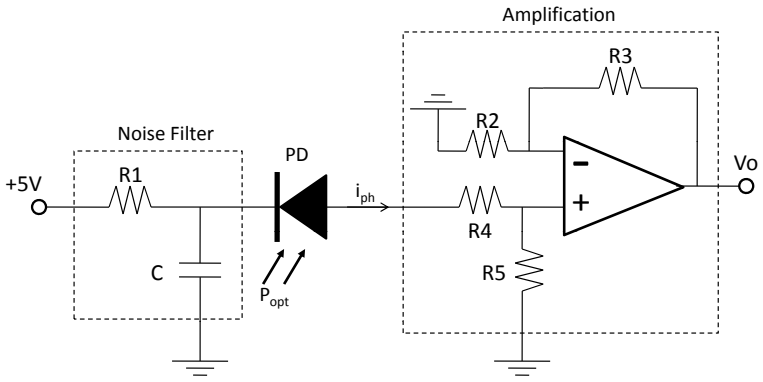

Figure 4.21: Signal conditioning circuitry for photodiode, involving amplification, and noise removal.

head [33]. Since one revolution of the high precision XY stage corresponds to a linear translation of 0.5 mm, the minimum vibration displacement that can be achieved is about 4.07e−05 mm which is less than 0.0001 mm. Furthermore, two PWM signals with a frequency of 330 Hz and 50% duty cycle, generated by a Dragon 12 (68HCS12) microcontroller by means of a Dual H-bridge, are used to drive the X and Y motors. The 48 pts/rev rotary encoder signals are fed back to the microcontroller to obtain the position measurements. The x and y positions are sent to the PC serially with a baud rate of 115.2 kbits/s. The power and position measurements are acquired synchronously every 1 s, for one complete travel range of the Y motor while keeping the X motor fixed at a certain distance.

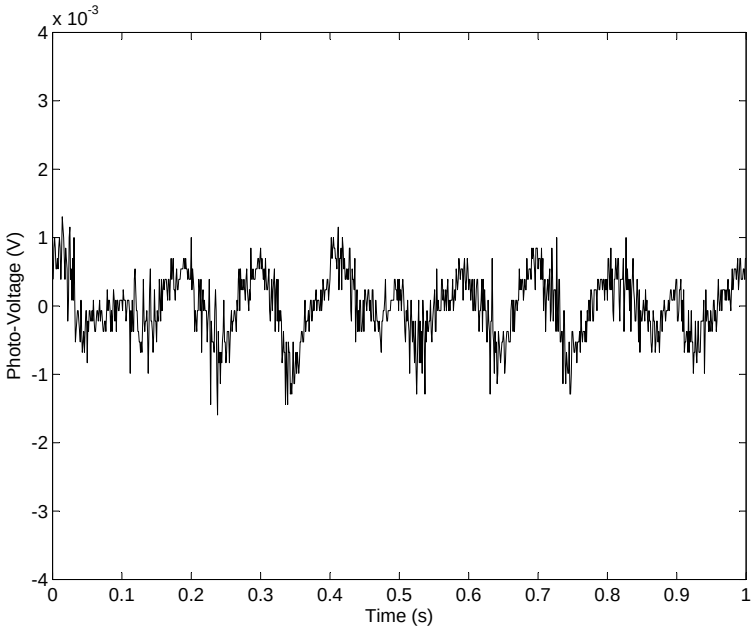

Figure 4.22: Plot of photocell output voltage in darkness.

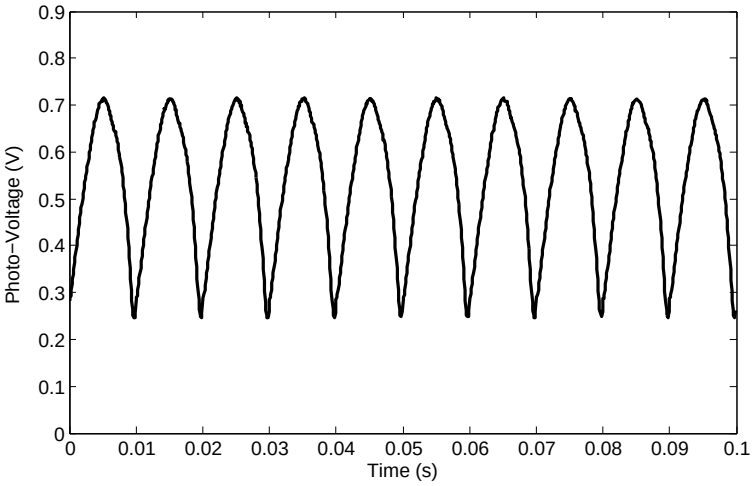

Figure 4.23: Plot of photocell output voltage in ambient light, when no laser beam is applied.

4.3.2 Optical Model Validation

Using the data illustrated in Figure 4.22 the calculated RMS (Root Mean Square) value for the output voltage of a photocell in darkness is about; 4.63e−004 V. To avoid the effect of ambient light on the photocells' output voltage as shown in Figure 4.23, the experiment is conducted in a dark room. Under ambient light conditions, the photodiodes' readings should be adjusted to account for additional induced photocurrent. Five different runs were car-

ried out, by scanning the beam spot along the y axis of the quadcell plane, while fixing the x position of the beam center at -1.05 cm, -0.55 cm, 0 cm, 0.55 cm, and 1.05 cm. By referring to Figure 4.8 and given that $\epsilon = 0.1$ cm, the beam spot was first scanned vertically along the left sides of photocells 2 and 3, where the x position of the beam center is -1.05 cm. The second scan was made along the vertical axis passing through the centers S_2 and S_3 of photocells 2 and 3, where $x = -0.55$ cm. Furthermore, a vertical scan at $x =$ 0 cm along the y axis of the array plane was carried out. The last two runs were made along the vertical line $x = 0.55$ cm passing through the centers S_1 and S_4, and along $x = 1.05$ cm passing through the right sides of photocells 1 and 4. The output photo-voltage is directly proportional to the optical power sensed by a photocell. The output voltage is derived as:

$$V_0 = A_v P_{\text{opt}} \Re \left(R_4 + R_5 \right), \tag{4.42}$$

where A_v is the amplification gain, P_{opt} is the incident optical power, and $R_4 + R_5$ is the load resistance. While $\Re$, provides an estimate of the amount of photocurrent i_{ph} expected at a certain wavelength λ.

$$i_{\text{ph}} = P_{\text{opt}} \Re. \tag{4.43}$$

According to typical responsivity curve for photodiode FDS1010 using Thorlabs calibration services, we have a $\Re \approx 0.35$ A/W for a wavelength of 633 nm [33]. Therefore, given a linear relation between the photocells' optical power, and acquired output voltage, the experimental normalized voltage measurements were compared to the theoretical model obtained through simulation by setting ϵ and δ to 0.1 cm and 0.3 cm.

At $x = 1.05$ cm, theoretically only photocell 1 will be detecting power as the y position of the beam center $y \geq 0.35$ cm and maximum power which is half of the total normalized power is detected at $y = 0.65$ cm. Photocell 4 starts to pick up power for $y < 0.35$ cm and captures half of the power at $y = -0.65$ cm. According to Figure 4.24, the maximum normalized voltage for photocell 1 was reached at $y = 0.71$ cm with a voltage percentage error of

3.72% from the theoretical maximum value. On the other hand, photocell 4 acquired maximum voltage at $y = -0.77$ cm with a voltage percentage error of 3.86%.

In the case where x is fixed to -1.05 cm, only photocell 2 will be detecting power for $y \geq 0.35$ cm, while the power reaches its peak value at $y = 0.65$ cm. In addition, according to the theoretical results photocell 3 detects optical power for $y < 0.35$ cm and acquires half of the power at $y = -0.65$ cm. Practically, by referring to Figure 4.25, the peak normalized voltage for photocell 2 was detected at $y = 0.67$ cm with a voltage percentage deviation of 8.85%. The peak normalized voltage for photocell 3 was reached at $y = -0.74$ cm and a percentage error of about 5.9% from the theoretical normalized voltage results.

At $x = 0.55$ cm as shown in Figure 4.26, the maximum optical power detected for photocell 1 is at $y = 0.65$ cm, however when compared to the experimental results, the peak value is detected at $y = 0.66$ cm, which gives an approximate percentage error of 2.03% in the y position of the beam center. For photocell 4 maximum optical power is attained theoretically at $y = -0.65$ cm, while the measured y position for the beam center was $y = -0.74$ cm, hence a percentage error of about 13.3%.

In Figure 4.27, at $x = -0.55$ cm, the maximum power obtained through experiment for photocell 2 was at $y = 0.68$ cm, that is a percentage error of 4.60% in the y position of the beam center and for photocell 3 the maximum power obtained was at $y = -0.77$ cm, thus giving us an error of 18.1%.

At $x = 0$ cm, the maximum theoretical normalized optical power is 0.3811 for all photocells, where photocells 1 and 2 attain it at $y = 0.65$ cm, and photocells 3 and 4 reach it at $y = -0.65$ cm. As can be seen from Figure 4.28, photocells 1 and 2 gained maximum optical power at $y = 0.83$ cm, and $y = 0.70$ cm. The voltage percentage error for the preceding cases was 20.7% for photocell 1 and 26.2% for photocell 2. Photocells 3 and 4 gained maximum optical power at $y = -0.42$ cm and $y = -0.74$ cm, where their relative voltage percentage errors were evaluated to be 28.8% and 23.3%.

Figures 4.24, 4.25 and 4.28 demonstrate discrepancies between the theoretical and experimental results in the values of the photo-voltage or optical power

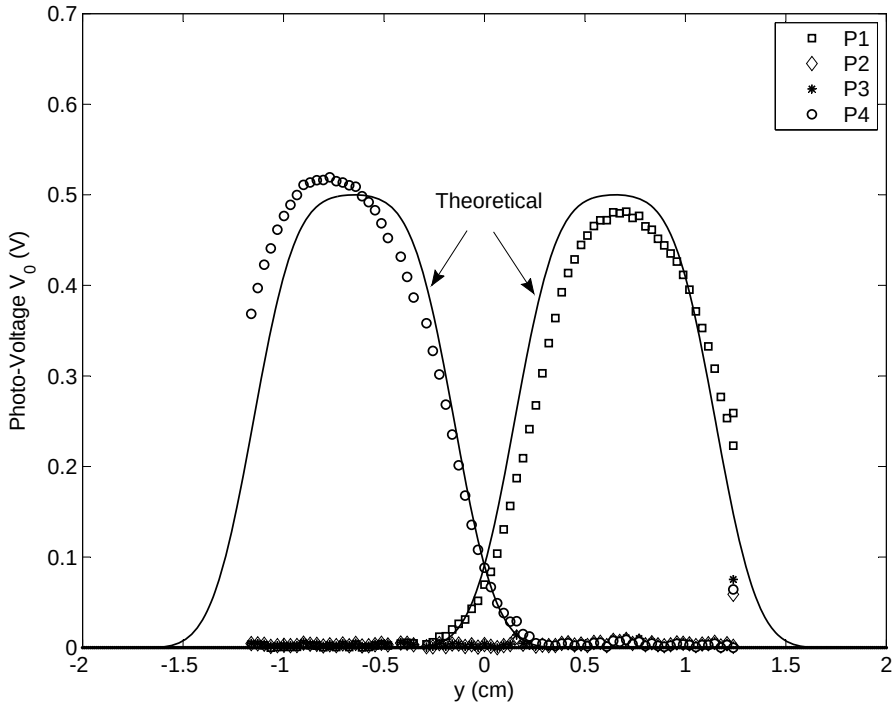

Figure 4.24: Plot of photocell output voltage vs. y position of the center of the beam while setting the x position at 1.05 cm.

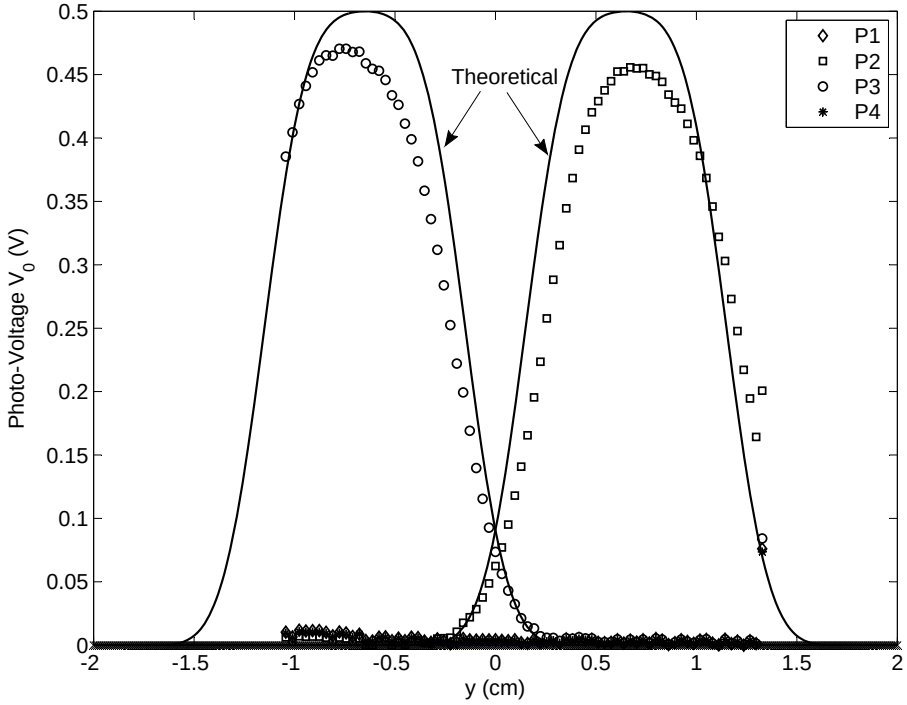

Figure 4.25: Plot of photocell output voltage vs. y position of the center of the beam while setting the x position at $-$ 1.05 cm.

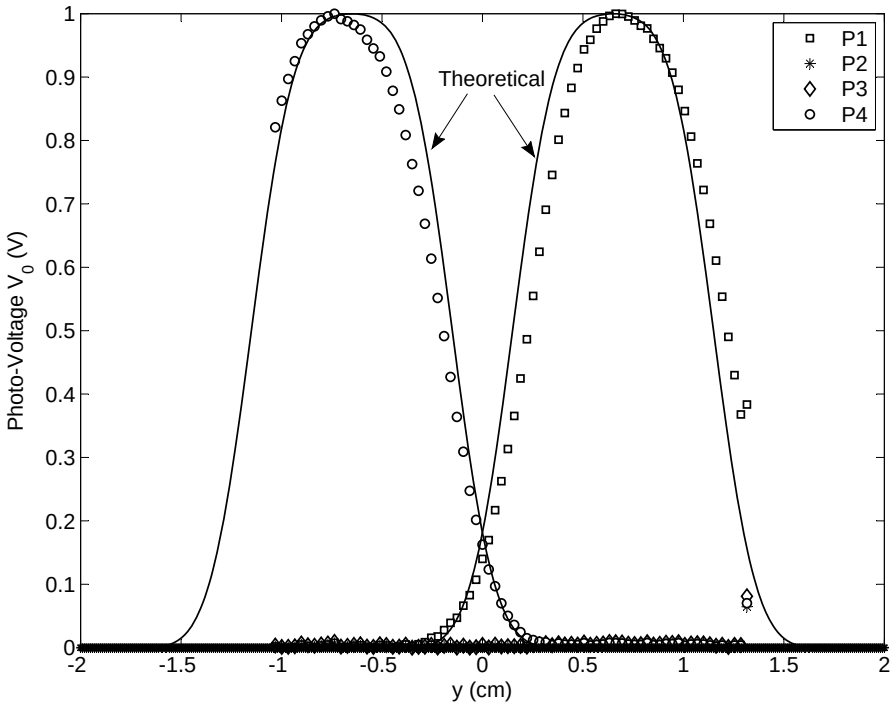

Figure 4.26: Plot of photocell output voltage vs. y position of the center of the beam while setting the x position at 0.55 cm.

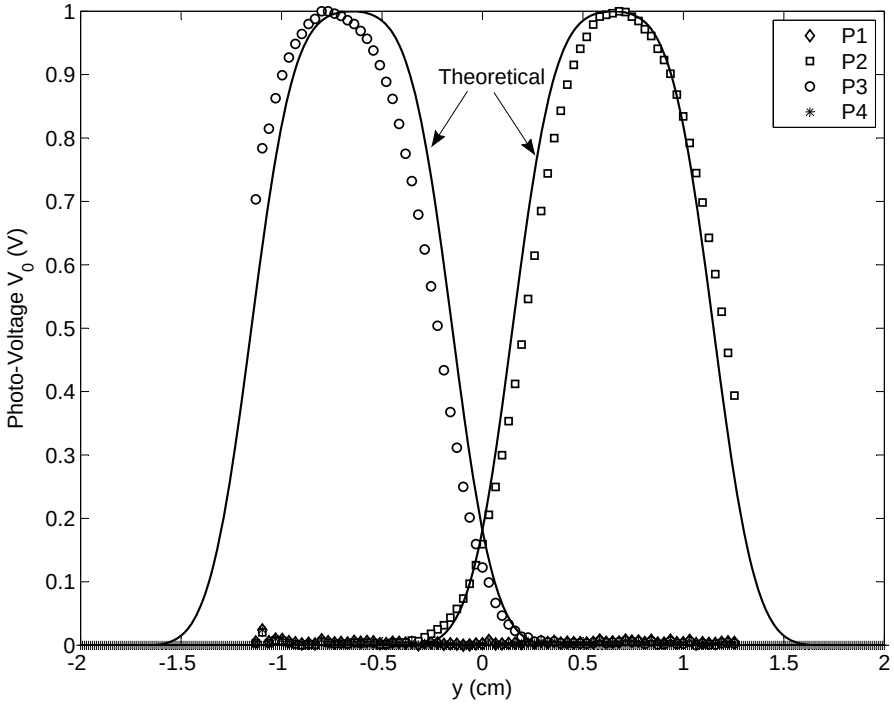

Figure 4.27: Plot of photocell output voltage vs. y position of the center of the beam while setting the x position at -0.55 cm.

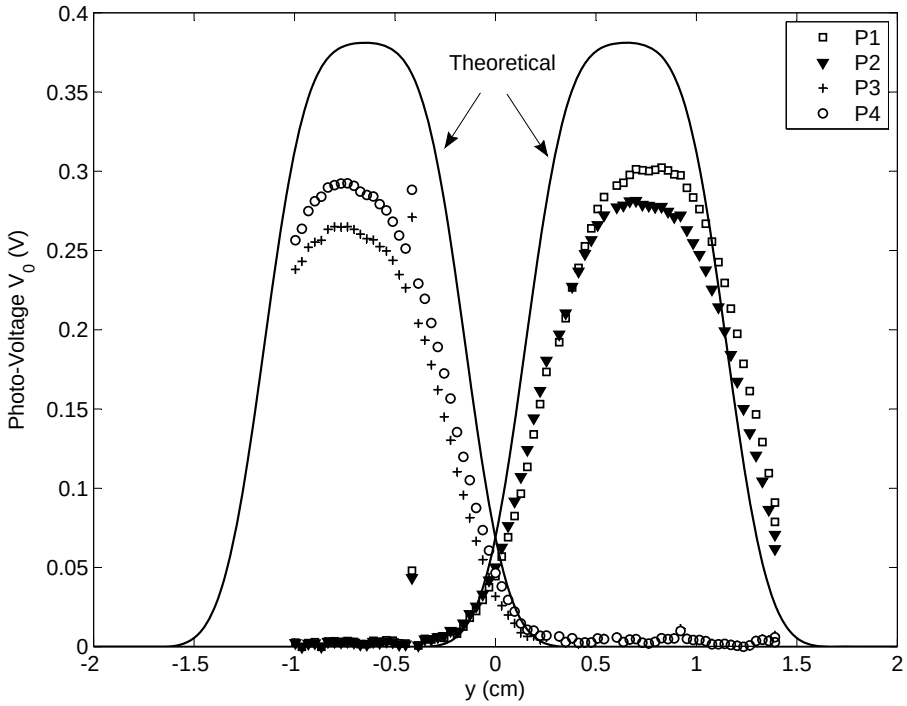

Figure 4.28: Plot of photocell output voltage vs. y position of the center of the beam while setting the x position at 0 cm.

and the y position of the beam center. However, Figures 4.26 and 4.27 have only shown a slight shift in the y position. To scan the beam spot along the y axis of the quadcell detector at a fixed x position, an initialization point has been selected such that the beam center is located at the upper right corner of photocell 1. This initial position of the beam center is adjusted manually until the normalized power acquired by photocell 1 is 0.25 ± 0.1. However, with this setting, we have a position uncertainty of $\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2}$, where 0.01 cm $\leqslant \Delta s \leqslant$ 0.1 cm. This leads to inaccuracies in both the x and y positions of the beam center as can be seen in Figure 4.29. A shift in the x and y positions of the beam center can cause a deviation between the measured voltage and the corresponding theoretical normalized spatial optical power distribution.

In theory, the beam diameter is assumed to be exactly equal to the side of the square photodetector, and 99% of the optical power is contained within a beam spot of radius $1.5W(z)$, where $W(z)$ is the beam waist at the photodetector plane. These parameters are not necessarily guaranteed in the experiment. Practically, the beam radius was adjusted manually by gradually increasing it using the beam expander, until the beam spot fits into the active area of one

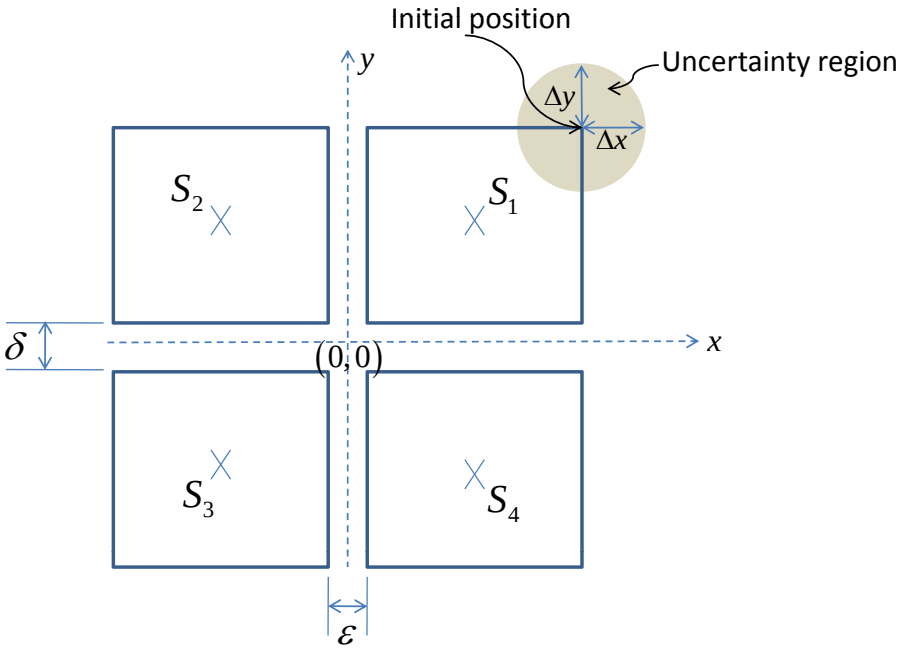

Figure 4.29: Uncertainity region when initializing the position of the beam center.

photocell. The radius of the beam spot will be 0.5 cm$-\Delta\rho$ cm where less than 99% of the optical power is contained, thus causing a reduction in the optical power obtained in the experiment and that found theoretically.

In addition, the other source of error between the theoretical and practical results in terms of the shift in the y position and the difference in optical powers, is due to having $\varepsilon = 0.1$ cm $+\Delta\varepsilon$ cm and $\delta = 0.3$ cm $+\Delta\delta$ cm. The errors $\Delta\varepsilon$ cm and $\Delta\delta$ cm are added distances due to the thickness of the ceramic wafer surrounding the Si detector, and the fact that the active area for FDS1010 is about 9.7 mm$\times$9.7 mm which is less than the assumed theoretical active area of 1 cm$\times$1 cm. Since the quad-cell detector is placed on a metallic plate, there will be inevitable optical diffraction and reflections which cause a difference between the theoretical and measured power, especially along the scan at $x = 0$ cm where the beam center is passed entirely along the metallic gap. Moreover, imperfections in the horizontal alignment of the quad-cell detector plane and slight asymmetry around the center of the sensor at the receiver optics can be additional sources of error.

5

Position Detection using Wavelet Network

The proposed position detection system is based on a wavelet network. As shown in Figure 5.1 the vector P is input to the network and the output is an estimated position vector $\hat{Y}$ which is compared to the desired position vector Y. Accordingly the error between them is used to retune the WN, in order to achieve a better match. In this chapter, we report the results obtained from training the developed WN using theoretical model with and without gaps and using the experimental data for testing. A Matlab code was built to run the WN training and testing. In order to detect the position of the center of the laser beam, the WN acts as a function approximation tool where its outputs $\hat{x}^p$ and $\hat{y}^p$ are estimates of the desired positions x^p and y^p, while the corresponding power distribution obtained from the quadcell $P^p = [P_1^p, P_2^p, P_3^p, P_4^p]$ are input to the network. The preceding superscript p represents the training pattern

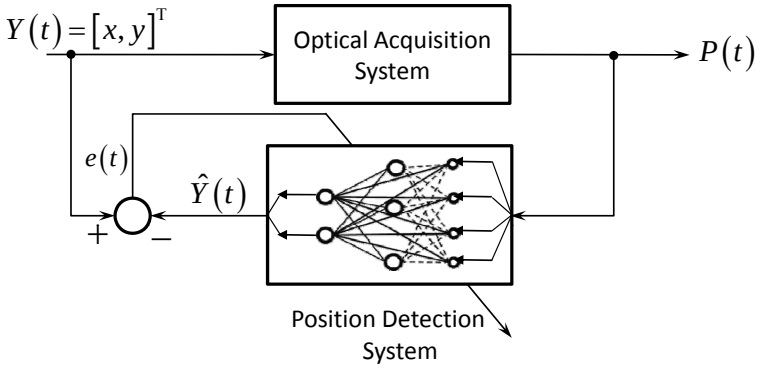

Figure 5.1: Position detection system block diagram.

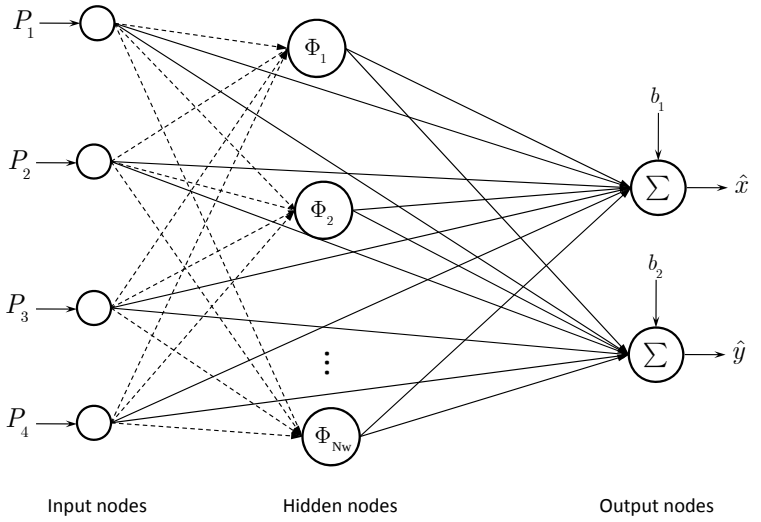

Figure 5.2: Wavelet network structure for the position detection problem.

number as described earlier in Chapter 2. Batch processing is used in off-line training of a total of N_p patterns. The feed-forward matrix equation for our WN, shown in Figure 5.2 can be stated as follows:

$$\hat{Y}^p = \begin{bmatrix} \hat{x}^p \\ \hat{y}^p \end{bmatrix} = (\mathrm{W_{oh}} * \Phi^p) + (\mathrm{W_{oi}} * \mathrm{I}^p)\,, \tag{5.1}$$

where the vectors:

$$\mathrm{I}^p = \begin{bmatrix} 1 & P_1^p & P_2^p & P_3^p & P_4^p \end{bmatrix}^{\mathrm{T}},$$

and,

$$\Phi^p = \begin{bmatrix} \Phi_1^p & \Phi_2^p & \cdots & \Phi_j^p & \cdots & \Phi_{N_w}^p \end{bmatrix}^{\mathrm{T}}.$$

The matrix $\mathrm{W_{oi}}$ consists of the direct linear coefficients between the input and output layers:

$$\mathrm{W_{oi}} = \begin{bmatrix} b_1 & a_{11} & a_{12} & a_{13} & a_{14} \\ b_2 & a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}. \tag{5.2}$$

While the matrix $\mathrm{W_{oh}}$ consists of the weights c_{kj} between the hidden and output layers:

$$\mathrm{W_{oh}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & . & . & . & c_{1N_w} \\ c_{21} & c_{22} & c_{23} & . & . & . & c_{2N_w} \end{bmatrix}. \tag{5.3}$$

5.1 Network Initialization

The initialization procedure adopted in this project is the one proposed by Zhang and Benveniste in [23]. To initialize the dilation and translation parameters the input domain of the signal is divided into a dyadic grid of the form shown in Figure 5.3. This grid has its foundation on the use of the first derivative of the Gaussian wavelet and it is a non-orthogonal grid, since the support of the wavelet used at a given dilation is higher than the translation step at this dilation [19]. The total number of wavelons available in the network depend on the selected number of levels (different dilations), $N_w = 2^{\mathrm{level}} - 1$. For the multi-dimensional case we handle the dyadic decomposition method on each input coordinate separately. Furthermore, the weights of the direct connections b_k and a_{ki} are estimated using the standard least squares method:

$$\hat{\theta}_{LS} = \left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}Y, \tag{5.4}$$

where X is the matrix of input vectors and Y is the target output training sequence over all the patterns. The weights c_{kj} associated with the wavelet activation functions are initially set to zero. For a more detailed explaination

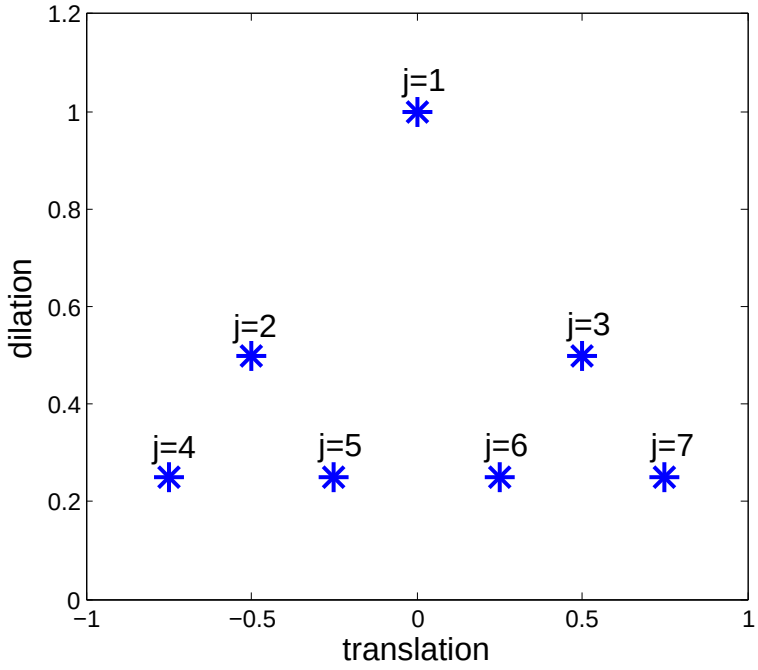

Figure 5.3: Dyadic grid for wavelet network Initialization.

of the least squares method refer to Appendix B.

5.2 Training and Testing of Wavelet Network

First, the theoretical model without gaps has been used to train the wavelet network for different values of N_w, while both the desired x and y patterns range between -2 cm and 2 cm. The values of μ and γ were set to 0.0001 and 0.9999. The plot of the MSE vs. number of learning iterations, for $N_w = 127$, $N_w = 255$, and $N_w = 511$, is shown in Figure 5.4. Due to the availability of non-unique patterns using this data set (that is input patterns with the same power values, giving different x and y output patterns) the MSE demonstrated a chaotic, non-converging behavior as the number of iterations increased. The MSE at $N_w = 127$ wavelons, has the lowest drop at 2.595, then suddenly reaches the highest peak at 2.637 after which it fluctuates around 2.6. We observe the least fluctuation at $N_w = 511$, where the MSE saturates to a value of about 2.619 at 10000 learning iterations.

In an attempt to force the MSE to converge, the training data set has been reduced by removing the patterns where none of the four photocells is detecting power as well as removing the patterns where only one photocell is detecting

power, by applying the following condition:

$$\left\{ \begin{array}{l} (P_1 = 0 \cap P_2 = 0 \cap P_3 = 0 \cap P_4 = 0) \\ \cup (P_1 \neq 0 \cap P_2 = 0 \cap P_3 = 0 \cap P_4 = 0) \\ \cup (P_1 = 0 \cap P_2 \neq 0 \cap P_3 = 0 \cap P_4 = 0) \\ \cup (P_1 = 0 \cap P_2 = 0 \cap P_3 \neq 0 \cap P_4 = 0) \\ \cup (P_1 = 0 \cap P_2 = 0 \cap P_3 = 0 \cap P_4 \neq 0) \end{array} \right\}' \tag{5.5}$$

Figure 5.5 shows the plot of the MSE vs. the number of learning iterations after applying condition 5.5 to the training data, for $N_w = 15$ and $N_w = 127$, while setting μ and γ to 0.1 and 0.9. Using these settings the MSE starts dropping from a value of about 0.530 for both curves, then experiences a highly fluctuating behavior after reaching a value of 0.446 at iteration number 1151 for $N_w = 127$, and a value of about 0.473 at iteration number 1729 for $N_w = 15$.

In order to achieve better results, the number of observations has been restricted to those where all four photocells are detecting power, therefore the preprocessing uniqueness condition:

$$P_1 \neq 0 \cap P_2 \neq 0 \cap P_3 \neq 0 \cap P_4 \neq 0, \tag{5.6}$$

has been imposed on the input data. After applying such a condition on the theoretical data without gaps, the MSE demonstrated a much better converging behavior as the number of learning epochs increased, as shown in Figure 5.6. In addition, the inverse relation between the MSE values and the number of wavelons N_w is clearly evident, where the MSE is decaying at a faster rate as N_w is increased. As illustrated in Figure 5.6, after 100,000 iterations the MSE reached the values 1.86e−03 for $N_w = 3$, 1.22e−03 for $N_w = 7$, 7.92e−04 for $N_w = 15$, and 5.94−04 for both $N_w = 63$ and $N_w = 127$.

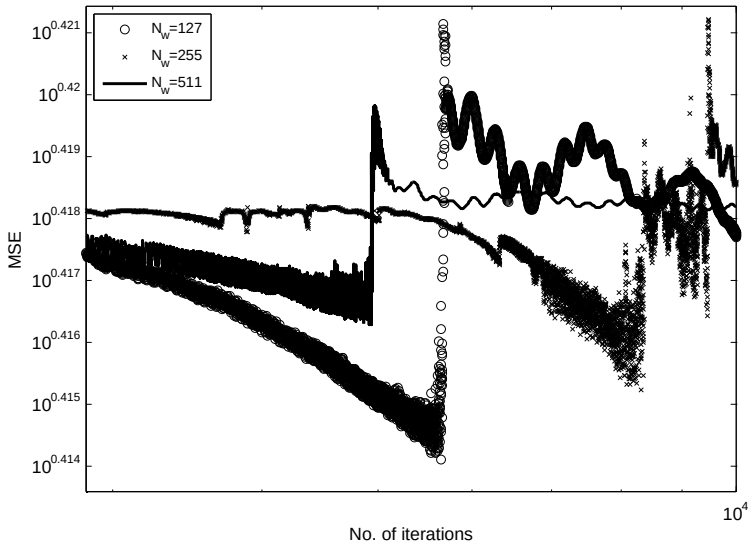

Figure 5.4: Plot of MSE vs. Iterations for different values of N_w, where the theoretical model without gaps is used for training the WN, μ= 0.0001, γ= 0.9999, no preprocessing condition is applied on the data.

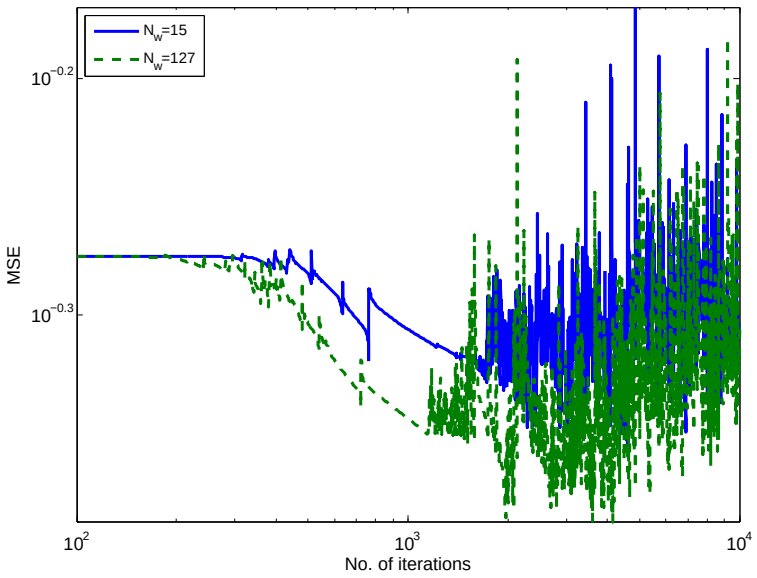

Figure 5.5: Plot of MSE vs. Iterations for different values of N_w, where the theoretical model without gaps is used for training the WN, μ= 0.1, γ= 0.9, preprocessing condition in equation 5.5 applied on the data.

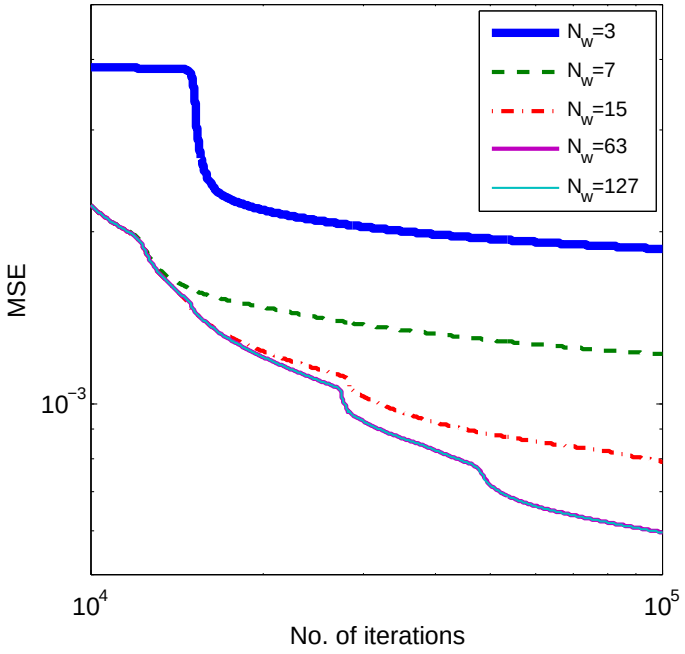

Figure 5.6: Plot of MSE vs. Iterations for different values of N_w, where the theoretical model without gaps is used for training the WN, $\mu = 0.1$, $\gamma = 0.9$.

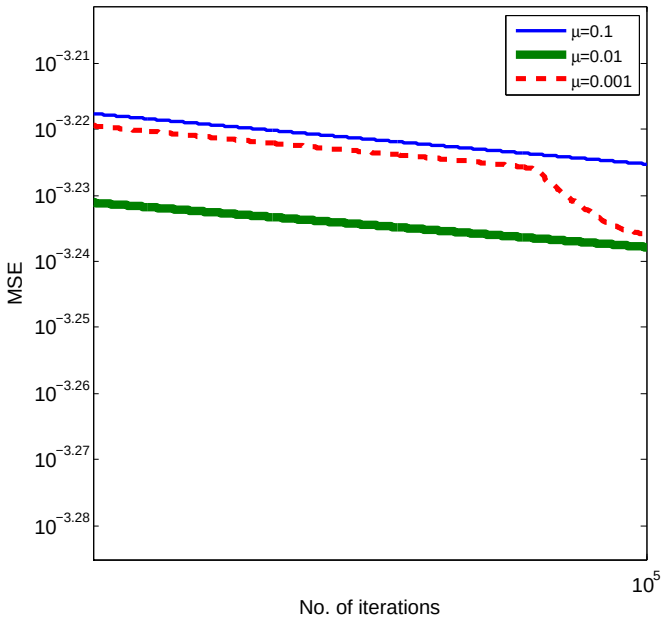

Figure 5.7: Plot of MSE vs. Iterations for different values of μ, where the simulated data without gaps is used for training the wavelet network, $N_w = 63$

To investigate the dependency of the MSE on μ, we first set N_w to 63 instead of 127 wavelons, since both achieve the lowest MSE as shown in Figure 5.6 and to reduce the complexity of the network structure as well as run time of the code. The WN is trained with $\mu = 0.1$, $\mu = 0.01$, and $\mu = 0.001$. As shown in Figure 5.7 the lowest MSE equal to 5.78e−04 was achieved at $\mu = 0.01$ for 100,000 iterations. It can be also observed that the slope of the MSE at $\mu = 0.001$ is relatively similar to that at $\mu = 0.1$, however it encounters a sudden change at iteration number 9800, thus becoming steeper and closely interluding the slope of the MSE at $\mu = 0.01$. After 100,000 iterations the MSE reaches a value of 5.95e−04 for $\mu = 0.1$ and a value of 5.81e−04 for $\mu = 0.001$.

To assess the performance of the WN, the data set at $x = 0$ cm has been removed from the training patterns, and preserved as testing data. In this case, the theoretical model with vertical gap $\delta = 0.3$ cm and horizontal gap $\epsilon = 0.1$ cm, has been used for training. The resolution for the x position was set to 0.1 cm, and that for the y position was set to 0.02 cm. Figure 5.8 shows the training error and the test error, after training the network for 100,000 iterations, for different values of N_w, while keeping μ and γ fixed at 0.01 and 0.99 respectively. Generally, in this regime, the test error is relatively higher than the training error. At the other extreme of too few hidden wavelons, the network does not have enough parameters to fit the training data well and again the test error is high. Thus, we seek an intermediate number of wavelons where a low test error will be attained. As can be seen in Figure 5.8, we have a minimum test MSE of 3.45e−03, at $N_w = 15$ and the corresponding training MSE has a value of 2.82e−04. As demonstrated in Figures 5.9 and 5.10, a better match has been achieved between the WN test output $\hat{y}$ and the theoretical data, than the WN test output $\hat{x}$ which is slightly oscillating about the position $x = 0$ cm. In addition, Figure 5.11 shows plots of the normalized optical power measured by the four photocells vs. the theoretical data y, and the WN test output $\hat{y}$, when scanning the beam center vertically along $x = 0$ cm.

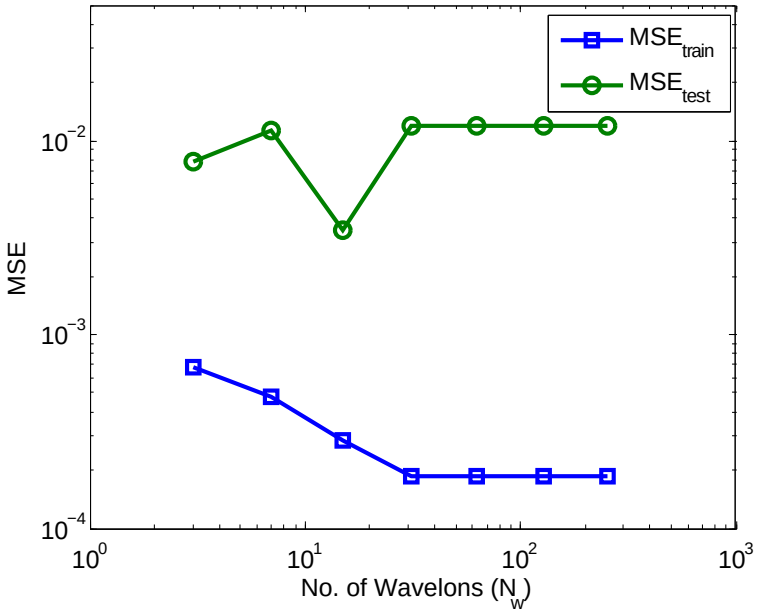

Figure 5.8: Comparing the MSE values vs. N_w after training the theoretical model with gaps and after testing for the theoretical data set at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

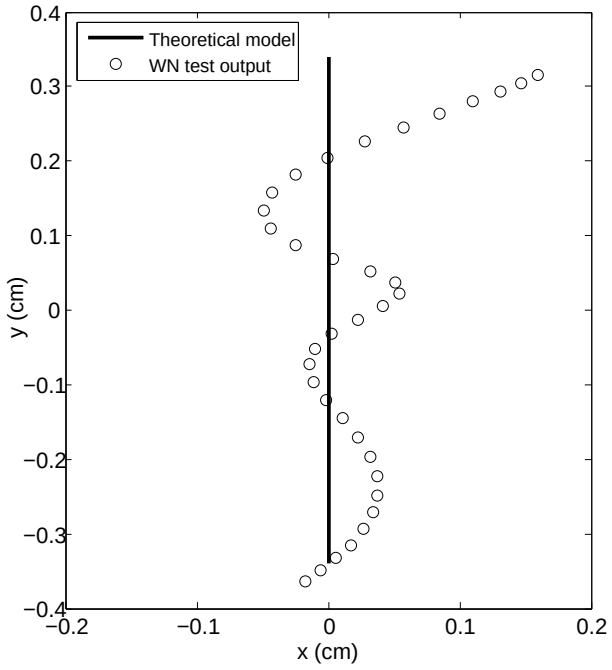

Figure 5.9: Comparing the WN test output and the theoretical model with gaps for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

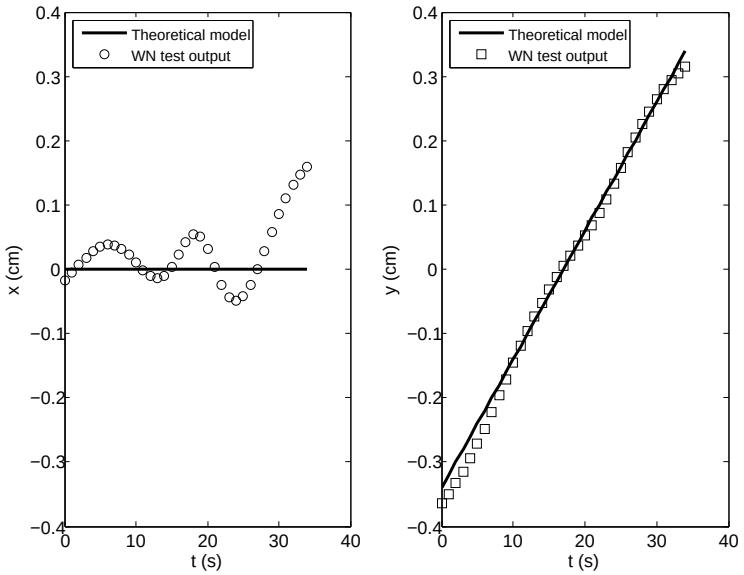

Figure 5.10: Comparing the WN test output and the theoretical model with gaps for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

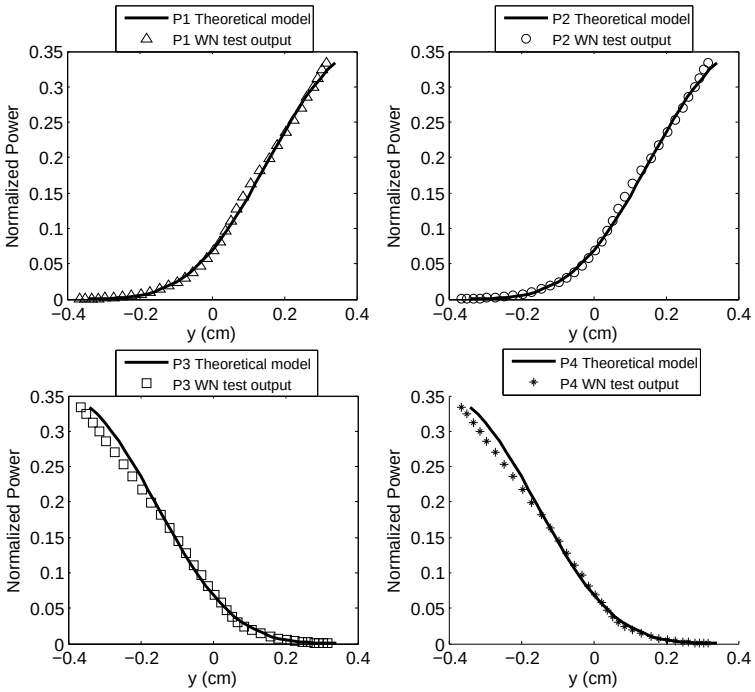

Figure 5.11: Comparing the WN test output and the theoretical model with gaps for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

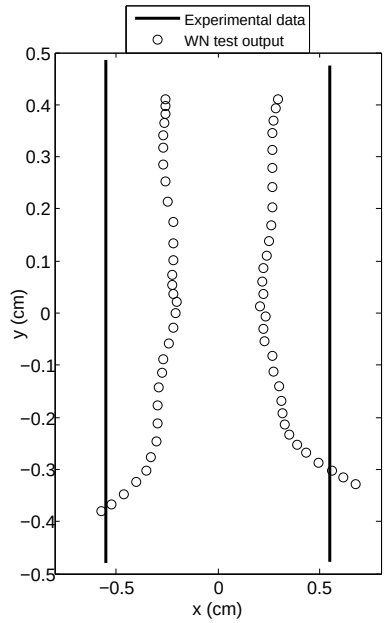

Figure 5.12: Comparing the WN test output and the experimental data for vertical scanning at $x = -0.55$ cm and $x = 0.55$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

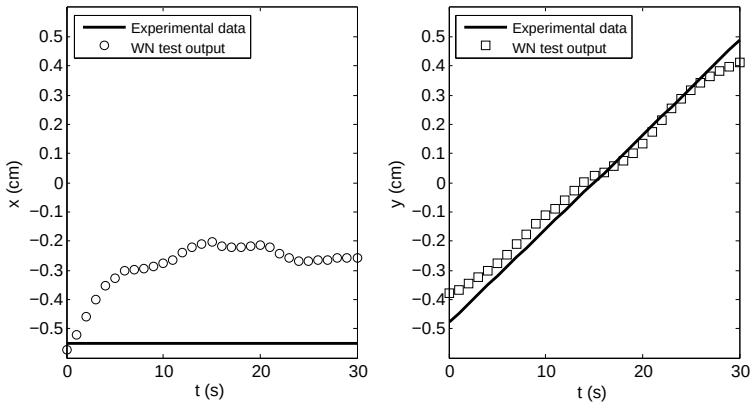

Figure 5.13: Comparing the WN test output and the experimental data for vertical scanning at $x = -0.55$ cm as a function of time. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

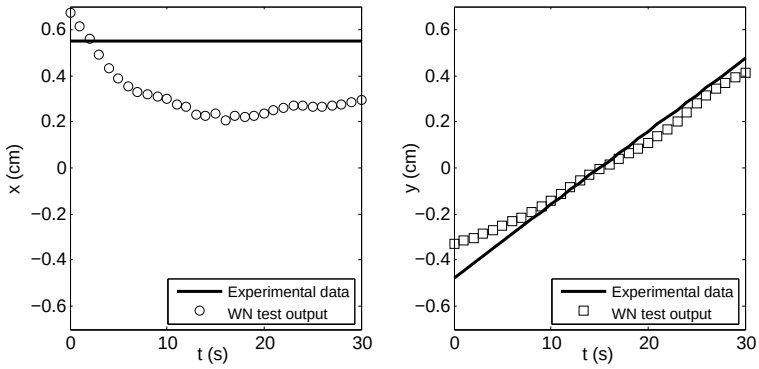

Figure 5.14: Comparing the WN test output and the experimental data for vertical scanning at $x = 0.55$ cm as a function of time. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

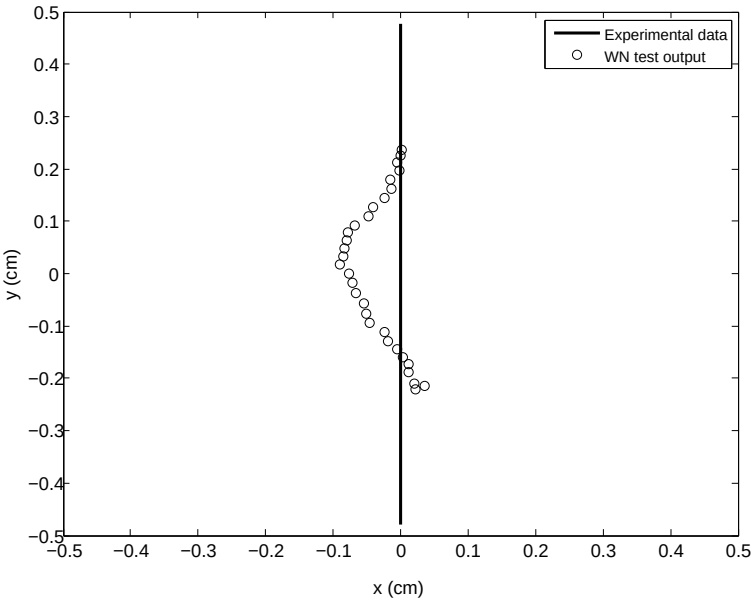

Figure 5.15: Comparing the WN test output and the experimental data for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

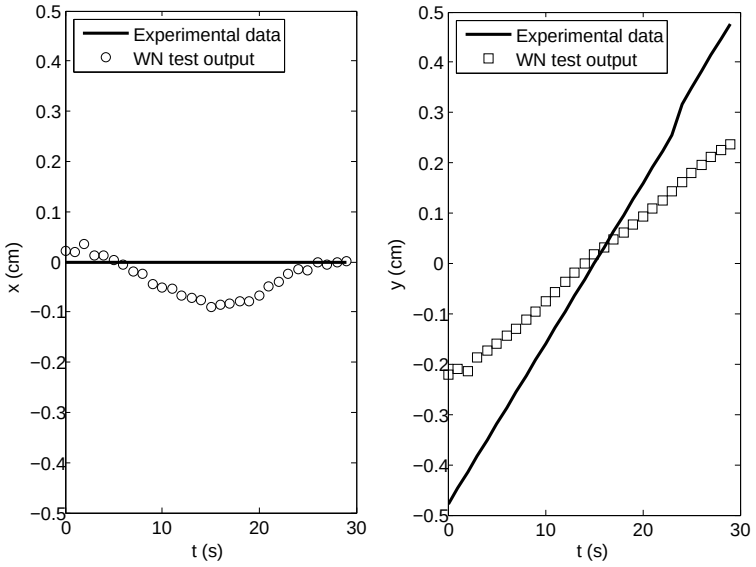

Figure 5.16: Comparing the WN test output and the experimental data for vertical scanning at $x = 0$ cm as a function of time. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

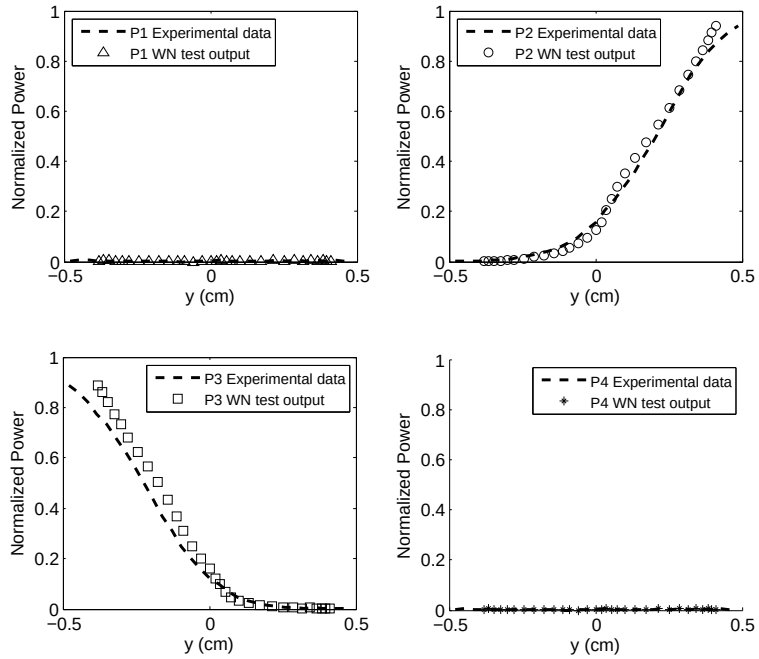

Figure 5.17: Comparing the WN test output and the experimental data for vertical scanning at $x = -0.55$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

Next, we studied the performance of the WN when the experimental data sets at $x = -0.55$ cm, $x = 0$ cm and $x = 0.55$ cm were used for testing. The experimental y data used for testing was within the range of -0.5 cm and 0.5 cm. The test errors obtained were 7.91e$-$02, 7.14e$-$02, and 2.66e$-$02 for the vertical scans at $x = -0.55$ cm, $x = 0.55$ cm and $x = 0$ cm.

The performance of the WN was further investigated by using the data set at $x = 0$ cm for testing and the theoretical model with vertical gap $\delta = 0.3$ cm and horizontal gap $\epsilon = 0.1$ cm for training. The resolution for the x position was set to 0.05 cm and that for the y position was set to 0.02 cm. Figure 5.20 shows the training error and the test error, after training the network for 100,000 iterations, for different values of N_w, while keeping μ and γ fixed at 0.01 and 0.99 respectively. As can be seen in Figure 5.20, we have a minimum test MSE of 7.90e$-$05, at $N_w = 15$ and the corresponding training MSE has a value of 2.48e$-$04. In this case, at $N_w = 15$, the test error is lower than the training error.

We further investigated the performance of the WN when the experimental data at $x = 0$ cm was used for testing. The theoretical model with vertical gap $\delta = 0.3$ cm and horizontal gap $\epsilon = 0.1$ cm, has been used for training. The resolution for the x position was set to 0.05 cm and that for the y position was set to 0.02 cm. The network was trained for 100,000 iterations, for different values of N_w, while keeping μ and γ fixed at 0.01 and 0.99 respectively. We obtained a minimum test error of about 1.22e$-$02 at $N_w = 63$ for the vertical scan at $x = 0$ cm as shown in Figure 5.23.

Using the proposed system and the uniqueness condition given in equation (5.6), the maximum position detection area A_{d} for an ideal n photocell array with no spatial gaps is represented in Figure 5.26 and can be computed as:

$$A_{\mathrm{d}} = \left(\sqrt{n} - 1\right)^2 A_{\mathrm{ph}}, \tag{5.7}$$

where A_{ph} is the active area for one photocell and n is the number of photocells with $\sqrt{n}$ an integer.

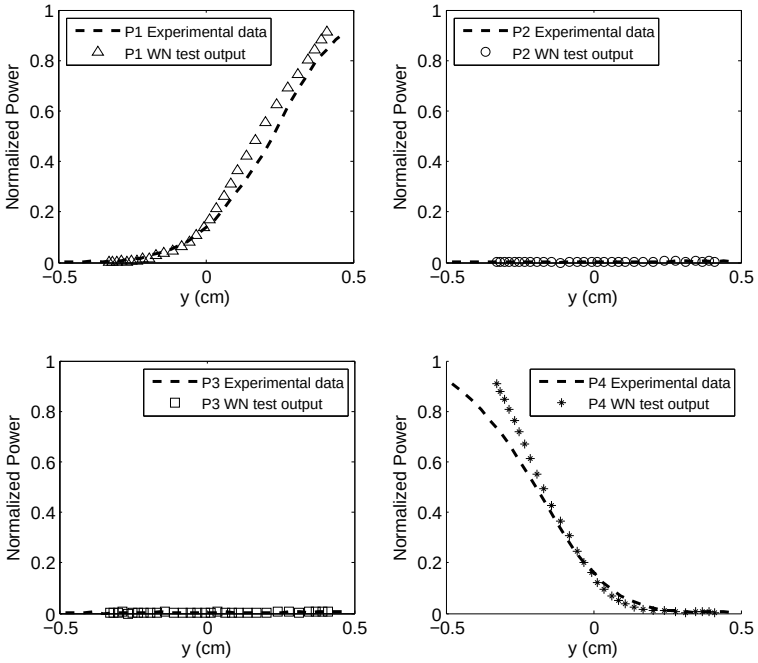

Figure 5.18: Comparing the WN test output and the experimental data for vertical scanning at $x = 0.55$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

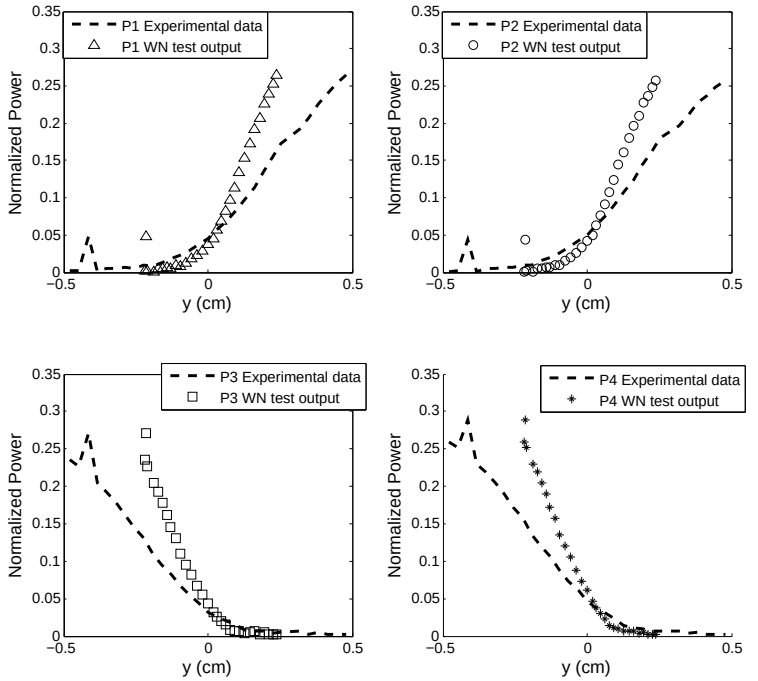

Figure 5.19: Comparing the WN test output and the experimental data for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.1 cm and the resolution for the y data used in the training is 0.02 cm.

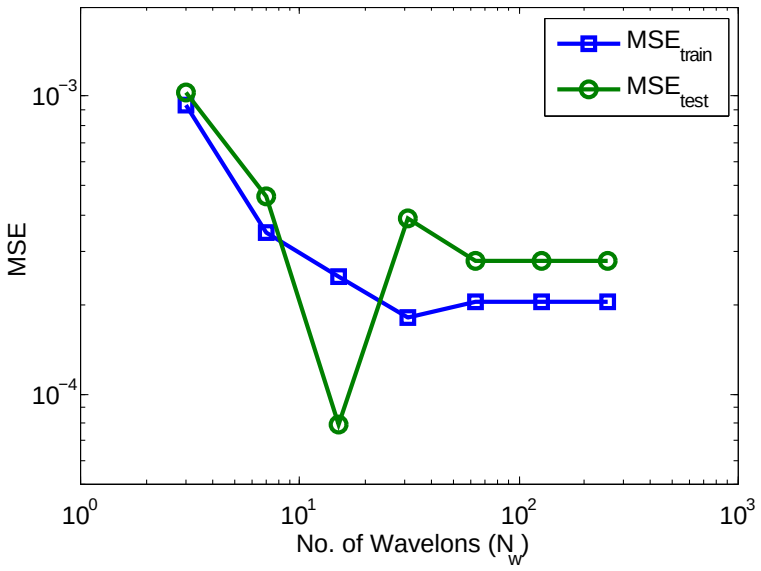

Figure 5.20: Comparing the MSE values vs. N_w after training the theoretical model with gaps and after testing for the theoretical data set at $x = 0$ cm. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

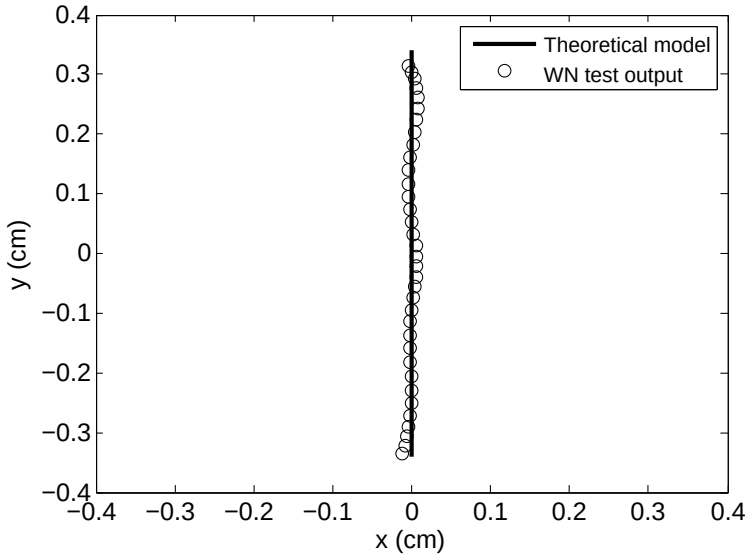

Figure 5.21: Comparing the WN test output and the theoretical model with gaps for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

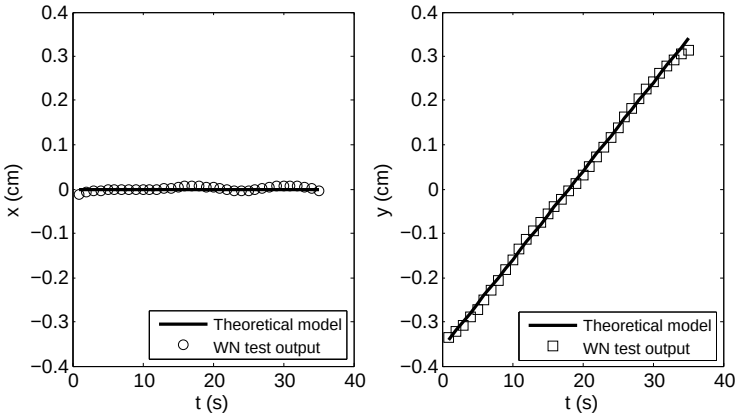

Figure 5.22: Comparing the WN test output and the theoretical model with gaps for vertical scanning at $x = 0$ cm as a function of time. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

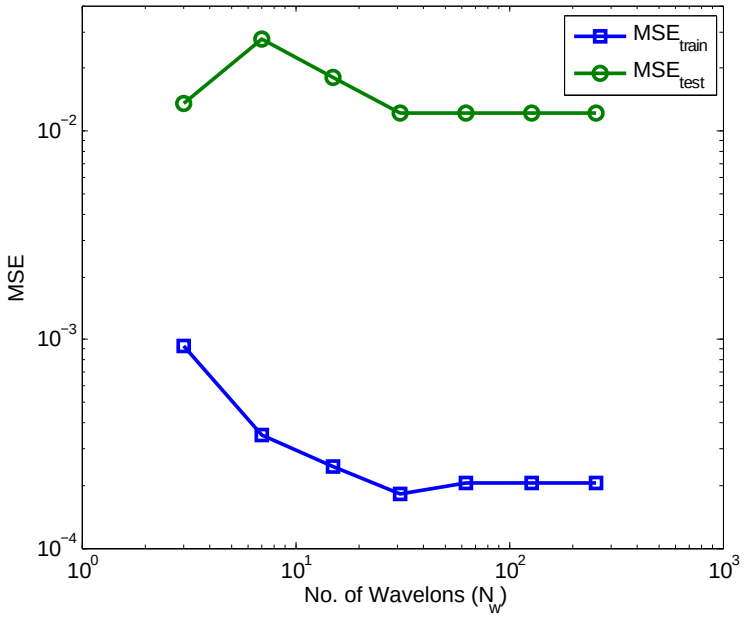

Figure 5.23: Comparing the MSE values vs. N_w after training the theoretical model with gaps and after testing for the experimental data set at $x = 0$ cm. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

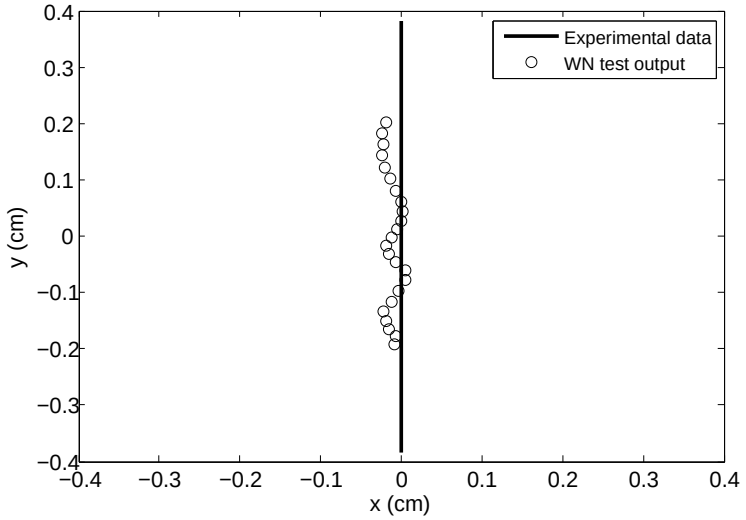

Figure 5.24: Comparing the WN test output and the experimental data for vertical scanning at $x = 0$ cm. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

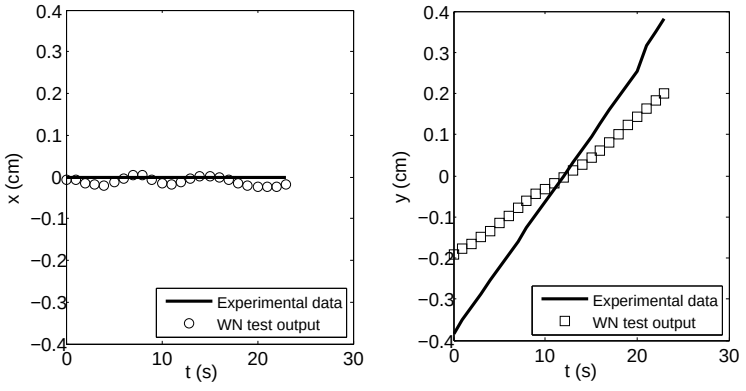

Figure 5.25: Comparing the WN test output and the experimental data for vertical scanning at $x = 0$ cm as a function of time. The resolution for the x data used in the training is 0.05 cm and the resolution for the y data used in the training is 0.02 cm.

5.3 Vibration Monitoring

In order to achieve a better fitting between the WN test output and the experimental data as shown in Figure 5.24, further adjustments should be made to improve the accuracy of the experimental setup. This involves reducing the vertical and horizontal gaps between the photocells, as well as eliminating any sources of power loss along the $x = 0$ cm axis of the photodetector array in the optical acquisition system. In addition, more vertical scans should be made within the detection area, with a lower resolution for the x position (≤ 0.05 cm).

One of the additional features of the proposed system is that it allows us to measure the rate of change of $\hat{Y}$ using equation (5.8), where dP/dt is the rate of change of power distribution and $\partial\hat{Y}/\partial P$ is the Jacobian matrix of $\hat{Y}$ with respect to P as the beam moves along the plane of photocells.

$$\frac{d\hat{Y}}{dt} = \frac{\partial\hat{Y}}{\partial P} \cdot \frac{dP}{dt}. \tag{5.8}$$

The Jacobian matrix $\partial\hat{Y}/\partial P$ can be computed using the trained WN by taking the partial derivative with respect to P of equation (5.1), as stated below:

$$\frac{\partial\hat{Y}^p}{\partial P_i} = \mathrm{W}_{\mathrm{oh}} * \frac{\partial\mathbf{\Phi}^p}{\partial P_i} + \begin{bmatrix} a_{1i} \\ a_{2i} \end{bmatrix}. \tag{5.9}$$

A potential application for the position detection system, is in vibration monitoring. Referring to Figure 4.1, if the mirror is placed on a vibrating platform, this will induce beam vibration where the position of the beam center will be changing as a function of time. Future work can be done to utilize the position information to find certain vibration characteristics such as speed, acceleration, frequency spectrum, and amplitude.

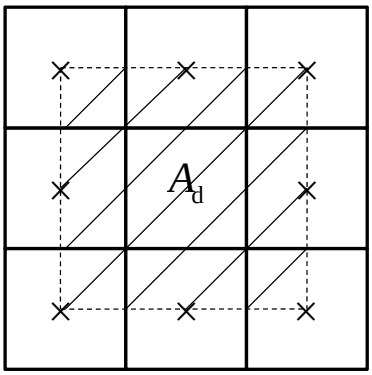

Figure 5.26: Shaded region indicates the area of detection and × represents the center of one photocell.

6

Conclusions

In this research, an optical position detection scheme using Gaussian beam analysis, and wavelet networks has been introduced. The closed form equations for the optical power covered by a certain area of overlap between the laser beam spot and one photodetector were derived, as the beam moves throughout the entire x–y plane. Accordingly, the power distribution acquired by a quadcell photodetector array was evaluated, taking into consideration the vertical and horizontal spatial gaps, δ and ϵ. A laboratory setup of the optical acquisition model was implemented to validate the results from the theoretical model and to assess the performance of the WN with experimental data. The input to the WN is the photodetector array power distribution and the output is an estimate of the x and y position of the laser beam center. The aspects of practical implementation and experimental limitations on the power distribution accuracy were discussed and the discrepancies with the theoretical results were presented.

Bibliography

[1] A. J. Makynen, J. T. Kostamovaara, and R. A. Myllyla, "A high-resolution lateral displacement sensing method using active illumination of a cooperative target and a focused four-quadrant position-sensitive detector," IEEE Transactions on Instrumentation and Measurement, vol. 44, no. 1, pp. 46-47, Feb. 1995.

[2] C. W. de Silva, *Vibration monitoring, testing, and instrumentation*. New York: CRC Press, 2007.

[3] K. Z. Tang, K. K. Tan, C. W. de Silva, T. H. Lee, K. C. Tan, and S. Y. Soh, "Application of vibration sensing in monitoring and control of machined health," IEEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings, pp. 377-378, July 2001.

[4] J. P. Sebastia, J. A. Lluch, J. R. L. Vizcaino, and J. S. Bellon, "Vibration detector based on GMR sensors," IEEE Transactions on Instrumentation and Measurement, vol. 58, no. 3, p. 707, March 2009.

[5] Y. Shan, J. E. Speich, and K. K. Leang, "Low-cost IR reflective sensors for submicrolevel position measurement and control," IEEE/ASME Trans. Mechatronics, vol. 13, no. 6, pp. 700-701, Dec. 2008.

[6] Z. Zhang and X. Bao, "Continuous and damped vibration detection based on fiber diversity detection sensor by rayleigh backscattering," Journal of Lightwave Technology, vol. 26, no. 7, p. 832, April 2008.

[7] J. A. Garcia-Souto and H. L. Rivera, "Multichannel fiber-optic interferometric sensor for measurments of temperature and vibrations in composite materials," IEEE Journal Selected Topics in Quantum Electronics, vol. 6, no. 5, p. 780, Sep./Oct. 2000.

[8] T. K. Gangopadhyay, S. Chakravorti, S. Chatterjee, and K. Bhattacharya, "Time-frequency analysis of multiple fringe and nonsinusoidal signals obtained from a fiber-optic vibration sensor using an extrinsic fabry-perot interferometer," Journal of Lightwave Technology, vol. 24, no. 5, pp. 2122-2123, May 2006.

[9] C. Wang, S. B. Trivedi, F. Jin, S. Stepanov, Z. Chen, J. Khurgin, P. Rodriguez, and N. S. Prasad, "Human life signs detection using high-sensitivity pulsed laser vibrometer," IEEE Sensors Journal, vol. 7, no. 9, p. 1370, Sep. 2007.

[10] A. J. Makynen, J. T. Kostamovaara, and R. A. Myllyla, "Displacement sensing resolution of position-sensitive detectors in atmospheric turbulence using retroreflected beam," IEEE Transactions on Instrumentation and Measurement, vol. 46, no. 5, pp. 1133-1134, Oct. 1997.

[11] J. Jason, H. Nilsson, B. Arvidsson, and A. Larsson, "Experimental Study of an Intensity Modulated Fiber-Optic Position Sensor with a Novel Readout System," IEEE Sensors Journal, vol. 8, no. 7, July 2008.

[12] L. P. Salles and D. W. de Lima Monteiro, "Designing the Response of an Optical Quad-Cell as Position-Sensitive Detector," IEEE Sensors Journal, vol. 10, no. 2, pp. 286-293, Feb. 2010.

[13] B. E. A. Saleh, *Fundamentals of photonics*. New York: Wiley, 1991.

[14] G. Laufer, *Introduction to optics and lasers in engineering*. Cambridge; New York: Cambridge University Press, 1996.

[15] D. A. McQuarrie, *Mathematical methods for scientists and engineers*. Sausalito; California: University Science Books, 2003.

[16] T. Poggio and F. Girosi, "Networks for Approximation and Learning," in Proc. IEEE, vol. 78, no. 9, Sept. 1990.

[17] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia: CBMS-NSF regional conference series in applied mathematics; 61, 1992.

[18] S. Mallat, *A Wavelet Tour of Signal Processing*. San Diego: Academic Press, 1998.

[19] E. S. Garcia-Trevino, V. Alarcon-Aquino, and J. F. Ramirez-Cruz, "Improving Wavelet-Networks Performance with a New Correlation-based Initialisation Method and Training Algorithm," in Proc. 15th International Conf. on Computing, IEEE 2006.

[20] Q. Zhang, "Using Wavelet Network in Nonparametric Estimation," IEEE Transactions on Neural Networks, vol. 8, no. 2, pp.227-229, July 1996.

[21] I. Daubechies, "The Wavelet Transform, Time-frequency Localization and Signal Analysis," IEEE Trans. Information Theory, vol. 36, no. 5, Sept. 1990.

[22] S. G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," IEEE Transactions of Pattern Analysis and Machine Intelligence, vol. 2, no. 7, July 1989.

[23] Q. Zhang and A. Benveniste, "Wavelet Networks," IEEE Transactions on Neural Networks, vol. 3, no. 6, Nov. 1992.

[24] T. Kugarajah and Q. Zhang, "Multidimensional Wavelet Frames," IEEE Transactions on Neural Networks, vol. 6, no. 6, Nov. 1995.

[25] Y. C. Pati and P. S. Krishnaprasad, "Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformations," Technical Research Report of the University of Maryland, TR 90-44.

[26] A. Boggess and F. J. Narcowich, *A First Course in Wavelets with Fourier Analysis*. New Jersey : Prentice-Hall, 2001.

[27] Y. Oussar and G. Dreyfus, "Initialization by Selection for Wavelet Network Training," Neurocomputing, vol. 34, pp. 134-143, 2000.

[28] Q. Zhang, "Wavelet Network: The Radial Structure and an Efficient Initialization Procedure," in European Control Conference (ECC), Groningen, Pays-Bas, 1993.

[29] A. E. Siegman, *Lasers*. University Science Books, 1986.

[30] Y. El-Ashi, R. Dhaouadi, and T. Landolsi, "Design of a novel optical vibrometer using Gaussian beam analysis," in Proc. 5th International Symposium on Mechatronics and its Applications (ISMA08), Amman, Jordan, May, 27-29 2008.

[31] Y. El-Ashi, R. Dhaouadi, and T. Landolsi, "Accuracy of a Gaussian Beam Optical Vibrometer with a Quad Photodetector Spatial Separation," Proc. of 3rd International Conf. on Modeling, Simulation and Applied Optimization, Sharjah, UAE, January, 20-22 2009.

[32] D. Haddad, P. Juncar, G. Geneves, and M. Wakim, "Gaussian Beams, and Spatial Modulation in Nanopositioning," IEEE Transactions on Instrumentation and Measurement, Oct. 2008.

[33] http://www.thorlabs.com, retrieved September 2008.

Appenndix A: Matlab Codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Program to calculate the area of overlap between the      %%%
%%% circular beam spot and the square photocell as the beam   %%%
%%% scans the plane of the photocell.                         %%%
%%% filename = 'Research22.m'                                 %%%
%%%                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc

xo=0.5;                 %x-dimension of the square photocell
yo=0.5;                 %y-dimension of the square photocell
r=0.5;                  %Radius of the circle
xn=-2:0.01:2;           %x-coordinate of the beam spot center
yn=-2:0.01:2;           %y-coordinate of the beam spot center
y1=yo; y2=-yo;
x1=xo; x2=-xo;

%*****************************************************%
%% Evaluating the points of intersection between the %%
%% circlular beam spot and the square photocell.     %%
%*****************************************************%
for s=1:length(xn)
 for t=1:length(yn)
  %Evaluating yup and ydn at x=x1
  [yup(1),ydn(1)]=feval('circy',x1,r,xn(s),yn(t));
  %Evaluating yup and ydn at x=x2
  [yup(2),ydn(2)]=feval('circy',x2,r,xn(s),yn(t));

  %Evaluating xup and xdn at y=y1
  [xup(1),xdn(1)]=feval('circx',y1,r,xn(s),yn(t));
  %Evaluating xup and xdn at y=y2
  [xup(2),xdn(2)]=feval('circx',y2,r,xn(s),yn(t));

   xaty1=[xup(1) xdn(1)];
   xaty2=[xup(2) xdn(2)];
```

```
yatx1=[yup(1) ydn(1)];
yatx2=[yup(2) ydn(2)];

for k=1:2
%If the elements are real, isreal=1, otherwise isreal=0:
  nxy1(k)=isreal(xaty1(k));
  nxy2(k)=isreal(xaty2(k));
  nyx1(k)=isreal(yatx1(k));
  nyx2(k)=isreal(yatx2(k));
end

%Replace the imaginary values by twice the dimensions
%of the square:
xaty1(find(nxy1==0))=2*xo;
xaty2(find(nxy2==0))=2*xo;
yatx1(find(nyx1==0))=2*yo;
yatx2(find(nyx2==0))=2*yo;

%Taking the xaty1 with in the range -xo≤x≤xo
jx1=find(xaty1≤xo & xaty1≥-xo);
%Taking the xaty2 with in the range -xo≤x≤xo
jx2=find(xaty2≤xo & xaty2≥-xo);
%Taking the yatx1 with in the range -yo≤y≤yo
jy1=find(yatx1≤yo & yatx1≥-yo);
%Taking the yatx2 with in the range -yo≤y≤yo
jy2=find(yatx2≤yo & yatx2≥-yo);

%Setting initial values to xa,xb,xc,xd
xa=i;xb=i;xc=x1;xd=x2;
%Setting initial values to ya,yb,yc,yd
ya=y1;yb=y2;yc=i;yd=i;

%***********************************%
%% Evaluating xa and xb, ya and yb. %%
%***********************************%

%Case: Circle intersects y1 at two different points
if length(jx1)==2 & xaty1(1)≠xaty1(2)
   xa=xaty1(1);
   xb=xaty1(2);
   ya=y1;
   yb=y1;
%Case: Upper and lower part of circle intersect y1
%at the same point
elseif length(jx1)==2 & xaty1(1)==xaty1(2)
   jx1=jx1(1);
   xa=xaty1(jx1);
%Case: Circle intersects y1 at one and only one point
elseif length(jx1)==1
   xa=xaty1(jx1);
%Case: No intersetion between circle and y1
elseif length(jx1)==0 & xa==i
   xa=i;
end

%Case: Circle intersects y2 at two different points
if length(jx2)==2 & xaty2(1)≠xaty2(2)
   xa=xaty2(1);
```

```
    xb=xaty2(2);
    ya=y2;
    yb=y2;
%Case: Upper and lower part of circle intersect y2
%at the same point
elseif length(jx2)==2 & xaty2(1)==xaty2(2)
    jx2=jx2(1);
    xb=xaty2(jx2);
%Case: Circle intersects y2 at one and only one point
elseif length(jx2)==1
    xb=xaty2(jx2);
%Case: No intersetion between circle and y2
elseif length(jx2)==0 & xb==i
    xb=i;
end



%************************************%
%% Evaluating xc and xd, yc and yd. %%
%************************************%

%Case: Circle intersects x1 at two different points
if length(jy1)==2 & yatx1(1)≠yatx1(2)
    yc=yatx1(1);
    yd=yatx1(2);
    xc=x1;
    xd=x1;
%Case: Upper and lower part of circle intersect x1
%at the same point
elseif length(jy1)==2 & yatx1(1)==yatx1(2)
    jy1=jy1(1);
    yc=yatx1(jy1);
%Case: Circle intersects x1 at one and only one point
elseif length(jy1)==1
    yc=yatx1(jy1);
%Case: No intersetion between circle and x1
elseif length(jy1)==0 & yc==i
    yc=i;
end

%Case: Circle intersects x2 at two different points
if length(jy2)==2 & yatx2(1)≠yatx2(2)
    yc=yatx2(1);
    yd=yatx2(2);
    xc=x2;
    xd=x2;
%Case: Upper and lower part of circle intersect x2
%at the same point
elseif length(jy2)==2 & yatx2(1)==yatx2(2)
    jy2=jy2(1);
    yd=yatx2(jy2);
%Case: Circle intersects x2 at one and only one point
elseif length(jy2)==1
    yd=yatx2(jy2);
%Case: No intersetion between circle and x1
elseif length(jy2)==0 & yd==i
    yd=i;
```

```
end

%*****************************************%
%% x-y coordinates of intersection points: %%
%*****************************************%

xint=[xa xb xc xd];
yint=[ya yb yc yd];

A=[xint;yint];
p=isreal(A);

if p==0
    [m,n]=find(A==i);
    if length(n)==4        %No intersection
       B=[i,i;i,i];
       xint=B(1,:);
       yint=B(2,:);
    elseif length(n)==1    %3-points of intersection
      if n(1)==1
         B=[A(:,2),A(:,3),A(:,4)];
      elseif n(1)==2
         B=[A(:,1),A(:,3),A(:,4)];
      elseif n(1)==3
         B=[A(:,1),A(:,2),A(:,4)];
      elseif n(1)==4
           B=[A(:,1),A(:,2),A(:,3)];
      end
      xint=B(1,:);
      yint=B(2,:);

      if (abs(B(:,1)-B(:,2))≤1.0e-015) & (abs(B(:,1)-B(:,2))≥0)
          C=[B(:,1),B(:,3)];
          xint=C(1,:);
          yint=C(2,:);
      elseif (abs(B(:,2)-B(:,3))≤1.0e-015) & (abs(B(:,2)-B(:,3))≥0)
          C=[B(:,1),B(:,2)];
          xint=C(1,:);
          yint=C(2,:);
      elseif (abs(B(:,1)-B(:,3))≤1.0e-015) & (abs(B(:,1)-B(:,3))≥0)
          C=[B(:,1),B(:,2)];
          xint=C(1,:);
          yint=C(2,:);
      end

    elseif length(n)==2    %2-points of intersection
      if n(1)==1 & n(2)==2
         B=[A(:,3),A(:,4)];
      elseif n(1)==1 & n(2)==3
         B=[A(:,2),A(:,4)];
      elseif n(1)==1 & n(2)==4
         B=[A(:,2),A(:,3)];
      elseif n(1)==2 & n(2)==3
         B=[A(:,1),A(:,4)];
      elseif n(1)==2 & n(2)==4
         B=[A(:,1),A(:,3)];
      elseif n(1)==3 & n(2)==4
         B=[A(:,1),A(:,2)];
```

```
            end

            if B(:,1)==B(:,2)
               C=B(:,1);
               xint=C(1,:);
               yint=C(2,:);
            else
               xint=B(1,:);
               yint=B(2,:);
            end

          elseif length(n)==3    %1-point of intersection
            f=10-sum(n);
            B=[A(:,f)];
            xint=B(1,:);
            yint=B(2,:);
          end
          %2-points of intersection, special case
          %(half area of circle inside square)
          elseif p~=0 & sum(sum(A))==2
            B=[A(:,1),A(:,2)];
            xint=B(1,:);
            yint=B(2,:);
          elseif p~=0            %4-points of intersection
            xint=A(1,:);
            yint=A(2,:);
          end

          %Distance between corner1 (xo,yo) and centre of circle
          D(1)=sqrt((x1-xn(s))^2+(y1-yn(t))^2);
          %Distance between corner2 (-xo,yo) and centre of circle
          D(2)=sqrt((x2-xn(s))^2+(y1-yn(t))^2);
          %Distance between corner3 (-xo,-yo) and centre of circle
          D(3)=sqrt((x2-xn(s))^2+(y2-yn(t))^2);
          %Distance between corner4 (xo,-yo) and centre of circle
          D(4)=sqrt((x1-xn(s))^2+(y2-yn(t))^2);

          c=find(D<r);           %Corners of square inside the circle

          Xn=xn(s);
          Yn=yn(t);
          Area=feval('AREAXY2',r,Xn,Yn,xint,yint,xo,yo,c);
          area(s,t)=Area;
end
end

%Plot of area vs. position of beam spot center along xy-directions
figure
mesh(xn,yn,area)
xlabel('xn (cm)')
ylabel('yn (cm)')
zlabel('Area of overlap (cm^2)')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Program to calculate the Power at the area of overlap      %%%
%%% overlap between the circular beam spot and the square      %%%
%%% photocell as the beam scans the plane of the photocell.    %%%
%%% filename = 'Research2POWER.m'                              %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc

xo=0.5;                  %x-dimension of the square photocell
yo=0.5;                  %y-dimension of the square photocell
r=0.5;                   %Radius of the circle
xn=-2:0.01:2;            %x-coordinate of the beam spot center
yn=-2:0.01:2;            %y-coordinate of the beam spot center
y1=yo; y2=-yo;
x1=xo; x2=-xo;

%*****************************************************%
%% Evaluating the points of intersection between the %%
%% circlular beam spot and the square photocell.     %%
%*****************************************************%
for s=1:length(xn)
 for t=1:length(yn)
  %Evaluating yup and ydn at x=x1
  [yup(1),ydn(1)]=feval('circy',x1,r,xn(s),yn(t));
  %Evaluating yup and ydn at x=x2
  [yup(2),ydn(2)]=feval('circy',x2,r,xn(s),yn(t));

  %Evaluating xup and xdn at y=y1
  [xup(1),xdn(1)]=feval('circx',y1,r,xn(s),yn(t));
  %Evaluating xup and xdn at y=y2
  [xup(2),xdn(2)]=feval('circx',y2,r,xn(s),yn(t));

   xaty1=[xup(1) xdn(1)];
   xaty2=[xup(2) xdn(2)];

   yatx1=[yup(1) ydn(1)];
   yatx2=[yup(2) ydn(2)];

   for k=1:2
   %If the elements are real, isreal=1, otherwise isreal=0:
     nxy1(k)=isreal(xaty1(k));
     nxy2(k)=isreal(xaty2(k));
     nyx1(k)=isreal(yatx1(k));
     nyx2(k)=isreal(yatx2(k));
   end

   %Replace the imaginary values by twice the dimensions
   %of the square:
   xaty1(find(nxy1==0))=2*xo;
   xaty2(find(nxy2==0))=2*xo;
   yatx1(find(nyx1==0))=2*yo;
   yatx2(find(nyx2==0))=2*yo;
```

```
%Taking the xaty1 with in the range −xo≤x≤xo
jx1=find(xaty1≤xo & xaty1≥−xo);
%Taking the xaty2 with in the range −xo≤x≤xo
jx2=find(xaty2≤xo & xaty2≥−xo);
%Taking the yatx1 with in the range −yo≤y≤yo
jy1=find(yatx1≤yo & yatx1≥−yo);
%Taking the yatx2 with in the range −yo≤y≤yo
jy2=find(yatx2≤yo & yatx2≥−yo);

%Setting initial values to xa,xb,xc,xd
xa=i;xb=i;xc=x1;xd=x2;
%Setting initial values to ya,yb,yc,yd
ya=y1;yb=y2;yc=i;yd=i;

%**********************************%
%% Evaluating xa and xb, ya and yb. %%
%**********************************%

%Case: Circle intersects y1 at two different points
if length(jx1)==2 & xaty1(1)≠xaty1(2)
   xa=xaty1(1);
   xb=xaty1(2);
   ya=y1;
   yb=y1;
%Case: Upper and lower part of circle intersect y1
%at the same point
elseif length(jx1)==2 & xaty1(1)==xaty1(2)
   jx1=jx1(1);
   xa=xaty1(jx1);
%Case: Circle intersects y1 at one and only one point
elseif length(jx1)==1
   xa=xaty1(jx1);
%Case: No intersetion between circle and y1
elseif length(jx1)==0 & xa==i
   xa=i;
end

%Case: Circle intersects y2 at two different points
if length(jx2)==2 & xaty2(1)≠xaty2(2)
   xa=xaty2(1);
   xb=xaty2(2);
   ya=y2;
   yb=y2;
%Case: Upper and lower part of circle intersect y2
%at the same point
elseif length(jx2)==2 & xaty2(1)==xaty2(2)
   jx2=jx2(1);
   xb=xaty2(jx2);
%Case: Circle intersects y2 at one and only one point
elseif length(jx2)==1
   xb=xaty2(jx2);
%Case: No intersetion between circle and y2
elseif length(jx2)==0 & xb==i
   xb=i;
end
```

```
%************************************%
%% Evaluating xc and xd, yc and yd. %%
%************************************%

%Case: Circle intersects x1 at two different points
if length(jy1)==2 & yatx1(1)≠yatx1(2)
   yc=yatx1(1);
   yd=yatx1(2);
   xc=x1;
   xd=x1;
%Case: Upper and lower part of circle intersect x1
%at the same point
elseif length(jy1)==2 & yatx1(1)==yatx1(2)
   jy1=jy1(1);
   yc=yatx1(jy1);
%Case: Circle intersects x1 at one and only one point
elseif length(jy1)==1
   yc=yatx1(jy1);
%Case: No intersetion between circle and x1
elseif length(jy1)==0 & yc==i
   yc=i;
end

%Case: Circle intersects x2 at two different points
if length(jy2)==2 & yatx2(1)≠yatx2(2)
   yc=yatx2(1);
   yd=yatx2(2);
   xc=x2;
   xd=x2;
%Case: Upper and lower part of circle intersect x2
%at the same point
elseif length(jy2)==2 & yatx2(1)==yatx2(2)
   jy2=jy2(1);
   yd=yatx2(jy2);
%Case: Circle intersects x2 at one and only one point
elseif length(jy2)==1
   yd=yatx2(jy2);
%Case: No intersetion between circle and x1
elseif length(jy2)==0 & yd==i
   yd=i;
end

%******************************************%
%% x-y coordinates of intersection points: %%
%******************************************%

xint=[xa xb xc xd];
yint=[ya yb yc yd];

A=[xint;yint];
p=isreal(A);

if p==0
    [m,n]=find(A==i);
    if length(n)==4        %No intersection
       B=[i,i;i,i];
       xint=B(1,:);
       yint=B(2,:);
```

```
elseif length(n)==1    %3-points of intersection
  if n(1)==1
     B=[A(:,2),A(:,3),A(:,4)];
  elseif n(1)==2
     B=[A(:,1),A(:,3),A(:,4)];
  elseif n(1)==3
     B=[A(:,1),A(:,2),A(:,4)];
  elseif n(1)==4
       B=[A(:,1),A(:,2),A(:,3)];
  end
  xint=B(1,:);
  yint=B(2,:);

  if (abs(B(:,1)-B(:,2))≤1.0e-015) & (abs(B(:,1)-B(:,2))≥0)
      C=[B(:,1),B(:,3)];
      xint=C(1,:);
      yint=C(2,:);
  elseif (abs(B(:,2)-B(:,3))≤1.0e-015) & (abs(B(:,2)-B(:,3))≥0)
      C=[B(:,1),B(:,2)];
      xint=C(1,:);
      yint=C(2,:);
  elseif (abs(B(:,1)-B(:,3))≤1.0e-015) & (abs(B(:,1)-B(:,3))≥0)
      C=[B(:,1),B(:,2)];
      xint=C(1,:);
      yint=C(2,:);
  end

elseif length(n)==2    %2-points of intersection
  if n(1)==1 & n(2)==2
     B=[A(:,3),A(:,4)];
  elseif n(1)==1 & n(2)==3
     B=[A(:,2),A(:,4)];
  elseif n(1)==1 & n(2)==4
     B=[A(:,2),A(:,3)];
  elseif n(1)==2 & n(2)==3
     B=[A(:,1),A(:,4)];
  elseif n(1)==2 & n(2)==4
     B=[A(:,1),A(:,3)];
  elseif n(1)==3 & n(2)==4
     B=[A(:,1),A(:,2)];
  end

  if B(:,1)==B(:,2)
     C=B(:,1);
     xint=C(1,:);
     yint=C(2,:);
  else
     xint=B(1,:);
     yint=B(2,:);
  end

elseif length(n)==3    %1-point of intersection
  f=10-sum(n);
  B=[A(:,f)];
  xint=B(1,:);
  yint=B(2,:);
end
%2-points of intersection, special case
```

```
        %(half area of circle inside square)
        elseif p≠0 & sum(sum(A))==2
          B=[A(:,1),A(:,2)];
          xint=B(1,:);
          yint=B(2,:);
        elseif p≠0            %4-points of intersection
          xint=A(1,:);
          yint=A(2,:);
        end

        %Distance between corner1 (xo,yo) and centre of circle
        D(1)=sqrt((x1-xn(s))^2+(y1-yn(t))^2);
        %Distance between corner2 (-xo,yo) and centre of circle
        D(2)=sqrt((x2-xn(s))^2+(y1-yn(t))^2);
        %Distance between corner3 (-xo,-yo) and centre of circle
        D(3)=sqrt((x2-xn(s))^2+(y2-yn(t))^2);
        %Distance between corner4 (xo,-yo) and centre of circle
        D(4)=sqrt((x1-xn(s))^2+(y2-yn(t))^2);

        %Corners of square inside the circle
        c=find(D<r);
        %Wavelength of a He-Ne laser source in cm
        lamda=6.33e-05;
        %Spot size of the beam 2Wo=2/3cm, to get 99% of Total power
        Wo=1/3;
        zo=(pi*Wo^2)/lamda;     %Rayleigh range in cm
        z=0;                    %Axial distance z in cm
        Io=1;                   %Maximum intensity value
        W=Wo*sqrt(1+(z/zo)^2);  %Beam Width
        I1=Io*(Wo/W)^2;
        AA=I1*(W/2)^2;
        BB=exp((-2*r^2)/W^2);

        Pc=I1*(2*pi)*((W/2)^2)*(1-BB);  %Power of spot with radius r
        PT=0.5*Io*pi*Wo^2;              %Total Power of the beam

        Power=feval('POWERXY2',r,xn(s),yn(t),xint,yint,xo,yo,c);
        power(s,t)=Power/Pc;
    end
    end

    %Plot of Normalized Power vs. position of beam
    %spot center along xy-directions
    figure
    mesh(xn,yn,power)
    xlabel('xn (cm)')
    ylabel('yn (cm)')
    zlabel('Normalized Power')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate x-coordinates of the intersection   %%%
%%% points between the circular beam spot and the upper        %%%
%%% or lower side of the square photocell.                     %%%
%%% filename = 'circx.m'                                       %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*       y = y value of upper or lower side of the           *%
%*           square photocell.                               *%
%* Output of the function:                                   *%
%*     xup = x-coordinate of the first point of intersection.*%
%*     xdn = x-coordinate of the second point of intersection.*%
%*************************************************************%

function [xup,xdn]=circx(y,r,xn,yn)
xup=sqrt(r^2-(y-yn)^2)+xn;
xdn=-sqrt(r^2-(y-yn)^2)+xn;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate y-coordinates of the intersection   %%%
%%% points between the circular beam spot and the right or     %%%
%%% left side of the square photocell.                         %%%
%%% filename = 'circy.m'                                       %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*       x = x value of upper or lower side of the           *%
%*           square photocell.                               *%
%* Output of the function:                                   *%
%*     yup = x-coordinate of the first point of intersection.*%
%*     ydn = x-coordinate of the second point of intersection.*%
%*************************************************************%

function [yup,ydn]=circy(x,r,xn,yn)
yup=sqrt(r^2-(x-xn)^2)+yn;
ydn=-sqrt(r^2-(x-xn)^2)+yn;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the area of overlap between the                  %%%
%%% circular beam spot and the square photocell.                            %%%
%%% Center of circular beam spot is moving along the x-axis                 %%%
%%% while the square photocell is fixed at the origin of the                %%%
%%% coordinate system.                                                      %%%
%%% Given, diameter of the circular beam spot is equivalent                 %%%
%%% to one side of the square photocell.                                    %%%
%%% filename = 'AREAX2.m'                                                   %%%
%%%                                                                         %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*     Area = Area of intersection between circle and square,*%
%*            as circle moves along x-axis.                  *%
%*************************************************************%

function Area=AREAX2(r,xn,yn,xint,yint,xo,yo,c)
px=isreal(xint);    %Check if there is intersection (x-coordinate)
py=isreal(yint);    %Check if there is intersection (y-coordinate)
Ac=pi*(r^2);        %Area of circle with radius r
%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(xint)==1)
    Area=0;
elseif px≠0 & py≠0
%Case 2: Centre of circle greater than or equal to xo
%(right-most side of square)
    if xn≥xo
        d=xn-xo;
        alpha=asin(d/r);
        Area=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
%Case 3: Centre of circle less than or equal to -xo
%(left-most side of square)
    elseif xn≤-xo
        d=-xo-xn;
        alpha=asin(d/r);
        Area=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
%Case 4: Centre of circle between 0 and xo (right half of square)
    elseif xn>0 & xn<xo
        d=xo-xn;
```

```
            alpha=asin(d/r);
            A1=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
            Area=Ac-A1;
    %Case 5: Centre of circle between -xo and 0 (left half of square)
        elseif xn<0 & xn>-xo
            d=xo+xn;
            alpha=asin(d/r);
            A1=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
            Area=Ac-A1;
        end
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the area of overlap between the                    %%%
%%% circular beam spot and the square photocell.                              %%%
%%% Center of circular beam spot is moving along the y-axis                   %%%
%%% while the square photocell is fixed at the origin of the                  %%%
%%% coordinate system.                                                        %%%
%%% Given, diameter of the circular beam spot is equivalent                   %%%
%%% to one side of the square photocell.                                      %%%
%%% filename = 'AREAY2.m'                                                     %%%
%%%                                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*     Area = Area of intersection between circle and square,*%
%*            as circle moves along y-axis.                  *%
%*************************************************************%

function Area=AREAY2(r,xn,yn,xint,yint,xo,yo,c)
px=isreal(xint);    %Check if there is intersection (x-coordinate)
py=isreal(yint);    %Check if there is intersection (y-coordinate)
Ac=pi*(r^2);        %Area of circle with radius r

%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(yint)==1)
    Area=0;
elseif px≠0 & py≠0
%Case 2: Centre of circle greater than or equal to yo
%(upper side of square)
    if yn≥yo
        d=yn-yo;
        beta=asin(d/r);
        Area=((r^2)/2)*(pi-2*beta-sin(2*beta));
%Case 3: Centre of circle less than or equal to -yo
%(lower side of square)
    elseif yn≤-yo
        d=-yo-yn;
        beta=asin(d/r);
        Area=((r^2)/2)*(pi-2*beta-sin(2*beta));
%Case 4: Centre of circle between 0 and yo (upper half of square)
    elseif yn>0 & yn<yo
```

```
            d=yo-yn;
            beta=asin(d/r);
            A1=((r^2)/2)*(pi-2*beta-sin(2*beta));
            Area=Ac-A1;
    %Case 5: Centre of circle between -yo and 0 (lower half of square)
        elseif yn<0 & yn>-yo
            d=yo+yn;
            beta=asin(d/r);
            A1=((r^2)/2)*(pi-2*beta-sin(2*beta));
            Area=Ac-A1;
        end
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the area of overlap between the                  %%%
%%% circular beam spot and the square photocell.                            %%%
%%% Center of circular beam spot is moving along both the x                 %%%
%%% and y axis while the square photocell is fixed at the                   %%%
%%% origin of the coordinate system.                                        %%%
%%% Given, diameter of the circular beam spot is equivalent                 %%%
%%% to one side of the square photocell.                                    %%%
%%% filename = 'AREAXY2.m'                                                  %%%
%%%                                                                         %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*     Area = Area of intersection between circle and square,*%
%*            as circle moves along both the x and y axis.   *%
%*************************************************************%

function Area=AREAXY2(r,xn,yn,xint,yint,xo,yo,c)
px=isreal(xint);     %Check if there is intersection (x-coordinate)
py=isreal(yint);     %Check if there is intersection (y-coordinate)
Ac=pi*(r^2);         %Area of circle with radius r
%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(xint)==1)
    Area=0;
elseif px≠0 & py≠0
%Case 2: Center of circle at the origin of the coordinate system
 if xn==0 & yn==0
    Area=Ac;
%Case 3: Center of circle moving along the y-axis only
 elseif xn==0 & yn≠0
    Area=feval('AREAY2',r,xn,yn,xint,yint,xo,yo,c);
%Case 4: Center of circle moving along the x-axis only
 elseif xn≠0 & yn==0
    Area=feval('AREAX2',r,xn,yn,xint,yint,xo,yo,c);
%Case 5: Center of circle moving along both x and y axis
 elseif xn≠0 & yn≠0

   if xn>xo & yn>yo                          %Region 1
      dx=xn-xo;
      dy=yn-yo;
```

```
elseif xn>xo & yn<-yo                    %Region 4
   dx=xn-xo;
   dy=-yn-yo;
elseif xn<-xo & yn>yo                    %Region 2
   dx=-xn-xo;
   dy=yn-yo;
elseif xn<-xo & yn<-yo                   %Region 3
   dx=-xn-xo;
   dy=-yn-yo;

elseif xn≥xo & (yn≤yo & yn>0)            %Region 5
   dx=xn-xo;
   dy=yo-yn;
elseif xn≥xo & (yn≥-yo & yn<0)           %Region 6
   dx=xn-xo;
   dy=yo+yn;
elseif xn≤-xo & (yn≤yo & yn>0)           %Region 7
   dx=-xn-xo;
   dy=yo-yn;
elseif xn≤-xo & (yn≥-yo & yn<0)          %Region 8
   dx=-xn-xo;
   dy=yo+yn;

elseif yn≥yo & (xn≤xo & xn>0)            %Region 11
   dx=xo-xn;
   dy=yn-yo;
elseif yn≥yo & (xn≥-xo & xn<0)           %Region 12
   dx=xo+xn;
   dy=yn-yo;
elseif yn≤-yo & (xn≤xo & xn>0)           %Region 9
   dx=xo-xn;
   dy=-yn-yo;
elseif yn≤-yo & (xn≥-xo & xn<0)          %Region 10
   dx=xo+xn;
   dy=-yn-yo;

elseif (xn<xo & xn>0) & (yn<yo & yn>0)   %Region 13
   dx=xo-xn;
   dy=yo-yn;
elseif (xn>-xo & xn<0) & (yn<yo & yn>0)  %Region 14
   dx=xo+xn;
   dy=yo-yn;
elseif (xn<xo & xn>0) & (yn>-yo & yn<0)  %Region 16
   dx=xo-xn;
   dy=yo+yn;
elseif (xn>-xo & xn<0) & (yn>-yo & yn<0) %Region 15
   dx=xo+xn;
   dy=yo+yn;
end

alpha=asin(dx/r);
beta=asin(dy/r);
phi=atan(dy/dx);

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 1, 2, 3 or 4:
if(xn>xo & yn>yo)|(xn>xo & yn<-yo)|
  (xn<-xo & yn>yo)|(xn<-xo & yn<-yo)
```

```
  Area=((r^2)/2)*((pi/2)-(alpha+beta)-
       (((sin(alpha))/(cos(phi)))*(cos(phi+alpha)))-
       (((sin(beta))/(sin(phi)))*(sin(phi-beta))));

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 5, 6, 7 or 8:
elseif(xn≥xo & (yn≤yo & yn>0))|(xn≥xo & (yn≥-yo & yn<0))|
      (xn≤-xo & (yn≤yo & yn>0))|(xn≤-xo &  (yn≥-yo & yn<0))
      Ax=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
         if length(c)==1 & length(xint)==2
             if dx≠0 & dy≠0
                 Axy=((r^2)/2)*((pi/2)-(alpha+beta)-
                 (((sin(alpha))/(cos(phi)))*(cos(phi+alpha)))-
                 (((sin(beta))/(sin(phi)))*(sin(phi-beta))));
                 Area=Ax-Axy;
             elseif dx≠0 & dy==0
                 Axy=((r^2)/2)*((pi/2)-alpha-(1/2)*sin(2*alpha));
                 Area=Ax-Axy;
             elseif dx==0 & dy≠0
                 Axy=((r^2)/2)*((pi/2)-beta-(1/2)*sin(2*beta));
                 Area=Ax-Axy;
             elseif dx==0 & dy==0
                 Area=(pi/4)*(r^2);
             end
         elseif length(c)==0 & length(xint)==2
             Area=Ax;
         end

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 9, 10, 11 or 12:
elseif (yn≥yo & (xn≤xo & xn>0))|(yn≥yo & (xn≥-xo & xn<0))|
       (yn≤-yo & (xn≤xo & xn>0))|(yn≤-yo & (xn≥-xo & xn<0))
       Ay=((r^2)/2)*(pi-2*beta-sin(2*beta));
         if length(c)==1 & length(xint)==2
             if dx≠0 & dy≠0
                 Axy=((r^2)/2)*((pi/2)-(alpha+beta)-
                 (((sin(alpha))/(cos(phi)))*(cos(phi+alpha)))-
                 (((sin(beta))/(sin(phi)))*(sin(phi-beta))));
                 Area=Ay-Axy;
             elseif dx≠0 & dy==0
                 Axy=((r^2)/2)*((pi/2)-alpha-(1/2)*sin(2*alpha));
                 Area=Ay-Axy;
             elseif dx==0 & dy≠0
                 Axy=((r^2)/2)*((pi/2)-beta-(1/2)*sin(2*beta));
                 Area=Ay-Axy;
             elseif dx==0 & dy==0
                 Area=(pi/4)*(r^2);
             end
         elseif length(c)==0 & length(xint)==2
             Area=Ay;
         end

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 13, 14, 15 or 16:
elseif((xn<xo & xn>0) & (yn<yo & yn>0))|
      ((xn>-xo & xn<0) & (yn<yo & yn>0))|
      ((xn<xo & xn>0) & (yn>-yo & yn<0))|
      ((xn>-xo & xn<0) & (yn>-yo & yn<0))
```

```
                if length(c)==1 & length(xint)==2
                    Ay=((r^2)/2)*(pi-2*beta-sin(2*beta));
                    Ax=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
                    Axy=((r^2)/2)*((pi/2)-(alpha+beta)-
                    (((sin(alpha))/(cos(phi)))*(cos(phi+alpha)))-
                    (((sin(beta))/(sin(phi)))*(sin(phi-beta))));
                    Area=Ac-(Ay+Ax-Axy);
                elseif length(c)==0 & length(xint)==4
                    Ay=((r^2)/2)*(pi-2*beta-sin(2*beta));
                    Ax=((r^2)/2)*(pi-2*alpha-sin(2*alpha));
                    Area=Ac-(Ay+Ax);
                end
            end
        end
    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the Power at the area of overlap    %%%
%%% between circular beam spot and the square photocell.       %%%
%%% Center of circular beam spot is moving along the x-axis    %%%
%%% while the square photocell is fixed at the origin          %%%
%%% of the coordinate system.                                  %%%
%%% Given, diameter of the circular beam spot is equivalent    %%%
%%% to one side of the square photocell.                       %%%
%%% filename = 'POWERX2.m'                                     %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*   Power = Power at the area of intersection between       *%
%*           circlular beam spot and square photocell, as the*%
%*           beam spot moves along x-axis.                   *%
%*************************************************************%

function Power=POWERX2(r,xn,yn,xint,yint,xo,yo,c)

lamda=6.33e-05;          %Wavelength of a He-Ne laser source in cm
%Spot size of the beam 2Wo=2/3cm, to get 99% of Total Power
Wo=1/3;
zo=(pi*Wo^2)/lamda;      %Rayleigh range in cm
z=0;                     %Axial distance z in cm
Io=1;                    %Maximum intensity value
W=Wo*sqrt(1+(z/zo)^2);   %Beam Width
I1=Io*(Wo/W)^2;
AA=I1*(W/2)^2;
BB=exp((-2*r^2)/W^2);



px=isreal(xint);         %Check if there is intersection (x-coordinate)
py=isreal(yint);         %Check if there is intersection (y-coordinate)
Pc=I1*(2*pi)*((W/2)^2)*(1-BB);       %Power of spot with radius r

%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(xint)==1)
    Power=0;
```

```
elseif px≠0 & py≠0
%Case 2: Centre of circle greater than or equal to xo
%(right−most side of square)
    if xn≥xo
        d=xn−xo;
        kx=(2/W^2)*d^2;
        Q=@(alpha)exp(−kx./((sin(alpha)).^2));
        alpha1=asin(d/r);
        alpha2=pi−alpha1;
        F= quad(Q,alpha1,alpha2);
        Power=AA*(−BB*(alpha2−alpha1)+F);
%Case 3: Centre of circle less than or equal to −xo
%(left−most side of square)
    elseif xn≤−xo
        d=−xo−xn;
        kx=(2/W^2)*d^2;
        Q=@(alpha)exp(−kx./((sin(alpha)).^2));
        alpha1=asin(d/r);
        alpha2=pi−alpha1;
        F= quad(Q,alpha1,alpha2);
        Power=AA*(−BB*(alpha2−alpha1)+F);
%Case 4: Centre of circle between 0 and xo (right half of square)
    elseif xn>0 & xn<xo
        d=xo−xn;
        kx=(2/W^2)*d^2;
        Q=@(alpha)exp(−kx./((sin(alpha)).^2));
        alpha1=asin(d/r);
        alpha2=pi−alpha1;
        F= quad(Q,alpha1,alpha2);
        P1=AA*(−BB*(alpha2−alpha1)+F);
        Power=Pc−P1;
%Case 5: Centre of circle between −xo and 0 (left half of square)
    elseif xn<0 & xn>−xo
        d=xo+xn;
        kx=(2/W^2)*d^2;
        Q=@(alpha)exp(−kx./((sin(alpha)).^2));
        alpha1=asin(d/r);
        alpha2=pi−alpha1;
        F= quad(Q,alpha1,alpha2);
        P1=AA*(−BB*(alpha2−alpha1)+F);
        Power=Pc−P1;
%Case 6: Centre of circle at the origin of the coordinate system
    elseif xn==0
        Power=Pc;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the Power at the area of overlap    %%%
%%% between circular beam spot and the square photocell.       %%%
%%% Center of circular beam spot is moving along the y-axis    %%%
%%% while the square photocell is fixed at the origin          %%%
%%% of the coordinate system.                                  %%%
%%% Given, diameter of the circular beam spot is equivalent    %%%
%%% to one side of the square photocell.                       %%%
%%% filename = 'POWERY2.m'                                     %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*   Power = Power at the area of intersection between       *%
%*           circlular beam spot and square photocell, as the*%
%*           beam spot moves along y-axis.                   *%
%*************************************************************%

function Power=POWERY2(r,xn,yn,xint,yint,xo,yo,c)

lamda=6.33e-05;          %Wavelength of a He-Ne laser source in cm
%Spot size of the beam 2Wo=2/3cm, to get 99% of Total Power
Wo=1/3;
zo=(pi*Wo^2)/lamda;      %Rayleigh range in cm
z=0;                     %Axial distance z in cm
Io=1;                    %Maximum intensity value
W=Wo*sqrt(1+(z/zo)^2);   %Beam Width
I1=Io*(Wo/W)^2;
AA=I1*(W/2)^2;
BB=exp((-2*r^2)/W^2);



px=isreal(xint);         %Check if there is intersection (x-coordinate)
py=isreal(yint);         %Check if there is intersection (y-coordinate)
Pc=I1*(2*pi)*((W/2)^2)*(1-BB);       %Power of spot with radius r

%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(xint)==1)
    Power=0;
```

```
elseif px≠0 & py≠0
%Case 2: Centre of circle greater than or equal to yo
%(upper side of square)
    if yn≥yo
        d=yn-yo;
        ky=(2/W^2)*d^2;
        Q=@(beta)exp(-ky./((sin(beta)).^2));
        beta1=asin(d/r);
        beta2=pi-beta1;
        F= quad(Q,beta1,beta2);
        Power=AA*(-BB*(beta2-beta1)+F);
%Case 3: Centre of circle less than or equal to -yo
%(lower side of square)
    elseif yn≤-yo
        d=-yo-yn;
        ky=(2/W^2)*d^2;
        Q=@(beta)exp(-ky./((sin(beta)).^2));
        beta1=asin(d/r);
        beta2=pi-beta1;
        F= quad(Q,beta1,beta2);
        Power=AA*(-BB*(beta2-beta1)+F);
%Case 4: Centre of circle between 0 and yo (upper half of square)
    elseif yn>0 & yn<yo
        d=yo-yn;
        ky=(2/W^2)*d^2;
        Q=@(beta)exp(-ky./((sin(beta)).^2));
        beta1=asin(d/r);
        beta2=pi-beta1;
        F= quad(Q,beta1,beta2);
        P1=AA*(-BB*(beta2-beta1)+F);
        Power=Pc-P1;
%Case 5: Centre of circle between -yo and 0 (lower half of square)
    elseif yn<0 & yn>-yo
        d=yo+yn;
        ky=(2/W^2)*d^2;
        Q=@(beta)exp(-ky./((sin(beta)).^2));
        beta1=asin(d/r);
        beta2=pi-beta1;
        F= quad(Q,beta1,beta2);
        P1=AA*(-BB*(beta2-beta1)+F);
        Power=Pc-P1;
%Case 6: Centre of circle at the origin of the coordinate system
    elseif yn==0
        Power=Pc;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to calculate the Power at the area of overlap    %%%
%%% between circular beam spot and the square photocell.       %%%
%%% Center of circular beam spot is moving along both the      %%%
%%% x and y axis while the square photocell is fixed at the    %%%
%%% origin of the coordinate system.                           %%%
%%% Given, diameter of the circular beam spot is equivalent    %%%
%%% to one side of the square photocell.                       %%%
%%% filename = 'POWERXY2.m'                                    %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%*************************************************************%
%* Input to the function:                                    *%
%*       r = radius of circular beam spot.                   *%
%*      xn = x-coordinate of the center of the beam spot.    *%
%*      yn = y-coordinate of the center of the beam spot.    *%
%*    xint = x-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*    yint = y-coordinates of the intersection points between*%
%*           the circular beam spot and the square photocell.*%
%*      xo = x-coordinate of the right side of the square    *%
%*           photocell.                                      *%
%*      yo = y-coordinate of the upper side of the square    *%
%*           photocell.                                      *%
%*       c = cornenrs of the square inside the circular      *%
%*           beam spot.                                      *%
%* Output of the function:                                   *%
%*   Power = Power at the area of intersection between       *%
%*           circlular beam spot and square photocell, as the*%
%*           beam spot moves along both the x and y axis.    *%
%*************************************************************%

function Power=POWERXY2(r,xn,yn,xint,yint,xo,yo,c)

lamda=6.33e-05;          %Wavelength of a He-Ne laser source in cm
%Spot size of the beam 2Wo=2/3cm, to get 99% of Total Power
Wo=1/3;
zo=(pi*Wo^2)/lamda;      %Rayleigh range in cm
z=0;                     %Axial distance z in cm
Io=1;                    %Maximum intensity value
W=Wo*sqrt(1+(z/zo)^2);   %Beam Width
I1=Io*(Wo/W)^2;
AA=I1*(W/2)^2;
BB=exp((-2*r^2)/W^2);

Pc=I1*(2*pi)*((W/2)^2)*(1-BB);    %Power of spot with radius r


px=isreal(xint);    %Check if there is intersection (x-coordinate)
py=isreal(yint);    %Check if there is intersection (y-coordinate)

%Case 1: No intersection between circle and square OR just touching
if (px==0 & py==0)|(length(xint)==1)
    Power=0;
```

```
elseif px≠0 & py≠0
%Case 2: Center of circle at the origin of the coordinate system
  if xn==0 & yn==0
     Power=Pc;
%Case 3: Center of circle moving along the y-axis only
  elseif xn==0 & yn≠0
     Power=feval('POWERY2',r,xn,yn,xint,yint,xo,yo,c);
%Case 4: Center of circle moving along the x-axis only
  elseif xn≠0 & yn==0
     Power=feval('POWERX2',r,xn,yn,xint,yint,xo,yo,c);
%Case 5: Center of circle moving along both x and y axis
  elseif xn≠0 & yn≠0

    if xn>xo & yn>yo                          %Region 1
       dx=xn-xo;
       dy=yn-yo;
    elseif xn<-xo & yn>yo                     %Region 2
       dx=-xn-xo;
       dy=yn-yo;
    elseif xn<-xo & yn<-yo                    %Region 3
       dx=-xn-xo;
       dy=-yn-yo;
    elseif xn>xo & yn<-yo                     %Region 4
       dx=xn-xo;
       dy=-yn-yo;

    elseif xn≥xo & (yn≤yo & yn>0)             %Region 5
       dx=xn-xo;
       dy=yo-yn;
    elseif xn≥xo & (yn≥-yo & yn<0)            %Region 6
       dx=xn-xo;
       dy=yo+yn;
    elseif xn≤-xo & (yn≤yo & yn>0)            %Region 7
       dx=-xn-xo;
       dy=yo-yn;
    elseif xn≤-xo & (yn≥-yo & yn<0)           %Region 8
       dx=-xn-xo;
       dy=yo+yn;

    elseif yn≤-yo & (xn≤xo & xn>0)            %Region 9
       dx=xo-xn;
       dy=-yn-yo;
    elseif yn≤-yo & (xn≥-xo & xn<0)           %Region 10
       dx=xo+xn;
       dy=-yn-yo;
    elseif yn≥yo & (xn≤xo & xn>0)             %Region 11
       dx=xo-xn;
       dy=yn-yo;
    elseif yn≥yo & (xn≥-xo & xn<0)            %Region 12
       dx=xo+xn;
       dy=yn-yo;

    elseif (xn<xo & xn>0) & (yn<yo & yn>0)    %Region 13
       dx=xo-xn;
       dy=yo-yn;
    elseif (xn>-xo & xn<0) & (yn<yo & yn>0)   %Region 14
       dx=xo+xn;
       dy=yo-yn;
```

```
elseif (xn>=-xo & xn<0) & (yn>=-yo & yn<0) %Region 15
   dx=xo+xn;
   dy=yo+yn;
elseif (xn<xo & xn>0) & (yn>=-yo & yn<0)  %Region 16
   dx=xo-xn;
   dy=yo+yn;
end

alpha1=asin(dx/r);
alpha2=pi-alpha1;

beta1=asin(dy/r);
beta2=pi-beta1;

phi=atan(dy/dx);

kx=(2/W^2)*dx^2;
ky=(2/W^2)*dy^2;

Qx=@(alpha)exp(-kx./((sin(alpha)).^2));
Qy=@(beta)exp(-ky./((sin(beta)).^2));

Fx=quad(Qx,(phi+(pi/2)),alpha2);
Fy=quad(Qy,(pi-phi),beta2);
Pxy=AA*(-BB*((pi/2)-(alpha1+beta1))+Fx+Fy);

Fxx=quad(Qx,alpha1,alpha2);
Px=AA*(-BB*(alpha2-alpha1)+Fxx);

Fyy=quad(Qy,beta1,beta2);
Py=AA*(-BB*(beta2-beta1)+Fyy);

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 1, 2, 3 or 4:
if(xn>xo & yn>yo)|(xn>xo & yn<-yo)|
  (xn<-xo & yn>yo)|(xn<-xo & yn<-yo)
  Power=Pxy;

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 5, 6, 7 or 8:
elseif(xn>=xo & (yn<=yo & yn>0))|(xn>=xo & (yn>=-yo & yn<0))|
      (xn<=-xo & (yn<=yo & yn>0))|(xn<=-xo & (yn>=-yo & yn<0))
        if length(c)==1 & length(xint)==2
            if dx==0 & dy==0
                Power=Pc/4;
            else
                Power=Px-Pxy;
            end
        elseif length(c)==0 & length(xint)==2
            Power=Px;
        end

%% Coordinates of the center of the circle (xn,yn) are located
%% in Region 9, 10, 11 or 12:
elseif (yn>=yo & (xn<=xo & xn>0))|(yn>=yo & (xn>=-xo & xn<0))|
       (yn<=-yo & (xn<=xo & xn>0))|(yn<=-yo & (xn>=-xo & xn<0))
        if length(c)==1 & length(xint)==2
            if dx==0 & dy==0
```

```
                    Power=Pc/4;
                else
                    Power=Py-Pxy;
                end
            elseif length(c)==0 & length(xint)==2
                Power=Py;
            end

    %% Coordinates of the center of the circle (xn,yn) are located
    %% in Region 13, 14, 15 or 16:
    elseif((xn<xo & xn>0) & (yn<yo & yn>0))|
          ((xn>-xo & xn<0) & (yn<yo & yn>0))|
          ((xn<xo & xn>0) & (yn>-yo & yn<0))|
          ((xn>-xo & xn<0) & (yn>-yo & yn<0))
            if length(c)==1 & length(xint)==2
                Power=Pc-(Py+Px-Pxy);
            elseif length(c)==0 & length(xint)==4
                Power=Pc-(Py+Px);
            end
    end
  end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Power calculation program for a quad-cell photodetector   %%%
%%% array at a certain epsilon and Δ, while moving the        %%%
%%% beam spot along both the x and y axis.                    %%%
%%% filename = 'Quadcell_POWER.m'                             %%%
%%%                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all

epsn=0.1;       %Epsilon (horizontal gap)
dlta=0.3;       %Delta (vertical gap)

xs1=0.5+(epsn/2);ys1=0.5+(dlta/2);       %Centre of photocell 1
xs2=-0.5-(epsn/2);ys2=0.5+(dlta/2);      %Centre of photocell 2
xs3=-0.5-(epsn/2);ys3=-0.5-(dlta/2);     %Centre of photocell 3
xs4=0.5+(epsn/2);ys4=-0.5-(dlta/2);      %Centre of photocell 4

xc=-2:0.01:2;           %x-coordinate of the beam spot center
yc=-2:0.01:2;           %y-coordinate of the beam spot center
nxc=length(xc);
nyc=length(yc);

Xc(1,:)=xc-xs1;Yc(1,:)=yc-ys1;
Xc(2,:)=xc-xs2;Yc(2,:)=yc-ys2;
Xc(3,:)=xc-xs3;Yc(3,:)=yc-ys3;
Xc(4,:)=xc-xs4;Yc(4,:)=yc-ys4;

load 'Pcentre'      %Normalized power for one photocell
V=-3:0.01:3;        %Range for the x and y plane for Pcentre
nxV=length(V);
nyV=nxV;

vx(1,1)=find(abs(V-(Xc(1,1)))≥0 & abs(V-(Xc(1,1)))≤0.001);
vx(1,2)=find(abs(V-(Xc(1,nxc)))≥0 & abs(V-(Xc(1,nxc)))≤0.001);

vx(2,1)=find(abs(V-(Xc(2,1)))≥0 & abs(V-(Xc(2,1)))≤0.001);
vx(2,2)=find(abs(V-(Xc(2,nxc)))≥0 & abs(V-(Xc(2,nxc)))≤0.001);

vx(3,1)=find(abs(V-(Xc(3,1)))≥0 & abs(V-(Xc(3,1)))≤0.001);
vx(3,2)=find(abs(V-(Xc(3,nxc)))≥0 & abs(V-(Xc(3,nxc)))≤0.001);

vx(4,1)=find(abs(V-(Xc(4,1)))≥0 & abs(V-(Xc(4,1)))≤0.001);
vx(4,2)=find(abs(V-(Xc(4,nxc)))≥0 & abs(V-(Xc(4,nxc)))≤0.001);

vy(1,1)=find(abs(V-(Yc(1,1)))≥0 & abs(V-(Yc(1,1)))≤0.001);
vy(1,2)=find(abs(V-(Yc(1,nyc)))≥0 & abs(V-(Yc(1,nyc)))≤0.001);

vy(2,1)=find(abs(V-(Yc(2,1)))≥0 & abs(V-(Yc(2,1)))≤0.001);
vy(2,2)=find(abs(V-(Yc(2,nyc)))≥0 & abs(V-(Yc(2,nyc)))≤0.001);

vy(3,1)=find(abs(V-(Yc(3,1)))≥0 & abs(V-(Yc(3,1)))≤0.001);
vy(3,2)=find(abs(V-(Yc(3,nyc)))≥0 & abs(V-(Yc(3,nyc)))≤0.001);
```

```
vy(4,1)=find(abs(V-(Yc(4,1)))>=0 & abs(V-(Yc(4,1)))<=0.001);
vy(4,2)=find(abs(V-(Yc(4,nyc)))>=0 & abs(V-(Yc(4,nyc)))<=0.001);

dx(1,1)=vx(1,1)-1;dx(1,2)=nxV-vx(1,2);
dx(2,1)=vx(2,1)-1;dx(2,2)=nxV-vx(2,2);
dx(3,1)=vx(3,1)-1;dx(3,2)=nxV-vx(3,2);
dx(4,1)=vx(4,1)-1;dx(4,2)=nxV-vx(4,2);

Dx(1,1)=dx(1,1)+1;Dx(1,2)=nxV-dx(1,2);
Dx(2,1)=dx(2,1)+1;Dx(2,2)=nxV-dx(2,2);
Dx(3,1)=dx(3,1)+1;Dx(3,2)=nxV-dx(3,2);
Dx(4,1)=dx(4,1)+1;Dx(4,2)=nxV-dx(4,2);

dy(1,1)=vy(1,1)-1;dy(1,2)=nyV-vy(1,2);
dy(2,1)=vy(2,1)-1;dy(2,2)=nyV-vy(2,2);
dy(3,1)=vy(3,1)-1;dy(3,2)=nyV-vy(3,2);
dy(4,1)=vy(4,1)-1;dy(4,2)=nyV-vy(4,2);

Dy(1,1)=dy(1,1)+1;Dy(1,2)=nyV-dy(1,2);
Dy(2,1)=dy(2,1)+1;Dy(2,2)=nyV-dy(2,2);
Dy(3,1)=dy(3,1)+1;Dy(3,2)=nyV-dy(3,2);
Dy(4,1)=dy(4,1)+1;Dy(4,2)=nyV-dy(4,2);

PS1=Pcentre(Dx(1,1):Dx(1,2),Dy(1,1):Dy(1,2));
PS2=Pcentre(Dx(2,1):Dx(2,2),Dy(2,1):Dy(2,2));
PS3=Pcentre(Dx(3,1):Dx(3,2),Dy(3,1):Dy(3,2));
PS4=Pcentre(Dx(4,1):Dx(4,2),Dy(4,1):Dy(4,2));
figure
%Plot of PS1 vs. position of centre of beam spot
mesh(xc,yc,PS1)
xlabel('x (cm)')
ylabel('y (cm)')
zlabel('Normalized Power')
colorbar

figure
%Plot of PS2 vs. position of centre of beam spot
mesh(xc,yc,PS2)
xlabel('x (cm)')
ylabel('y (cm)')
zlabel('Normalized Power')
colorbar

figure
%Plot of PS3 vs. position of centre of beam spot
mesh(xc,yc,PS3)
xlabel('x (cm)')
ylabel('y (cm)')
zlabel('Normalized Power')
colorbar

figure
%Plot of PS4 vs. position of centre of beam spot
mesh(xc,yc,PS4)
xlabel('x (cm)')
ylabel('y (cm)')
zlabel('Normalized Power')
colorbar
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Power calculation program for a quad-cell photodetector   %%%
%%% array at different values for epsilon and Δ, while        %%%
%%% moving the beam spot along both the x and y axis.         %%%
%%% Epsilon and Δ are assumed to be equal.                    %%%
%%% filename = 'Quadcell_POWER_modifiedfurther.m'             %%%
%%%                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all

for k=1:10
  epsn=0.1*(k-1);     %Epsilon (horizontal gap)
  dlta=0.1*(k-1);     %Delta (vertical gap)

  xs1=0.5+(epsn/2);ys1=0.5+(dlta/2);       %Centre of photocell 1
  xs2=-0.5-(epsn/2);ys2=0.5+(dlta/2);      %Centre of photocell 2
  xs3=-0.5-(epsn/2);ys3=-0.5-(dlta/2);     %Centre of photocell 3
  xs4=0.5+(epsn/2);ys4=-0.5-(dlta/2);      %Centre of photocell 4

  xc=-2:0.01:2;           %x-coordinate of the beam spot center
  yc=-2:0.01:2;           %y-coordinate of the beam spot center
  nxc=length(xc);
  nyc=length(yc);

  Xc(1,:)=xc-xs1;Yc(1,:)=yc-ys1;
  Xc(2,:)=xc-xs2;Yc(2,:)=yc-ys2;
  Xc(3,:)=xc-xs3;Yc(3,:)=yc-ys3;
  Xc(4,:)=xc-xs4;Yc(4,:)=yc-ys4;

  load 'Pcentre'      %Normalized power for one photocell
  V=-3:0.01:3;        %Range for the x and y plane for Pcentre
  nxV=length(V);
  nyV=nxV;

  vx(1,1)=find(abs(V-(Xc(1,1)))≥0 & abs(V-(Xc(1,1)))≤0.001);
  vx(1,2)=find(abs(V-(Xc(1,nxc)))≥0 & abs(V-(Xc(1,nxc)))≤0.001);

  vx(2,1)=find(abs(V-(Xc(2,1)))≥0 & abs(V-(Xc(2,1)))≤0.001);
  vx(2,2)=find(abs(V-(Xc(2,nxc)))≥0 & abs(V-(Xc(2,nxc)))≤0.001);

  vx(3,1)=find(abs(V-(Xc(3,1)))≥0 & abs(V-(Xc(3,1)))≤0.001);
  vx(3,2)=find(abs(V-(Xc(3,nxc)))≥0 & abs(V-(Xc(3,nxc)))≤0.001);

  vx(4,1)=find(abs(V-(Xc(4,1)))≥0 & abs(V-(Xc(4,1)))≤0.001);
  vx(4,2)=find(abs(V-(Xc(4,nxc)))≥0 & abs(V-(Xc(4,nxc)))≤0.001);

  vy(1,1)=find(abs(V-(Yc(1,1)))≥0 & abs(V-(Yc(1,1)))≤0.001);
  vy(1,2)=find(abs(V-(Yc(1,nyc)))≥0 & abs(V-(Yc(1,nyc)))≤0.001);

  vy(2,1)=find(abs(V-(Yc(2,1)))≥0 & abs(V-(Yc(2,1)))≤0.001);
  vy(2,2)=find(abs(V-(Yc(2,nyc)))≥0 & abs(V-(Yc(2,nyc)))≤0.001);
```

```
  vy(3,1)=find(abs(V-(Yc(3,1)))≥0 & abs(V-(Yc(3,1)))≤0.001);
  vy(3,2)=find(abs(V-(Yc(3,nyc)))≥0 & abs(V-(Yc(3,nyc)))≤0.001);

  vy(4,1)=find(abs(V-(Yc(4,1)))≥0 & abs(V-(Yc(4,1)))≤0.001);
  vy(4,2)=find(abs(V-(Yc(4,nyc)))≥0 & abs(V-(Yc(4,nyc)))≤0.001);

  dx(1,1)=vx(1,1)-1;dx(1,2)=nxV-vx(1,2);
  dx(2,1)=vx(2,1)-1;dx(2,2)=nxV-vx(2,2);
  dx(3,1)=vx(3,1)-1;dx(3,2)=nxV-vx(3,2);
  dx(4,1)=vx(4,1)-1;dx(4,2)=nxV-vx(4,2);

  Dx(1,1)=dx(1,1)+1;Dx(1,2)=nxV-dx(1,2);
  Dx(2,1)=dx(2,1)+1;Dx(2,2)=nxV-dx(2,2);
  Dx(3,1)=dx(3,1)+1;Dx(3,2)=nxV-dx(3,2);
  Dx(4,1)=dx(4,1)+1;Dx(4,2)=nxV-dx(4,2);

  dy(1,1)=vy(1,1)-1;dy(1,2)=nyV-vy(1,2);
  dy(2,1)=vy(2,1)-1;dy(2,2)=nyV-vy(2,2);
  dy(3,1)=vy(3,1)-1;dy(3,2)=nyV-vy(3,2);
  dy(4,1)=vy(4,1)-1;dy(4,2)=nyV-vy(4,2);

  Dy(1,1)=dy(1,1)+1;Dy(1,2)=nyV-dy(1,2);
  Dy(2,1)=dy(2,1)+1;Dy(2,2)=nyV-dy(2,2);
  Dy(3,1)=dy(3,1)+1;Dy(3,2)=nyV-dy(3,2);
  Dy(4,1)=dy(4,1)+1;Dy(4,2)=nyV-dy(4,2);

  PS1(:,:,k)=Pcentre(Dx(1,1):Dx(1,2),Dy(1,1):Dy(1,2));
  PS2(:,:,k)=Pcentre(Dx(2,1):Dx(2,2),Dy(2,1):Dy(2,2));
  PS3(:,:,k)=Pcentre(Dx(3,1):Dx(3,2),Dy(3,1):Dy(3,2));
  PS4(:,:,k)=Pcentre(Dx(4,1):Dx(4,2),Dy(4,1):Dy(4,2));

  ps1(k)=PS1(201,201,k);

end

epsn=0:0.1:0.9;
%Plot of normalized power of photocell 1 at (0,0) vs. Epsilon
plot(epsn,ps1)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Power calculation program for a quad-cell photodetector   %%%
%%% array at different values for Epsilon and Delta, while    %%%
%%% moving the beam spot along both the x and y axis.         %%%
%%% filename = 'Quadcell_POWER_modifiedfurther2D.m'           %%%
%%%                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2007                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all

for k=1:10
  for p=1:10
    epsn=0.1*(k-1);     %Epsilon (horizontal gap)
    dlta=0.1*(p-1);     %Delta (vertical gap)
    xs1=0.5+(epsn/2);ys1=0.5+(dlta/2);       %Centre of photocell 1
    xs2=-0.5-(epsn/2);ys2=0.5+(dlta/2);      %Centre of photocell 2
    xs3=-0.5-(epsn/2);ys3=-0.5-(dlta/2);     %Centre of photocell 3
    xs4=0.5+(epsn/2);ys4=-0.5-(dlta/2);      %Centre of photocell 4

    xc=-2:0.01:2;           %x-coordinate of the beam spot center
    yc=-2:0.01:2;           %y-coordinate of the beam spot center

    nxc=length(xc);
    nyc=length(yc);

    Xc(1,:)=xc-xs1;Yc(1,:)=yc-ys1;
    Xc(2,:)=xc-xs2;Yc(2,:)=yc-ys2;
    Xc(3,:)=xc-xs3;Yc(3,:)=yc-ys3;
    Xc(4,:)=xc-xs4;Yc(4,:)=yc-ys4;

    load 'Pcentre'      %Normalized power for one photocell
    V=-3:0.01:3;        %Range for the x and y plane for Pcentre
    nxV=length(V);
    nyV=nxV;

    vx(1,1)=find(abs(V-(Xc(1,1)))>=0 & abs(V-(Xc(1,1)))<=0.001);
    vx(1,2)=find(abs(V-(Xc(1,nxc)))>=0 & abs(V-(Xc(1,nxc)))<=0.001);

    vx(2,1)=find(abs(V-(Xc(2,1)))>=0 & abs(V-(Xc(2,1)))<=0.001);
    vx(2,2)=find(abs(V-(Xc(2,nxc)))>=0 & abs(V-(Xc(2,nxc)))<=0.001);

    vx(3,1)=find(abs(V-(Xc(3,1)))>=0 & abs(V-(Xc(3,1)))<=0.001);
    vx(3,2)=find(abs(V-(Xc(3,nxc)))>=0 & abs(V-(Xc(3,nxc)))<=0.001);

    vx(4,1)=find(abs(V-(Xc(4,1)))>=0 & abs(V-(Xc(4,1)))<=0.001);
    vx(4,2)=find(abs(V-(Xc(4,nxc)))>=0 & abs(V-(Xc(4,nxc)))<=0.001);

    vy(1,1)=find(abs(V-(Yc(1,1)))>=0 & abs(V-(Yc(1,1)))<=0.001);
    vy(1,2)=find(abs(V-(Yc(1,nyc)))>=0 & abs(V-(Yc(1,nyc)))<=0.001);

    vy(2,1)=find(abs(V-(Yc(2,1)))>=0 & abs(V-(Yc(2,1)))<=0.001);
    vy(2,2)=find(abs(V-(Yc(2,nyc)))>=0 & abs(V-(Yc(2,nyc)))<=0.001);

    vy(3,1)=find(abs(V-(Yc(3,1)))>=0 & abs(V-(Yc(3,1)))<=0.001);
```

```
    vy(3,2)=find(abs(V-(Yc(3,nyc)))>=0 & abs(V-(Yc(3,nyc)))<=0.001);

    vy(4,1)=find(abs(V-(Yc(4,1)))>=0 & abs(V-(Yc(4,1)))<=0.001);
    vy(4,2)=find(abs(V-(Yc(4,nyc)))>=0 & abs(V-(Yc(4,nyc)))<=0.001);

    dx(1,1)=vx(1,1)-1;dx(1,2)=nxV-vx(1,2);
    dx(2,1)=vx(2,1)-1;dx(2,2)=nxV-vx(2,2);
    dx(3,1)=vx(3,1)-1;dx(3,2)=nxV-vx(3,2);
    dx(4,1)=vx(4,1)-1;dx(4,2)=nxV-vx(4,2);

    Dx(1,1)=dx(1,1)+1;Dx(1,2)=nxV-dx(1,2);
    Dx(2,1)=dx(2,1)+1;Dx(2,2)=nxV-dx(2,2);
    Dx(3,1)=dx(3,1)+1;Dx(3,2)=nxV-dx(3,2);
    Dx(4,1)=dx(4,1)+1;Dx(4,2)=nxV-dx(4,2);

    dy(1,1)=vy(1,1)-1;dy(1,2)=nyV-vy(1,2);
    dy(2,1)=vy(2,1)-1;dy(2,2)=nyV-vy(2,2);
    dy(3,1)=vy(3,1)-1;dy(3,2)=nyV-vy(3,2);
    dy(4,1)=vy(4,1)-1;dy(4,2)=nyV-vy(4,2);

    Dy(1,1)=dy(1,1)+1;Dy(1,2)=nyV-dy(1,2);
    Dy(2,1)=dy(2,1)+1;Dy(2,2)=nyV-dy(2,2);
    Dy(3,1)=dy(3,1)+1;Dy(3,2)=nyV-dy(3,2);
    Dy(4,1)=dy(4,1)+1;Dy(4,2)=nyV-dy(4,2);

    PS1(:,:)=Pcentre(Dx(1,1):Dx(1,2),Dy(1,1):Dy(1,2));
    PS2(:,:)=Pcentre(Dx(2,1):Dx(2,2),Dy(2,1):Dy(2,2));
    PS3(:,:)=Pcentre(Dx(3,1):Dx(3,2),Dy(3,1):Dy(3,2));
    PS4(:,:)=Pcentre(Dx(4,1):Dx(4,2),Dy(4,1):Dy(4,2));

    ps1(k,p)=PS1(201,201);
  end
end

epsn=0:0.1:0.9;
dlta=0:0.1:0.9;

%2D Plot of normalized power of photocell 1 at (0,0) vs.
%Epsilon and Delta
surf(epsn,dlta,ps1)
colormap winter
axis([0 0.9 0 0.9 0 0.25])
xlabel('Epsilon (cm)')
ylabel('Delta (cm)')
zlabel('Normalized Power')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet network training for multiple input multiple      %%%
%%% output (MIMO) function approximation using dyadic grid    %%%
%%% for WN initialization.                                     %%%
%%% filename = 'wavnet52_dyadic_initialization.m'              %%%
%%%                                                           %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clear all
clc

load 'inoutzero.txt' %Loading input/output simulated data (no gaps)
% load 'inout.txt'   %Loading input/output simulated data (with gaps)
%  load 'inoutexp'   %Loading input/output experimental data

tic

%-------------------------------------------------------------------------%
%                      WN Initialization                                  %
%-------------------------------------------------------------------------%

%Input simulated data (no gaps)
x=[inoutzero(:,3),inoutzero(:,4),inoutzero(:,5),inoutzero(:,6)];
%Desired simulated data (no gaps)
Y=[inoutzero(:,1),inoutzero(:,2)];

%Input simulated data (with gaps)
% x=[inout(:,3),inout(:,4),inout(:,5),inout(:,6)];
 %Desired simulated data (with gaps)
% Y=[inout(:,1),inout(:,2)];

%Input experimental data
% x=[inoutexp(:,3),inoutexp(:,4),inoutexp(:,5),inoutexp(:,6)];
%Desired experimental data
% Y=[inoutexp(:,1),inoutexp(:,2)];

%Applying preprocessing condition on data:
nzi1=find((x(:,1)≠0 & x(:,2)≠0 & x(:,3)≠0 & x(:,4)≠0));
for ni1=1:length(nzi1)
    x_new1(ni1,:)=x(nzi1(ni1),:);
    Y_new1(ni1,:)=Y(nzi1(ni1),:);
end
x=x_new1;   %Input data after condition
Y=Y_new1;   %Desired output data after condition


tstr=find(Y(:,1)==0);
%Input data used for testing
x_test=x(tstr(1):tstr(end),:);
%Desired output data used for testing
Y_test=Y(tstr(1):tstr(end),:)';
%Input data used for training
x=[x(1:tstr(1)-1,:);x(tstr(end)+1:end,:)];
%Desired output data used for training
```

```
Y=[Y(1:tstr(1)-1,:);Y(tstr(end)+1:end,:)];

Y=Y';
u=0.01;          %Setting the learning rate
gamma=1-u;       %Setting the momentum coefficient
Ax=size(x);
Ay=size(Y);
Np=Ax(1,1);      %no. of patterns
Ni=Ax(1,2);      %no. of input nodes
Nl=3;            %no. of levels
Nw=(2^Nl)-1;     %no. of hidden nodes
No=Ay(1,1);      %no. of output nodes
I=ones(1,Np);
input=[I; x'];   %input matrix
C=input';

%Initializing Woi using Least Squares Method
for k=1:No
    Woi1=(inv(C'*C))*C'*Y(k,:)';
    Woi1=Woi1';
    Woi(k,:)=Woi1;
end

%Initializing Woh to zeros
Woh=zeros(No,Nw);

%Initializing translation and dialation parameters
%using dyadic grid
ak=min(x);
bk=max(x);

for i=1:Ni
    for L=1:Nl
        div=(bk(i)-ak(i))/(2^L);
        n=1;
        p=0;
        while p~=bk(i)
            p=ak(1,i)+(n*div);
            M(n)=p;
            D(n)=div;
            n=n+1;
        end

        for w=1:(n/2)
            f(1,w,L)=M(2*w-1);
            g(1,w,L)=D(2*w-1);
        end
    end
    F=squeeze(f);
    G=squeeze(g);
    [pf,qf]=find(F~=0);
    [pg,qg]=find(G~=0);

    for w=1:length(pf)
        s(w)=F(pf(w),qf(w));
    end
    if F(1,1)==0
        m(:,i)=[F(1,1),s];
```

```
    elseif F(1,1)≠0
        m(:,i)=s;
    end

    for w=1:length(pg)
        t(w)=G(pg(w),qg(w));
    end
    if G(1,1)==0
        d(:,i)=[G(1,1),t];
    elseif G(1,1)≠0
        d(:,i)=t;
    end
end

DWoi=zeros(No,Ni+1,Np);
DWoh=zeros(No,Nw,Np);
Dm=zeros(Nw,Ni,Np);
Dd=zeros(Nw,Ni,Np);

DWoh_aver=zeros(No,Nw);
DWoi_aver=zeros(No,Ni+1);
Dm_aver=zeros(Nw,Ni);
Dd_aver=zeros(Nw,Ni);

DeltaWoi_old=0*Woi;
DeltaWoh_old=0*Woh;
Deltam_old=0*m;
Deltad_old=0*d;

iterations=10^5;

z=zeros(Nw,Ni,Np);

phi=zeros(Nw,Ni,Np);
phi_p=zeros(Nw,Ni,Np);
PHI=zeros(1,Nw,Np);
PHI_p=zeros(Nw,Ni,Np);

Y_hat=zeros(No,Np);
E=zeros(No,Np);
MSE=zeros(1,iterations);

for loops=1:iterations
loops
%-------------------------------------------------------------------------%
%                      Feedforward algorithm                              %
%-------------------------------------------------------------------------%

PHI=ones(1,Nw,Np);
for j=1:Nw
    for i=1:Ni
        z(j,i,:)=(x(:,i)-m(j,i))./d(j,i);
        phi(j,i,:)=(-z(j,i,:)).*exp(-0.5*(z(j,i,:).^2));
        phi_p(j,i,:)=(((z(j,i,:)).^2)-1).*exp(-0.5*(z(j,i,:).^2));
        PHI(1,j,:)=PHI(1,j,:).*phi(j,i,:);
    end
end
```

```
%Use squeeze command on PHI to reduce it from a 3D to a 2D matrix
PHI=squeeze(PHI);

%Batch computation of the wavelet network output Y_hat
%using the feedforward equation
Y_hat=(Woh*PHI) + (Woi*input);

%Computing the error E between the desired output Y
%and the WN output Y_hat
E=Y-Y_hat;

%Computing the sum of square error (SSE)
SSE=sum(sum(E.*E));

%Computing the mean square error (MSE) for every iteration
MSE(loops)=SSE/Np;

%-------------------------------------------------------------------------%
%                      Backpropagation algorithm                          %
%-------------------------------------------------------------------------%

for k=1:No
    DWoi(k,1,:)=E(k,:)*u;
end

for k=1:No
    for i=1:Ni
     DWoi(k,i+1,:)=E(k,:)*u.*x(:,i)';
    end
end

for k=1:No
    for j=1:Nw
     DWoh(k,j,:)=E(k,:)*u.*PHI(j,:);
    end
end

for j=1:Nw
    for i=1:Ni
        P=phi(j,:,:);
        P(1,i,:)=phi_p(j,i,:);
        PHI_p(j,i,:)=prod(P);
    end
end

for j=1:Nw
    for i=1:Ni
    EDmy=zeros(1,1,Np);
    EDdy=zeros(1,1,Np);
     for k=1:No
         p=squeeze(PHI_p(j,i,:));
         Dmy(k,:)=-(Woh(k,j)/d(j,i)).*p;
         zz=squeeze(z(j,i,:));
         Ddy(k,:)=Dmy(k,:).*zz';
         eDmy(1,1,:)=E(k,:).*Dmy(k,:);
         eDdy(1,1,:)=E(k,:).*Ddy(k,:);
         EDmy=EDmy+eDmy;
         EDdy=EDdy+eDdy;
```

```
     end
     Dm(j,i,:)=EDmy;
     Dd(j,i,:)=EDdy;
    end
end

for k=1:No
    for j=1:Nw
        DWoh_aver(k,j)=mean(DWoh(k,j,:));
    end
end

for k=1:No
    for i=1:Ni+1
        DWoi_aver(k,i)=mean(DWoi(k,i,:));
    end
end

for j=1:Nw
    for i=1:Ni
        Dm_aver(j,i)=mean(Dm(j,i,:));
        Dd_aver(j,i)=mean(Dd(j,i,:));
    end
end

DeltaWoi=DWoi_aver+gamma*DeltaWoi_old;
DeltaWoh=DWoh_aver+gamma*DeltaWoh_old;
Deltam=Dm_aver+gamma*Deltam_old;
Deltad=Dd_aver+gamma*Deltad_old;

%Updating Woi, Woh, m and d
Woi=Woi+DeltaWoi;
Woh=Woh+DeltaWoh;
m=m+Deltam;
d=d+Deltad;

DeltaWoi_old=DeltaWoi;
DeltaWoh_old=DeltaWoh;
Deltam_old=Deltam;
Deltad_old=Deltad;

end

figure(1)
        loglog(MSE,'*')
        xlabel('Iterations')
        ylabel('MSE')
        grid on
save 'RP,simulated data no gaps,w=gaussian,Nl=5,Nw=31,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_05'
figure(2)
plot3(Y_hat(1,:),Y_hat(2,:),x(:,1))
hold on
plot3(Y(1,:),Y(2,:),x(:,1),'r')
xlabel('x')
ylabel('y')
zlabel('P1')
```

```
figure(3)
plot3(Y_hat(1,:),Y_hat(2,:),x(:,2))
hold on
plot3(Y(1,:),Y(2,:),x(:,2),'r')
xlabel('x')
ylabel('y')
zlabel('P2')

figure(4)
plot3(Y_hat(1,:),Y_hat(2,:),x(:,3))
hold on
plot3(Y(1,:),Y(2,:),x(:,3),'r')
xlabel('x')
ylabel('y')
zlabel('P3')

figure(5)
plot3(Y_hat(1,:),Y_hat(2,:),x(:,4))
hold on
plot3(Y(1,:),Y(2,:),x(:,4),'r')
xlabel('x')
ylabel('y')
zlabel('P4')

figure(6)
plot(Y_hat(1,:),'LineStyle',':','LineWidth',2,'Color',[0 0 0])
hold on
plot(Y(1,:),'LineWidth',2,'Color',[0 0 0])
xlabel('N_p')
ylabel('x')
legend('Network output','Desired output')

figure(7)
plot(Y_hat(2,:),'LineStyle',':','LineWidth',2,'Color',[0 0 0])
hold on
plot(Y(2,:),'LineWidth',2,'Color',[0 0 0])
xlabel('N_p')
ylabel('y')
legend('Network output','Desired output')

figure(8)
f=find(Y(1,:)==0);
handlevector(1)=plot(Y(2,f(1):f(end)),x(f(1):f(end),1),
'LineWidth',2,'Color',[0 0 0],'DisplayName','P1 Desired Output');
hold on
handlevector(2)=plot(Y_hat(2,f(1):f(end)),x(f(1):f(end),1),
'Marker','^','LineStyle','none','Color',[0 0 0],
'DisplayName','P1 WN output');
xlabel('y (cm)')
ylabel('Normalized Power')
legend(handlevector([1 2]))

figure(9)
handlevector(1)=plot(Y(2,f(1):f(end)),x(f(1):f(end),2),
'LineWidth',2,'Color',[0 0 0],'DisplayName','P2 Desired Output');
hold on
handlevector(2)=plot(Y_hat(2,f(1):f(end)),x(f(1):f(end),2),
'Marker','o','LineStyle','none','Color',[0 0 0],
```

```
'DisplayName','P2 WN output');
xlabel('y (cm)')
ylabel('Normalized Power')
legend(handlevector([1 2]))

figure(10)
handlevector(1)=plot(Y(2,f(1):f(end)),x(f(1):f(end),3),
'LineWidth',2,'Color',[0 0 0],'DisplayName','P3 Desired Output');
hold on
handlevector(2)=plot(Y_hat(2,f(1):f(end)),x(f(1):f(end),3),
'Marker','square','LineStyle','none','Color',[0 0 0],
'DisplayName','P3 WN output');
xlabel('y (cm)')
ylabel('Normalized Power')
legend(handlevector([1 2]))

figure(11)
hold on
handlevector(1)=plot(Y(2,f(1):f(end)),x(f(1):f(end),4),
'LineWidth',2,'Color',[0 0 0],'DisplayName','P4 Desired Output');
hold on
handlevector(2)=plot(Y_hat(2,f(1):f(end)),x(f(1):f(end),4),
'Marker','*','LineStyle','none','Color',[0 0 0],
'DisplayName','P4 WN output');
xlabel('y (cm)')
ylabel('Normalized Power')
legend(handlevector([1 2]))

toc
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Wavelet network testing for multiple input multiple                    %%%
%%% output (MIMO) function approximation using dyadic grid                 %%%
%%% for WN initialization.                                                  %%%
%%% filename = 'wavnet52_forward_testing.m'                                 %%%
%%%                                                                         %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
clc
load 'simulated data with gaps,w=gaussian,Nl=6,Nw=63,gamma=0_99,u=0_01,
loops=10to4,LS_init,Wavelon_init,testing_neg0_55,divx=0_1,divy=0_02'
load 'x_test'
load 'Y_test'
N=size(x_test);
Np=N(1,1);
I=ones(1,Np);
input_tst=[I; x_test'];
z=zeros(Nw,Ni,Np);

phi=zeros(Nw,Ni,Np);
phi_p=zeros(Nw,Ni,Np);

PHI_p=zeros(Nw,Ni,Np);

Y_hat_tst=zeros(No,Np);
E=zeros(No,Np);


PHI=ones(1,Nw,Np);
for j=1:Nw
    for i=1:Ni
        z(j,i,:)=(x_test(:,i)-m(j,i))./d(j,i);
        phi(j,i,:)=(-z(j,i,:)).*exp(-0.5*(z(j,i,:).^2));
        phi_p(j,i,:)=(((z(j,i,:)).^2)-1).*exp(-0.5*(z(j,i,:).^2));
        PHI(1,j,:)=PHI(1,j,:).*phi(j,i,:);
    end
end
PHI=squeeze(PHI);
Y_hat_tst=(Woh*PHI) + (Woi*input_tst);
E=Y_test-Y_hat_tst;

SSE=sum(sum(E.*E));

MSE=SSE/Np;
MSE_test6=MSE;
save('MSE_test,simulated data with gaps,w=gaussian,Nl=6,Nw=63,
gamma=0_99,u=0_01,loops=10to4,LS_init,Wavelon_init,
testing_neg0_55,divx=0_1,divy=0_02', 'MSE_test6')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Comparing the MSE values after training simulated data     %%%
%%% with gaps and testing for simulated data set at x=0 cm.    %%%
%%% filename = 'wavnet52_mse_reverse_simwithgaps_               %%%
%%%              divx0_05_divy0_02_testing.m'                   %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=2,Nw=3,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=2,Nw=3,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(1)=MSE(end);
MSE_test(1)=MSE_RPtest2;

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=3,Nw=7,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=3,Nw=7,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(2)=MSE(end);
MSE_test(2)=MSE_RPtest3;

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=4,Nw=15,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=4,Nw=15,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(3)=MSE(end);
MSE_test(3)=MSE_RPtest4;

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=5,Nw=31,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=5,Nw=31,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(4)=MSE(end);
MSE_test(4)=MSE_RPtest5;

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=6,Nw=63,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=6,Nw=63,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(5)=MSE(end);
MSE_test(5)=MSE_RPtest6;

load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=7,Nw=127,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=7,Nw=127,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
MSE_train(6)=MSE(end);
MSE_test(6)=MSE_RPtest7;
%
load 'MSE_test,RP,simulated data gaps,w=gaussian,Nl=8,Nw=255,gamma=0_99,
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02'
load 'RP,simulated data gaps,w=gaussian,Nl=8,Nw=255,gamma=0_99,
```

```
u=0_01,loops=10to5,testing_0,divx=0_05,divy=0_02';
MSE_train(7)=MSE(end);
MSE_test(7)=MSE_RPtest8;


Nw=[3,7,15,31,63,127,255];
figure
loglog(Nw,MSE_train,'-*',Nw,MSE_test,'-o');
legend('MSE_t_r_a_i_n','MSE_t_e_s_t');
xlabel('No. of Wavelons (Nw)')
ylabel('MSE')

i_trn_min=find(MSE_train==min(MSE_train));
Nw_min_MSE_train=Nw(i_trn_min)

i_tst_min=find(MSE_test==min(MSE_test));
Nw_min_MSE_test=Nw(i_tst_min)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Finding the MSE value after training simulated data                    %%%
%%% with gaps and testing for experimental data set at x=0 cm.%%%
%%% filename = 'wavenet52_reverse_testing.m'                                %%%
%%%                                                                         %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc

load 'RP,simulated data gaps,w=gaussian,Nl=6,Nw=63,gamma=0_99,u=0_01,
loops=10to5,testing_0,divx=0_05,divy=0_02'

load 'inoutexp'
x_test=[inoutexp(:,3),inoutexp(:,4),inoutexp(:,5),inoutexp(:,6)];
Y_test=[inoutexp(:,1),inoutexp(:,2)];

nz2=find(Y_test(:,1)==-0.55);
for ni2=1:length(nz2)
    x_2(ni2,:)=x_test(nz2(ni2),:);
    Y_2(ni2,:)=Y_test(nz2(ni2),:);
end
nz22=find((Y_2(:,2)<0.4) & (Y_2(:,2)>-0.4));
for ni22=1:length(nz22)
    x_22(ni22,:)=x_2(nz22(ni22),:);
    Y_22(ni22,:)=Y_2(nz22(ni22),:);
end

nz3=find(Y_test(:,1)==0);
for ni3=1:length(nz3)
    x_3(ni3,:)=x_test(nz3(ni3),:);
    Y_3(ni3,:)=Y_test(nz3(ni3),:);
end
nz32=find((Y_3(:,2)<0.4) & (Y_3(:,2)>-0.4));
for ni32=1:length(nz32)
    x_32(ni32,:)=x_3(nz32(ni32),:);
    Y_32(ni32,:)=Y_3(nz32(ni32),:);
end

nz4=find(Y_test(:,1)==0.55);
for ni4=1:length(nz4)
    x_4(ni4,:)=x_test(nz4(ni4),:);
    Y_4(ni4,:)=Y_test(nz4(ni4),:);
end
nz42=find((Y_4(:,2)<0.4) & (Y_4(:,2)>-0.4));
for ni42=1:length(nz42)
    x_42(ni42,:)=x_4(nz42(ni42),:);
    Y_42(ni42,:)=Y_4(nz42(ni42),:);
end
x_test=[x_22];
Y_test=[Y_22]';

N=size(x_test);
Np=N(1,1);
I=ones(1,Np);
input_tst=[I; x_test'];
```

```
z=zeros(Nw,Ni,Np);

phi=zeros(Nw,Ni,Np);
phi_p=zeros(Nw,Ni,Np);

PHI_p=zeros(Nw,Ni,Np);

Y_hat_tst=zeros(No,Np);
E=zeros(No,Np);


PHI=ones(1,Nw,Np);
for j=1:Nw
    for i=1:Ni
        z(j,i,:)=(x_test(:,i)-m(j,i))./d(j,i);
        phi(j,i,:)=(-z(j,i,:)).*exp(-0.5*(z(j,i,:).^2));
        phi_p(j,i,:)=(((z(j,i,:)).^2)-1).*exp(-0.5*(z(j,i,:).^2));
        PHI(1,j,:)=PHI(1,j,:).*phi(j,i,:);
    end
end
PHI=squeeze(PHI);
Y_hat_tst=(Woh*PHI) + (Woi*input_tst);
E=Y_test-Y_hat_tst;

SSE=sum(sum(E.*E));

MSE=SSE/Np;

t=0:length(Y_test(1,:))-1;

figure(3)
subplot(1,2,1)
handlevector(1)=plot(t,Y_test(1,f(1):f(end)),
'LineWidth',2,'Color',[0 0 0],'DisplayName','Experimental data')
hold on
handlevector(2)=plot(t,Y_hat_tst(1,f(1):f(end)),
'Marker','o','LineStyle','none','Color',[0 0 0],
'DisplayName','WN test output')
axis([0 30 -0.6 0.6])
xlabel('t (s)')
ylabel('x (cm)')
legend(handlevector([1 2]))

subplot(1,2,2)
handlevector(1)=plot(t,Y_test(2,f(1):f(end)),
'LineWidth',2,'Color',[0 0 0],'DisplayName','Experimental data')
hold on
handlevector(2)=plot(t,Y_hat_tst(2,f(1):f(end)),
'Marker','square','LineStyle','none','Color',[0 0 0],
'DisplayName','WN test output')
axis([0 30 -0.6 0.6])
xlabel('t (s)')
ylabel('y (cm)')
legend(handlevector([1 2]))
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Saving the input/output simulated data with gaps in the   %%%
%%% file 'inout.txt'.                                          %%%
%%% filename = 'powerinout.m'                                  %%%
%%%                                                            %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all

xc=-2:0.01:2;          %x-coordinate of beam spot centre
yc=-2:0.01:2;          %y-coordiate of beam spot centre

l_x=length(xc);
l_y=length(yc);
olddiv=0.01;
newdivx=0.05;   %Specify resolution for the x position
newdivy=0.02;   %Specify resolution for the y position
dvx=newdivx/olddiv;
dvy=newdivy/olddiv;

if rem(l_x,dvx)~=0
nx=((l_x-1)/dvx)+1;
else
nx=l_x/dvx;
end

if rem(l_y,dvy)~=0
ny=((l_y-1)/dvy)+1;
else
ny=l_y/dvy;
end

load 'PS1'
load 'PS2'
load 'PS3'
load 'PS4'
fid = fopen('inout.txt', 'wt');
for s=1:nx
 for t=1:ny
  i=dvx*(s-1)+1;
  j=dvy*(t-1)+1;
  inout=[xc(i);yc(j);PS1(i,j);PS2(i,j);PS3(i,j);PS4(i,j)];
  fprintf(fid, '%6.3f %6.3f %12.6f %12.6f %12.6f %12.6f\n', inout);
 end
end

fclose(fid)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Saving the input/output simulated data with no gaps in the%%%
%%% file 'inoutzero.txt.txt'.                                              %%%
%%% filename = 'powerinoutzero.m'                                          %%%
%%%                                                                        %%%
%%% Written by: Yasmine El-Ashi, Fall 2009                                 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all

xc=-2:0.01:2;           %x-coordinate of beam spot centre
yc=-2:0.01:2;           %y-coordinate of beam spot centre

l_x=length(xc);
l_y=length(yc);
olddiv=0.01;
newdivx=0.05;   %Specify resolution for the x position
newdivy=0.05;   %Specify resolution for the y position
dvx=newdivx/olddiv;
dvy=newdivy/olddiv;

if rem(l_x,dvx)~=0
nx=((l_x-1)/dvy)+1;
else
nx=l_x/dvx;
end

if rem(l_y,dvy)~=0
ny=((l_y-1)/dvy)+1;
else
ny=l_y/dvy;
end

load 'PS1zero'
load 'PS2zero'
load 'PS3zero'
load 'PS4zero'
fid = fopen('inoutzero.txt', 'wt');
for s=1:nx
 for t=1:ny
  i=dvx*(s-1)+1;
  j=dvy*(t-1)+1;
  inoutzero=[xc(i);yc(j);PS1zero(i,j);PS2zero(i,j);
  PS3zero(i,j);PS4zero(i,j)];
  fprintf(fid, '%6.3f %6.3f %12.6f %12.6f %12.6f %12.6f\n', inoutzero);
 end
end

fclose(fid)
```

Appenndix B: User Manual for Wavelet Network Code

This is a user manual to explain the command lines for the wavelet network training code for multiple input, multiple output (MIMO) function approximation. The code can be divided into three main sections, namely; WN initialization, feedforward algorithm and backpropagation algorithm. Before running the code, the input/output data should be saved in the same directory as a .txt or .MAT file. The data should be stored as a 2 dimensional matrix, where the number of rows indicate the number of patterns or observations available, and each column represents either an input or output variable.

.1 WN Initialization

First, load the input/output data file:

```
load 'inoutzero.txt'
%Input data
x=[inoutzero(:,3),inoutzero(:,4),inoutzero(:,5),inoutzero(:,6)];
%Desired output data
Y=[inoutzero(:,1),inoutzero(:,2)];
Y=Y';
```

Let us define the column vector:

$$X_i = \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^p \\ \vdots \\ x_i^{N_p} \end{bmatrix}$$

and the row vector:

$$Y_k = \begin{bmatrix} y_k^1 & y_k^2 & \cdots & y_k^p & \cdots & y_k^{N_p} \end{bmatrix}$$

such that the input matrix,

$$\mathrm{X} = \begin{bmatrix} X_1 & X_2 & \cdots & X_i & \cdots & X_{N_i} \end{bmatrix}$$

and the output matrix,

$$\mathrm{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \\ \vdots \\ Y_{N_o} \end{bmatrix}$$

In this case, the input/output file `'inoutzero.txt'` contains 4 input variables allocated in columns 3, 4, 5, and 6, as well as 2 output variables in columns 1 and 2. Thus, on the one hand, the matrix `x` = X consists of the input data where the number of rows equal the number of patterns N_p and the number of columns equal the number of input variables N_i. On the other hand, the matrix `Y` = Y consists of the desired output data that will be used for WN training where the number of columns equal N_p and the number of rows equal N_o, the total number of output variables.
Next, the following parameters are defined:

```
u=0.01;          %Setting the learning rate
gamma=1-u;       %Setting the momentum coefficient
Ax=size(x);
Ay=size(Y);
Np=Ax(1,1);      %no. of patterns
Ni=Ax(1,2);      %no. of input nodes
Nl=3;            %no. of levels
Nw=(2^Nl)-1;     %no. of hidden nodes
No=Ay(1,1);      %no. of output nodes
```

You only need to manually set the value for the learning rate `u` and the number of levels `Nl`.

.1.1 Initializing $\mathrm{W_{oh}}$ and $\mathrm{W_{oi}}$

The matrix $\mathrm{W_{oh}}$ which consists of the weights c_{kj} as shown in equation (5.3) is first initialized to zeros as follows:

```
Woh=zeros(No,Nw);
```

The matrix $\mathrm{W_{oi}}$ which consists of the direct linear coefficients between the input and output layers as shown in equation (5.2) is initialized using the least

squares method. First, let us recall the feedforward equation for the kth output of the WN for a certain pattern p, given by the following equation:

$$\hat{y}_k^p = \sum_{j=1}^{N_w} c_{kj}\Phi_j + \sum_{i=1}^{N_i} a_{ki}x_i^p + b_k. \tag{1}$$

By only considering the linear part for the feedfoward equation, the WN output in equation (1) can be reduced to the following:

$$\tilde{y}_k^p = b_k + \sum_{i=1}^{N_i} a_{ki}x_i^p. \tag{2}$$

Equation (2) can also be written as:

$$\tilde{y}_k(p) = b_k + \sum_{i=1}^{N_i} a_{ki}x_i(p). \tag{3}$$

By assuming the direct connections only $\tilde{y}_k(p)$ can be computed as follows:

$$\tilde{y}_k(p) = b_k + a_{k1}x_1(p) + a_{k2}x_2(p) + \ldots + a_{kN_i}x_{N_i}(p). \tag{4}$$

Next, let us define the error $\varepsilon_k(p)$ as the difference between the actual desired output $y_k(p)$ and the estimated output $\tilde{y}_k(p)$:

$$\varepsilon_k(p) = y_k(p) - \tilde{y}_k(p) = y_k(p) - b_k - a_{k1}x_1(p) - a_{k2}x_2(p) - \ldots - a_{kN_i}x_{N_i}(p). \tag{5}$$

Equation (5) can be rewritten as follows:

$$y_k(p) = b_k + a_{k1}x_1(p) + a_{k2}x_2(p) + \ldots + a_{kN_i}x_{N_i}(p) + \varepsilon_k(p). \tag{6}$$

Let us assume that the input and the actual output are measured for $1 \leqslant p \leqslant N_p$. By substituting $p = n, n+1, \ldots, N_p$ into equation (6) and combining the resulting equations into the vector matrix equation, we obtain:

$$\begin{bmatrix} y_k(n) \\ y_k(n+1) \\ . \\ . \\ . \\ y_k(N_p) \end{bmatrix} = \begin{bmatrix} 1 & x_1(n) & . & . & . & x_{N_i}(n) \\ 1 & x_1(n+1) & . & . & . & x_{N_i}(n+1) \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ 1 & x_1(N_p) & . & . & . & x_{N_i}(N_p) \end{bmatrix} \begin{bmatrix} b_k \\ a_{k1} \\ . \\ . \\ . \\ a_{kN_i} \end{bmatrix} + \begin{bmatrix} \varepsilon_k(n) \\ \varepsilon_k(n+1) \\ . \\ . \\ . \\ \varepsilon_k(N_p) \end{bmatrix}.$$

In addition, let us define:

$$Y_k(N_p) = \begin{bmatrix} y_k(n) \\ y_k(n+1) \\ . \\ . \\ . \\ y_k(N_p) \end{bmatrix}, E_k(N_p) = \begin{bmatrix} \varepsilon_k(n) \\ \varepsilon_k(n+1) \\ . \\ . \\ . \\ \varepsilon_k(N_p) \end{bmatrix}, U_k(N_p) = \begin{bmatrix} b_k \\ a_{k1} \\ . \\ . \\ . \\ a_{kN_i} \end{bmatrix},$$

and,

$$C(N_p) = \begin{bmatrix} 1 & x_1(n) & . & . & . & x_{N_i}(p) \\ 1 & x_1(n+1) & . & . & . & x_{N_i}(p+1) \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ 1 & x_1(N_p) & . & . & . & x_{N_i}(N_p) \end{bmatrix}.$$

Therefore, the vector matrix equation can be written as follows:

$$Y_k(N_p) = C(N_p) U_k(N_p) + E_k(N_p). \tag{7}$$

Let us, define the performance index as:

$$J_{N_p} = \frac{1}{2}\sum_{p=n}^{N_p} \varepsilon_k^2(p) = \frac{1}{2} E_k^{\mathrm{T}}(N_p) E_k(N_p). \tag{8}$$

Thus, our problem becomes that of determining $U_k(N_p)$ such that the parameter values, $b_k, a_{k1}, a_{k2}, \ldots, a_{kN_i}$ will best fit the observed data. Let,

$$\begin{aligned} J_{N_p} &= \frac{1}{2} E_k^{\mathrm{T}}(N_p) E_k(N_p) \\ &= \frac{1}{2}[Y_k(N_p) - C(N_p) U_k(N_p)]^{\mathrm{T}} [Y_k(N_p) - C(N_p) U_k(N_p)] \\ &= \frac{1}{2}\left[Y_k^{\mathrm{T}}(N_p) - U_k^{\mathrm{T}}(N_p) C^{\mathrm{T}}(N_p)\right] [Y_k(N_p) - C(N_p) U_k(N_p)] \\ &= \frac{1}{2}\left\{-U_k^{\mathrm{T}}(N_p) C^{\mathrm{T}}(N_p) Y_k(N_p) + U_k^{\mathrm{T}}(N_p) C^{\mathrm{T}}(N_p) C(N_p) U_k(N_p)\right\} \\ &+ \frac{1}{2}\left\{Y_k^{\mathrm{T}}(N_p) Y_k(N_p) - Y_k^{\mathrm{T}}(N_p) C(N_p) U_k(N_p)\right\} \end{aligned} \tag{9}$$

To minimize J_{N_p} with respect to $U_k(N_p)$, we set:

$$\frac{\partial J_{N_p}}{\partial U_k(N_p)} = C^{\mathrm{T}}(N_p) C(N_p) U_k(N_p) - C^{\mathrm{T}}(N_p) Y_k(N_p) = 0. \tag{10}$$

Let us denote $U_k(N_p)$ that satisfies equation (10) as $\tilde{U}_k(N_p)$. Then, we have:

$$C^{\mathrm{T}}(N_p) C(N_p) \tilde{U}_k(N_p) = C^{\mathrm{T}}(N_p) Y_k(N_p). \tag{11}$$

In this analysis, we assume that $C^{\mathrm{T}}(N_p) C(N_p)$ is nonsingular, therefore the inverse of $C^{\mathrm{T}}(N_p) C(N_p)$ exists. Hence, solving equation (11) for $\tilde{U}_k(N_p)$, we obtain:

$$\tilde{U}_k(N_p) = \left[C^{\mathrm{T}}(N_p) C(N_p)\right]^{-1} C^{\mathrm{T}}(N_p) Y_k(N_p). \tag{12}$$

Therefore, the following `for` loop is used to initialize the matrix $\mathrm{W_{oi}}$:

```
I=ones(1,Np);
input=[I; x'];  %input matrix
C=input';
```

```
for k=1:No
    Woi1=(inv(C'*C))*C'*Y(k,:)';
    Woi1=Woi1';
    Woi(k,:)=Woi1;
end
```

where, `C` $= C(N_p)$, `Y(k,:)'` $= Y_k(N_p)$, and `Woi1` $= \tilde{U}_k(N_p)$.

.1.2 Dyadic Initialization

In this case, the translation and dialation parameters will be initialized using the dyadic grid method. On the one hand, the translation matrix, `m` = m, having m_{ji} as its elements, has `Nw` rows and `Ni` columns. On the other hand, the dilation matrix, `d` = d, having d_{ji} as its elements, is also composed of `Nw` rows, and `Ni` columns.

To be able to explain the steps used in the dyadic initialization algorithm, we will use the dyadic grid shown in Figure 5.3 as an example. As illustrated in Figure 5.3 we have 3 levels, that is `Nl` is set to 3, hence `Nw` the number of wavelons is 7. In addition, given such an example, each input variable is assumed to have a range of $[-1, +1]$. Therefore, the following parameters `ak` and `bk` can be defined in our code as:

```
ak=min(x);
bk=max(x);
```

where `ak=[-1 -1 -1 -1]` and `bk=[1 1 1 1]`, provided that we have four input variables, that is `Ni` = 4. The block of code used for the dyadic initialization is given as follows:

```
for i=1:Ni
    for L=1:Nl
        div=(bk(i)-ak(i))/(2^L);
        n=1;
        p=0;
        while p≠bk(i)
            p=ak(1,i)+(n*div);
```

```
            M(n)=p;
            D(n)=div;
            n=n+1;
        end

        for w=1:(n/2)
            f(1,w,L)=M(2*w-1);
            g(1,w,L)=D(2*w-1);
        end
    end
    F=squeeze(f);
    G=squeeze(g);
    [pf,qf]=find(F≠0);
    [pg,qg]=find(G≠0);

    for w=1:length(pf)
        s(w)=F(pf(w),qf(w));
    end
    if F(1,1)==0
        m(:,i)=[F(1,1),s];
    elseif F(1,1)≠0
        m(:,i)=s;
    end

    for w=1:length(pg)
        t(w)=G(pg(w),qg(w));
    end
    if G(1,1)==0
        d(:,i)=[G(1,1),t];
    elseif G(1,1)≠0
        d(:,i)=t;
    end
end
```

The main `for` loop is repeated `Ni` times, such that at every new entry of the value `i`, the column vectors `m(:,i)` $= m_i$ and `d(:,i)` $= d_i$ are initialized,

where:

$$m_i = \begin{bmatrix} m_{1i} \\ m_{2i} \\ . \\ . \\ . \\ m_{N_w i} \end{bmatrix}, d_i = \begin{bmatrix} d_{1i} \\ d_{2i} \\ . \\ . \\ . \\ d_{N_w i} \end{bmatrix}.$$

First, let us set `i` $= 1$, and `L` $= 1$. In addition, let `div=(bk(i)-ak(i))/(2^L)` $= \Delta x$ represent the differential distance between each interval, if the translation scale which ranges between `ak(i)` $= -1$ and `bk(i)` $= 1$, is to be divided dyadically, that is in powers of 2.

For the first level at `L` $= 1$, $\Delta x = 1$. The parameters `n=1` and `p=0` are then initialized to be used in the following `while` loop:

```
while p≠bk(i)
    p=ak(1,i)+(n*div);
    M(n)=p;
    D(n)=div;
    n=n+1;
end
```

`p` should be initialized to a value less than `bk`. We exit the `while` loop once `p` $=$ `bk(i)`. The output of the above `while` loop at `L` $= 1$, can be stated as follows:

```
>> M

M =

     0     1

>> D
```

```
D =

     1     1

>> n

n =

     3
```

The output of the `while` loop at `L` = 2, can be stated as follows:

```
>> M

M =

   -0.5000         0    0.5000    1.0000

>> D

D =

    0.5000    0.5000    0.5000    0.5000

>> n

n =

     5
```

The output of the `while` loop at `L` = 3, can be stated as follows:

```
>> M

M =

   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
1.0000
```

```
>> D

D =

    0.2500    0.2500    0.2500    0.2500    0.2500    0.2500    0.2500
0.2500

>> n

n =

     9
```

It can be realized that as `L` increments, the size of the row vectors `M` and `D` increases by 2^L. The difference between the enteries of `M` is Δx which becomes smaller as `L` increases. The elements of `D` are all equal to Δx which depends on the value of `L`.

Next, we enter into the following `for` loop:

```
for w=1:(n/2)
    f(1,w,L)=M(2*w-1);
    g(1,w,L)=D(2*w-1);
end
```

The above loop generates two 3D arrays `f` and `g`. In general, the third dimension is numbered by *pages*. Thus, a 3D array has rows, columns, and pages. Each page contains a 2D array of rows and columns. The `for` loop will exterminate once `w=n/2`. However, if `n` is an odd number, as the case beforehand, the `for` loop will implicitly stop at `w=floor(n/2)`. Each page for both `f` and `g` indicate the level `L`. While, at every page we have one row and `w` columns. The columns of `f` at level `L` are equivalent to the odd numbered indices of row vector `M`. Similarly, the columns of `g` at level `L` are equivalent to the odd numbered indicies of row vector `D`. The end result of `f` and `g` for `L=1:3`, is as follows:

```
>> f

f(:,:,1) =

     0     0     0     0


f(:,:,2) =

   -0.5000    0.5000         0         0


f(:,:,3) =

   -0.7500   -0.2500    0.2500    0.7500

>> g

g(:,:,1) =

     1     0     0     0


g(:,:,2) =

    0.5000    0.5000         0         0


g(:,:,3) =

    0.2500    0.2500    0.2500    0.2500
```

A point to note, is that just as all of the columns of a 2D array must have the same number of rows and vice versa, all of the pages of a 3D array must have the same number of rows and columns. Thus for the first level, or first page, it is easy to observe that 3 additional zeros have been appended to the original vector, and for the second level or second page we have two additional zeros, for both `f` and `g`.

Next, the matrices F and G are defined as follows:

```
F=squeeze(f);
G=squeeze(g);
```

If a 3D array is composed of a single row vector at every page, the squeeze function will transform the row vectors of such a 3D array into column vectors of a 2 dimensional matrix. Thus, the pages of the 3D array become the columns of a 2D matrix, and the columns of the 3D array become the rows of the 2D matrix. Therefore, for this case we have:

```
>> F

F =

         0   −0.5000   −0.7500
         0    0.5000   −0.2500
         0         0    0.2500
         0         0    0.7500

>> G

G =

    1.0000    0.5000    0.2500
         0    0.5000    0.2500
         0         0    0.2500
         0         0    0.2500
```

Let us further define, the following parameters:

```
[pf,qf]=find(F≠0);
[pg,qg]=find(G≠0);
```

where pf and qf are the row and column indices where an element of F is not equal to zero. Moreover, pg and qg are the row and column indices where an element of G is not equal to zero. For this example we have:

```
 >> [pf,qf]

ans =

     1     2
     2     2
     1     3
     2     3
     3     3
     4     3

>> [pg,qg]

ans =

     1     1
     1     2
     2     2
     1     3
     2     3
     3     3
     4     3
```

Next, we need to extract the initial translation and dilation parameters from the matrices `F` and `G` such that:

$$m_1 = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \\ m_{41} \\ m_{51} \\ m_{61} \\ m_{71} \end{bmatrix} = \begin{bmatrix} \mathrm{F}(1,1) \\ \mathrm{F}(1,2) \\ \mathrm{F}(2,2) \\ \mathrm{F}(1,3) \\ \mathrm{F}(2,3) \\ \mathrm{F}(3,3) \\ \mathrm{F}(4,3) \end{bmatrix},$$

and,

$$d_1 = \begin{bmatrix} d_{11} \\ d_{21} \\ d_{31} \\ d_{41} \\ d_{51} \\ d_{61} \\ d_{71} \end{bmatrix} = \begin{bmatrix} \mathrm{G}(1,1) \\ \mathrm{G}(1,2) \\ \mathrm{G}(2,2) \\ \mathrm{G}(1,3) \\ \mathrm{G}(2,3) \\ \mathrm{G}(3,3) \\ \mathrm{G}(4,3) \end{bmatrix}.$$

To do that we first run the following `for` loops, to generate the row vectors `s` and `t` which consist of the nonzero elements of `F` and `G`:

```
for w=1:length(pf)
    s(w)=F(pf(w),qf(w));
end

for w=1:length(pg)
    t(w)=G(pg(w),qg(w));
end
```

Since, $m_{11} = \mathrm{F}(1,1)$ is zero in this example, the following `if` statements need to be applied:

```
if F(1,1)==0
    m(:,i)=[F(1,1),s];
elseif F(1,1)≠0
    m(:,i)=s;
end
```

Using such a dyadic grid the dilation parameters should not be zero, however just for consistency we apply the same `if` statements as the code above:

```
if G(1,1)==0
    d(:,i)=[G(1,1),t];
elseif G(1,1)≠0
```

```
    d(:,i)=t;
end
```

The `i` of the main `for` loop is then incremented and the preceding procedure is repeated until `i=Ni`. The end result of the initial values for the matrices `m` = m and `d` = d is as follows:

```
>> m

m =

         0         0         0         0
   -0.5000   -0.5000   -0.5000   -0.5000
    0.5000    0.5000    0.5000    0.5000
   -0.7500   -0.7500   -0.7500   -0.7500
   -0.2500   -0.2500   -0.2500   -0.2500
    0.2500    0.2500    0.2500    0.2500
    0.7500    0.7500    0.7500    0.7500

>> d

d =

    1.0000    1.0000    1.0000    1.0000
    0.5000    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000    0.5000
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
```

where,

$$\mathrm{m} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1i} & \cdots & m_{1N_i} \\ m_{21} & m_{22} & \cdots & m_{2i} & \cdots & m_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ m_{j1} & m_{j2} & \cdots & m_{ji} & \cdots & m_{jN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ m_{N_w1} & m_{N_w2} & \cdots & m_{N_wi} & \cdots & m_{N_wN_i} \end{bmatrix} \tag{13}$$

and,

$$\mathrm{d} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1i} & \cdots & d_{1N_i} \\ d_{21} & d_{22} & \cdots & d_{2i} & \cdots & d_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{j1} & d_{j2} & \cdots & d_{ji} & \cdots & d_{jN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{N_w1} & d_{N_w2} & \cdots & d_{N_wi} & \cdots & d_{N_wN_i} \end{bmatrix} \tag{14}$$

The algorithm described for dyadic initialization of the translation and dialation parameters is not necessarily general. However to ensure that it works for different ranges, it is recommended to have equal absolute values for `ak` and `bk`, if the range extends to the negative scale. In addition, it is also recommended to have `ak` and `bk` as integers or rational numbers rather than irrational values. Further adjustments can be made to the above dyadic initialization code, to account for irrational cases.

.2 Feedforward Algorithm

The feedforward matrix equation for the wavelet network for a pattern p can be stated as follows:

$$\hat{Y}^p = \begin{bmatrix} \hat{y}_1^p \\ \hat{y}_2^p \\ \vdots \\ \hat{y}_k^p \\ \vdots \\ \hat{y}_{N_o}^p \end{bmatrix} = (\mathrm{W_{oh}} * \Phi^p) + (\mathrm{W_{oi}} * I^p)\,, \tag{15}$$

where the vectors:

$$I^p = \begin{bmatrix} 1 & x_1^p & \cdots & x_i^p & \cdots & x_{N_i}^p \end{bmatrix}^{\mathrm{T}},$$

and,

$$\Phi^p = \begin{bmatrix} \Phi_1^p & \Phi_2^p & \cdots & \Phi_j^p & \cdots & \Phi_{N_w}^p \end{bmatrix}^{\mathrm{T}}.$$

The symbol '*' in equation (15) represents a matrix multiplication. Since, batch processing is used in training the wavelet network for a total of N_p patterns, equation (15) can be rewritten in the following form:

$$\hat{\mathrm{Y}} = \left[\hat{Y}^1 \cdots \hat{Y}^p \cdots \hat{Y}^{N_p}\right] = (\mathrm{W_{oh}} * \Phi) + (\mathrm{W_{oi}} * \mathrm{I})\,, \tag{16}$$

where the matrices:

$$\mathrm{I} = \begin{bmatrix} I^1 & I^2 & \cdots & I^p & \cdots & I^{N_p} \end{bmatrix}.$$

and,

$$\Phi = \begin{bmatrix} \Phi^1 & \Phi^2 & \cdots & \Phi^p & \cdots & \Phi^{N_p} \end{bmatrix},$$

In order to compute the $N_o \times N_p$ matrix `Y_hat` $= \hat{\mathrm{Y}}$, let us first define the 3D array `PHI` $= \mathbf{\Phi}$ with N_p pages. Each page of $\mathbf{\Phi}$ consists of the row vector Φ^p. The elements of the 3D array `PHI` are initialized to ones:

```
PHI=ones(1,Nw,Np);
```

to be used in the following `for` loops for computing the parameters Φ_j^p:

```
for j=1:Nw
    for i=1:Ni
        z(j,i,:)=(x(:,i)-m(j,i))./d(j,i);
        phi(j,i,:)=(-z(j,i,:)).*exp(-0.5*(z(j,i,:).^2));
        phi_p(j,i,:)=(((z(j,i,:)).^2)-1).*exp(-0.5*(z(j,i,:).^2));
        PHI(1,j,:)=PHI(1,j,:).*phi(j,i,:);
    end
end
```

In this case, `z` $= \mathbf{Z}$ is a 3D array with N_p pages, where each page constitutes of the matrix:

$$\mathrm{Z}^p = \begin{bmatrix} z_{11}^p & z_{12}^p & \cdots & z_{1i}^p & \cdots & z_{1N_i}^p \\ z_{21}^p & z_{22}^p & \cdots & z_{2i}^p & \cdots & z_{2N_i}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{j1}^p & z_{j2}^p & \cdots & z_{ji}^p & \cdots & z_{jN_i}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{N_w1}^p & z_{N_w2}^p & \cdots & z_{N_wi}^p & \cdots & z_{N_wN_i}^p \end{bmatrix}.$$

Each element z_{ji}^p is represented by the following equation:

$$z_{ji}^p = \frac{(x_i^p - m_{ji})}{d_{ji}}. \tag{17}$$

Therefore, in order to batchly compute the elements of the 3D array $\mathbf{Z}$, we need to introduce a term known as *page vector*, which is a vector that extends across all the pages of a 3D array at a certain row and column. Consequently,

the page vector `z(j,i,:)` $= Z_{ji}$ can be stated as follows:

$$Z_{ji} = \begin{bmatrix} z_{ji}^{1} \\ z_{ji}^{2} \\ \vdots \\ z_{ji}^{p} \\ \vdots \\ z_{ji}^{N_p} \end{bmatrix} = \frac{1}{d_{ji}} . \left(X_i - m_{ji} \right) . \tag{18}$$

The symbol '.' in equation (18) represents an element by element multiplication. The line of code used to replicate equation (18) is as follows:

```
z(j,i,:)=(x(:,i)-m(j,i))./d(j,i);
```

Next, let us define `phi` $= \varphi$ a 3D array with N_p pages, where each page constitutes of the matrix:

$$\varphi^{p} = \begin{bmatrix} \varphi_{11}^{p} & \varphi_{12}^{p} & \cdots & \varphi_{1i}^{p} & \cdots & \varphi_{1N_i}^{p} \\ \varphi_{21}^{p} & \varphi_{22}^{p} & \cdots & \varphi_{2i}^{p} & \cdots & \varphi_{2N_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \varphi_{j1}^{p} & \varphi_{j2}^{p} & \cdots & \varphi_{ji}^{p} & \cdots & \varphi_{jN_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \varphi_{N_w 1}^{p} & \varphi_{N_w 2}^{p} & \cdots & \varphi_{N_w i}^{p} & \cdots & \varphi_{N_w N_i}^{p} \end{bmatrix} .$$

Each element φ_{ji}^{p} is represented by the following equation:

$$\varphi_{ji}^{p} = \psi \left(z_{ji}^{p} \right) = -z_{ji}^{p} \exp \left(-\frac{z_{ji}^{p} z_{ji}^{p}}{2} \right) . \tag{19}$$

In order, to batchly compute the elements of the 3D array φ, we will define the following page vector:

$$\varphi_{ji} = \begin{bmatrix} \varphi_{ji}^{1} \\ \varphi_{ji}^{2} \\ \vdots \\ \varphi_{ji}^{p} \\ \vdots \\ \varphi_{ji}^{N_p} \end{bmatrix} = -Z_{ji}.\exp\left(-\frac{Z_{ji}.Z_{ji}}{2}\right). \tag{20}$$

The line of code used to compute `phi(j,i,:)` $= \varphi_{ji}$ is as follows:

```
phi(j,i,:)=(-z(j,i,:)).*exp(-0.5*(z(j,i,:).^2));
```

Similarly, we will define `phi_p` $= \dot{\varphi}$ a 3D array with N_p pages, where each page constitutes of the matrix:

$$\dot{\varphi}^{p} = \begin{bmatrix} \dot{\varphi}_{11}^{p} & \dot{\varphi}_{12}^{p} & \cdots & \dot{\varphi}_{1i}^{p} & \cdots & \dot{\varphi}_{1N_i}^{p} \\ \dot{\varphi}_{21}^{p} & \dot{\varphi}_{22}^{p} & \cdots & \dot{\varphi}_{2i}^{p} & \cdots & \dot{\varphi}_{2N_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \dot{\varphi}_{j1}^{p} & \dot{\varphi}_{j2}^{p} & \cdots & \dot{\varphi}_{ji}^{p} & \cdots & \dot{\varphi}_{jN_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \dot{\varphi}_{N_w1}^{p} & \dot{\varphi}_{N_w2}^{p} & \cdots & \dot{\varphi}_{N_wi}^{p} & \cdots & \dot{\varphi}_{N_wN_i}^{p} \end{bmatrix}.$$

Each element $\dot{\varphi}_{ji}^{p}$ is represented by the following equation:

$$\dot{\varphi}_{ji}^{p} = \frac{\partial \varphi_{ji}^{p}}{\partial z_{ji}^{p}} = \left(z_{ji}^{p} z_{ji}^{p} - 1\right) \exp\left(-\frac{z_{ji}^{p} z_{ji}^{p}}{2}\right). \tag{21}$$

To batchly compute the elements of the 3D array $\dot{\varphi}$, the following page vector will be defined:

$$\dot{\varphi}_{ji} = \begin{bmatrix} \dot{\varphi}_{ji}^{1} \\ \dot{\varphi}_{ji}^{2} \\ \vdots \\ \dot{\varphi}_{ji}^{p} \\ \vdots \\ \dot{\varphi}_{ji}^{N_p} \end{bmatrix} = (Z_{ji}.Z_{ji} - 1) \,.\, \exp\left(-\frac{Z_{ji}.Z_{ji}}{2}\right). \tag{22}$$

The line of code used to compute `phi_p` $= \dot{\varphi}_{ji}$ is as follows:

```
phi_p(j,i,:)=(((z(j,i,:)).^2)-1).*exp(-0.5*(z(j,i,:).^2));
```

Next, let us define the page vector Φ_j which can be evaluated as follows:

$$\Phi_j = \begin{bmatrix} \Phi_j^{1} \\ \Phi_j^{2} \\ \vdots \\ \Phi_j^{p} \\ \vdots \\ \Phi_j^{N_p} \end{bmatrix} = \prod_{n=1}^{Ni} \varphi_{jn} = \varphi_{j1}.\varphi_{j2}.\ldots.\varphi_{ji}.\ldots.\varphi_{jN_i}. \tag{23}$$

Equation (23) can be computed using the following code:

```
PHI(1,j,:)=PHI(1,j,:).*phi(j,i,:);
```

where, `PHI(1,j,:)` $= \Phi_j$. After constructing the 3D array $\mathbf{\Phi}$, the command `PHI=squeeze(PHI);` is used to transform the 3D array $\mathbf{\Phi}$ into the 2 dimensional matrix Φ, that will be used to compute `Y_hat` $= \hat{\mathrm{Y}}$ as follows:

```
Y_hat=(Woh*PHI) + (Woi*input);
```

where, `input` $=$ I. The error $\mathrm{E} = \mathrm{Y} - \hat{\mathrm{Y}}$ between the wavelet network output $\hat{\mathrm{Y}}$ and the desired output Y is evaluated as follows:

```
E=Y-Y_hat;
```

where,

$$\mathrm{E} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_k \\ \vdots \\ E_{N_o} \end{bmatrix}$$

and the row vector,

$$\begin{aligned} E_k &= \begin{bmatrix} e_k^1 & e_k^2 & \cdots & e_k^p & \cdots & e_k^{N_p} \end{bmatrix} \\ &= \begin{bmatrix} y_k^1 - \hat{y}_k^1 & y_k^2 - \hat{y}_k^2 & \cdots & y_k^p - \hat{y}_k^p & \cdots & y_k^{N_p} - \hat{y}_k^{N_p} \end{bmatrix} \end{aligned}$$

Then we compute the sum of square error (SSE) and the mean square error (MSE) for every iteration as stated below:

```
%Computing the sum of square error (SSE)
SSE=sum(sum(E.*E));

%Computing the mean square error (MSE) for every iteration
MSE(loops)=SSE/Np;
```

.3 Backpropagation Algorithm

In this section we will be using equations (3.12) to (3.19) to be able to compute the parameters of $\Delta\theta\,(l)$ for updating the components of the vector θ. The

vector $\Delta\theta(l)$ can be defined as follows:

$$\Delta\theta(l) = \begin{bmatrix} \Delta b_k(l) \\ \Delta a_{ki}(l) \\ \Delta c_{kj}(l) \\ \Delta m_{ji}(l) \\ \Delta d_{ji}(l) \end{bmatrix} = -\frac{\mu}{N_p}\frac{\partial J}{\partial \theta} + \gamma\Delta\theta(l-1). \tag{24}$$

The network parameter vector θ consists of the components of the matrices $\mathrm{W_{oi}}$, $\mathrm{W_{oh}}$, m and d. Thus, in order to update θ every epoch using $\theta_{\text{new}} = \theta_{\text{old}} + \Delta\theta(l)$, we need to update the components of $\mathrm{W_{oi}}$, $\mathrm{W_{oh}}$, m and d.

.3.1 Updating the parameters of $\mathrm{W_{oi}}$

Let us first, state the following equations for $\Delta b_k(l)$:

$$\begin{aligned} \Delta b_k(l) &= -\frac{\mu}{N_p}\frac{\partial J}{\partial b_k} + \gamma\Delta b_k(l-1) \\ &= -\frac{\mu}{N_p}\left(-\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} e_n^p \frac{\partial \hat{y}_n^p}{\partial b_k}\right) + \gamma\Delta b_k(l-1) \\ &= \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} \mu e_n^p \frac{\partial \hat{y}_n^p}{\partial b_k}\right) + \gamma\Delta b_k(l-1). \end{aligned} \tag{25}$$

We have,

$$\frac{\partial \hat{y}_n^p}{\partial b_k} = \delta_{nk}, \tag{26}$$

where the Kronecker's symbol delta defined as: $\delta_{nk} = 1$ for $n = k$ and $\delta_{nk} = 0$ for $n \neq k$. Therefore, equation (25) can be rewritten as:

$$\Delta b_k(l) = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p\right) + \gamma\Delta b_k(l-1). \tag{27}$$

Similarly, let us define $\Delta a_{ki}(l)$ as follows:

$$\begin{aligned}\Delta a_{ki}(l) &= -\frac{\mu}{N_p}\frac{\partial J}{\partial a_{ki}} + \gamma \Delta a_{ki}(l-1) \\ &= -\frac{\mu}{N_p}\left(-\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} e_n^p \frac{\partial \hat{y}_n^p}{\partial a_{ki}}\right) + \gamma \Delta a_{ki}(l-1) \\ &= \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} \mu e_n^p \frac{\partial \hat{y}_n^p}{\partial a_{ki}}\right) + \gamma \Delta a_{ki}(l-1). \end{aligned} \tag{28}$$

Since,

$$\frac{\partial \hat{y}_n^p}{\partial a_{ki}} = \delta_{nk} x_i^p, \tag{29}$$

equation (28) can be rewritten as:

$$\Delta a_{ki}(l) = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p x_i^p\right) + \gamma \Delta a_{ki}(l-1). \tag{30}$$

Let `DWoi` $= \mathbf{DW_{oi}}$ be a 3D array with N_p pages, where each page constitutes of the following matrix:

$$\mathrm{DW}_{\mathrm{oi}}^p = \begin{bmatrix} db_1^p & da_{11}^p & \cdots & da_{1i}^p & \cdots & da_{1N_i}^p \\ db_2^p & da_{21}^p & \cdots & da_{2i}^p & \cdots & da_{2N_i}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ db_k^p & da_{k1}^p & \cdots & da_{ki}^p & \cdots & da_{kN_i}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ db_{N_o}^p & da_{N_o1}^p & \cdots & da_{N_oi}^p & \cdots & da_{N_oN_i}^p \end{bmatrix}. \tag{31}$$

Next, we can define the page vectors `DWoi(k,1,:)` $= Db_k$ and `DWoi(k,i,:)` $= Da_{ki}$ as stated below:

$$Db_k = \begin{bmatrix} db_k^1 & db_k^2 & \cdots & db_k^p & \cdots & db_k^{N_p} \end{bmatrix} = \mu E_k, \tag{32}$$

and,

$$Da_{ki} = \begin{bmatrix} da_{ki}^1 & da_{ki}^2 & \cdots & da_{ki}^p & \cdots & da_{ki}^{N_p} \end{bmatrix} = \mu E_k . X_i^{\mathrm{T}}. \tag{33}$$

The components of Db_k and Da_{ki} are computed as follows:

```
for k=1:No
    DWoi(k,1,:)=E(k,:)*u;
end
```

```
for k=1:No
    for i=1:Ni
     DWoi(k,i+1,:)=E(k,:)*u.*x(:,i)';
    end
end
```

Let,

$$d\bar{b}_k = \text{mean}\left[Db_k\right] = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p\right), \tag{34}$$

and,

$$d\bar{a}_{ki} = \text{mean}\left[Da_{ki}\right] = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p x_i^p\right). \tag{35}$$

Thus we can define the matrix `DWoi_aver` $= \text{D}\bar{\text{W}}_{\text{oi}}$ as follows:

$$\text{D}\bar{\text{W}}_{\text{oi}} = \begin{bmatrix} d\bar{b}_1 & d\bar{a}_{11} & \cdots & d\bar{a}_{1i} & \cdots & d\bar{a}_{1N_i} \\ d\bar{b}_2 & d\bar{a}_{21} & \cdots & d\bar{a}_{2i} & \cdots & d\bar{a}_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{b}_k & d\bar{a}_{k1} & \cdots & d\bar{a}_{ki} & \cdots & d\bar{a}_{kN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{b}_{N_o} & d\bar{a}_{N_o1} & \cdots & d\bar{a}_{N_oi} & \cdots & d\bar{a}_{N_oN_i} \end{bmatrix}. \tag{36}$$

The matrix $\text{D}\bar{\text{W}}_{\text{oi}}$ is computed using the code:

```
for k=1:No
    for i=1:Ni+1
        DWoi_aver(k,i)=mean(DWoi(k,i,:));
    end
```

```
end
```

In matrix form, equations (27) and (30) can be stated as:

$$\Delta \mathrm{W}_{\mathrm{oi}}(l) = \mathrm{D}\bar{\mathrm{W}}_{\mathrm{oi}} + \gamma \Delta \mathrm{W}_{\mathrm{oi}}(l-1). \tag{37}$$

where,

$$\Delta \mathrm{W}_{\mathrm{oi}}(l) = \begin{bmatrix} \Delta b_1(l) & \Delta a_{11}(l) & \cdots & \Delta a_{1i}(l) & \cdots & \Delta a_{1N_i}(l) \\ \Delta b_2(l) & \Delta a_{21}(l) & \cdots & \Delta a_{2i}(l) & \cdots & \Delta a_{2N_i}(l) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta b_k(l) & \Delta a_{k1}(l) & \cdots & \Delta a_{ki}(l) & \cdots & \Delta a_{kN_i}(l) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta b_{N_o}(l) & \Delta a_{N_o1}(l) & \cdots & \Delta a_{N_oi}(l) & \cdots & \Delta a_{N_oN_i}(l) \end{bmatrix}. \tag{38}$$

Using `DeltaWoi` $= \Delta \mathrm{W}_{\mathrm{oi}}(l)$ and `DeltaWoi_old` $= \Delta \mathrm{W}_{\mathrm{oi}}(l-1)$, equation (37) can be computed as follows:

```
DeltaWoi=DWoi_aver+gamma*DeltaWoi_old;
```

The parameters of W_{oi} are then updated as follows:

```
Woi=Woi+DeltaWoi;
```

Then for the next epoch we have,

```
DeltaWoi_old=DeltaWoi;
```

.3.2 Updating the parameters of $\mathrm{W_{oh}}$

Let us state the following equations for $\Delta c_{kj}\,(l)$:

$$\begin{aligned}\Delta c_{kj}\,(l) &= -\frac{\mu}{N_p}\frac{\partial J}{\partial c_{kj}} + \gamma \Delta c_{kj}\,(l-1) \\ &= -\frac{\mu}{N_p}\left(-\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} e_n^p \frac{\partial \hat{y}_n^p}{\partial c_{kj}}\right) + \gamma \Delta c_{kj}\,(l-1) \\ &= \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{n=1}^{N_o} \mu e_n^p \frac{\partial \hat{y}_n^p}{\partial c_{kj}}\right) + \gamma \Delta c_{kj}\,(l-1) \end{aligned} \tag{39}$$

Since,

$$\frac{\partial \hat{y}_n^p}{\partial c_{kj}} = \delta_{nk}\Phi_j^p, \tag{40}$$

equation (39) can be rewritten as:

$$\Delta c_{kj}\,(l) = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p \Phi_j^p\right) + \gamma \Delta c_{kj}\,(l-1)\,. \tag{41}$$

Let `DWoh` $= \mathbf{DW_{oh}}$ be a 3D array with N_p pages, where each page constitutes of the following matrix:

$$\mathrm{DW_{oh}^p} = \begin{bmatrix} dc_{11}^p & dc_{12}^p & \cdots & dc_{1i}^p & \cdots & dc_{1N_w}^p \\ dc_{21}^p & dc_{22}^p & \cdots & dc_{2i}^p & \cdots & dc_{2N_w}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dc_{k1}^p & dc_{k2}^p & \cdots & dc_{ki}^p & \cdots & dc_{kN_w}^p \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dc_{N_o1}^p & dc_{N_o2}^p & \cdots & dc_{N_oi}^p & \cdots & dc_{N_oN_w}^p \end{bmatrix}. \tag{42}$$

Next, let us define the page vector `DWoh(k,j,:)` $= Dc_{kj}$ as stated below:

$$Dc_{kj} = \begin{bmatrix} dc_{kj}^1 & dc_{kj}^2 & \cdots & dc_{kj}^p & \cdots & dc_{kj}^{N_p} \end{bmatrix} = \mu E_k.\Phi_j. \tag{43}$$

The components of Dc_{kj} are computed as follows:

```
for k=1:No
    for j=1:Nw
     DWoh(k,j,:)=E(k,:)*u.*PHI(j,:);
    end
end
```

Let,

$$d\bar{c}_{kj} = \text{mean}\left[Dc_{kj}\right] = \frac{1}{N_p}\left(\sum_{p=1}^{N_p} \mu e_k^p \Phi_j^p\right). \tag{44}$$

Next, define the matrix `DWoi_aver` $= \mathrm{D}\bar{\mathrm{W}}_{\mathrm{oi}}$ as follows:

$$\mathrm{D}\bar{\mathrm{W}}_{\mathrm{oh}} = \begin{bmatrix} d\bar{c}_{11} & d\bar{c}_{12} & \cdots & d\bar{c}_{1i} & \cdots & d\bar{c}_{1N_w} \\ d\bar{c}_{21} & d\bar{c}_{22} & \cdots & d\bar{c}_{2i} & \cdots & d\bar{c}_{2N_w} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{c}_{k1} & d\bar{c}_{k2} & \cdots & d\bar{c}_{ki} & \cdots & d\bar{c}_{kN_w} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{c}_{N_o1} & d\bar{c}_{N_o2} & \cdots & d\bar{c}_{N_oi} & \cdots & d\bar{c}_{N_oN_w} \end{bmatrix}. \tag{45}$$

The matrix $\mathrm{D}\bar{\mathrm{W}}_{\mathrm{oi}}$ is computed using the code:

```
for k=1:No
    for j=1:Nw
        DWoh_aver(k,j)=mean(DWoh(k,j,:));
    end
end
```

In matrix form equation (41) can be stated as:

$$\Delta \mathrm{W}_{\mathrm{oh}}(l) = \mathrm{D}\bar{\mathrm{W}}_{\mathrm{oh}} + \gamma \Delta \mathrm{W}_{\mathrm{oh}}(l-1), \tag{46}$$

where,

$$\Delta \mathrm{W_{oh}}(l) = \begin{bmatrix} \Delta c_{11}(l) & \Delta c_{12}(l) & \cdots & \Delta c_{1i}(l) & \cdots & \Delta c_{1N_w}(l) \\ \Delta c_{21}(l) & \Delta c_{22}(l) & \cdots & \Delta c_{2i}(l) & \cdots & \Delta c_{2N_w}(l) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta c_{k1}(l) & \Delta c_{k2}(l) & \cdots & \Delta c_{ki}(l) & \cdots & \Delta c_{kN_w}(l) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta c_{N_o1}(l) & \Delta c_{N_o2}(l) & \cdots & \Delta c_{N_oi}(l) & \cdots & \Delta c_{N_oN_w}(l) \end{bmatrix}. \quad (47)$$

Using `DeltaWoh` $= \Delta \mathrm{W_{oh}}(l)$ and `DeltaWoh_old` $= \mathrm{W_{oh}}(l-1)$, equation (46) can be computed as follows:

```
DeltaWoh=DWoh_aver+gamma*DeltaWoh_old;
```

The parameters of $\mathrm{W_{oh}}$ are then updated as follows:

```
Woh=Woh+DeltaWoh;
```

Then for the next epoch we have,

```
DeltaWoh_old=DeltaWoh;
```

.3.3 Updating the parameters of m and d

Let us first state the following equations for $\Delta m_{ji}(l)$ and $\Delta d_{ji}(l)$:

$$\begin{aligned} \Delta m_{ji}(l) &= -\frac{\mu}{N_p}\frac{\partial J}{\partial m_{ji}} + \gamma \Delta m_{ji}(l-1) \\ &= -\frac{\mu}{N_p}\left(-\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} e_k^p \frac{\partial \hat{y}_k^p}{\partial m_{ji}}\right) + \gamma \Delta m_{ji}(l-1) \\ &= \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} \mu e_k^p \frac{\partial \hat{y}_k^p}{\partial m_{ji}}\right) + \gamma \Delta m_{ji}(l-1), \end{aligned} \quad (48)$$

and,

$$\begin{aligned}\Delta d_{ji}(l) &= -\frac{\mu}{N_p}\frac{\partial J}{\partial d_{ji}} + \gamma \Delta d_{ji}(l-1) \\ &= -\frac{\mu}{N_p}\left(-\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} e_k^p \frac{\partial \hat{y}_k^p}{\partial d_{ji}}\right) + \gamma \Delta d_{ji}(l-1) \\ &= \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} \mu e_k^p \frac{\partial \hat{y}_k^p}{\partial d_{ji}}\right) + \gamma \Delta d_{ji}(l-1). \end{aligned} \tag{49}$$

Since,

$$\frac{\partial \hat{y}_k^p}{\partial m_{ji}} = c_{kj}\frac{\partial \Phi_j^p}{\partial z_{ji}^p}\frac{\partial z_{ji}^p}{\partial m_{ji}} = c_{kj}\dot{\Phi}_{ji}^p\left(-\frac{1}{d_{ji}}\right) = \left(-\frac{c_{kj}}{d_{ji}}\right)\dot{\Phi}_{ji}^p, \tag{50}$$

and,

$$\frac{\partial \hat{y}_k^p}{\partial d_{ji}} = c_{kj}\frac{\partial \Phi_j^p}{\partial z_{ji}^p}\frac{\partial z_{ji}^p}{\partial d_{ji}} = c_{kj}\dot{\Phi}_{ji}^p\left(-\frac{z_{ji}^p}{d_{ji}}\right) = \left(-\frac{c_{kj}}{d_{ji}}\right)\dot{\Phi}_{ji}^p z_{ji}^p, \tag{51}$$

then equations (48) and (49) can be rewritten as:

$$\Delta m_{ji}(l) = \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} \mu e_k^p\left[-\frac{c_{kj}}{d_{ji}}\dot{\Phi}_{ji}^p\right]\right) + \gamma \Delta m_{ji}(l-1), \tag{52}$$

and,

$$\Delta d_{ji}(l) = \frac{1}{N_p}\left(\sum_{p=1}^{N_p}\sum_{k=1}^{N_o} \mu e_k^p\left[-\frac{c_{kj}}{d_{ji}}\dot{\Phi}_{ji}^p z_{ji}^p\right]\right) + \gamma \Delta d_{ji}(l-1). \tag{53}$$

As can be seen from equations (52) and (53) to evaluate $\Delta m_{ji}(l)$ and $\Delta d_{ji}(l)$ we need to first find $\dot{\Phi}_{ji}^p$. Let `PHI_p` $= \dot{\mathbf{\Phi}}$ be a 3D array with N_p pages, where

each page consists of the following matrix:

$$\dot{\Phi}^p = \begin{bmatrix} \dot{\Phi}^p_{11} & \dot{\Phi}^p_{12} & \cdots & \dot{\Phi}^p_{1i} & \cdots & \dot{\Phi}^p_{1N_i} \\ \dot{\Phi}^p_{21} & \dot{\Phi}^p_{22} & \cdots & \dot{\Phi}^p_{2i} & \cdots & \dot{\Phi}^p_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \dot{\Phi}^p_{j1} & \dot{\Phi}^p_{j2} & \cdots & \dot{\Phi}^p_{ji} & \cdots & \dot{\Phi}^p_{jN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \dot{\Phi}^p_{N_w1} & \dot{\Phi}^p_{N_w2} & \cdots & \dot{\Phi}^p_{N_wi} & \cdots & \dot{\Phi}^p_{N_wN_i} \end{bmatrix}. \tag{54}$$

Using equation (23) the page vector $\dot{\Phi}_{ji}$ can be evaluated as follows:

$$\dot{\Phi}_{ji} = \begin{bmatrix} \dot{\Phi}^1_{ji} \\ \dot{\Phi}^2_{ji} \\ \vdots \\ \dot{\Phi}^p_{ji} \\ \vdots \\ \dot{\Phi}^{N_p}_{ji} \end{bmatrix} = \varphi_{j1}.\varphi_{j2}.\ldots.\dot{\varphi}_{ji}.\ldots.\varphi_{jN_i}. \tag{55}$$

The following block of code is used to evaluate the components of the 3D array $\dot{\mathbf{\Phi}}$:

```
for j=1:Nw
    for i=1:Ni
        P=phi(j,:,:);
        P(1,i,:)=phi_p(j,i,:);
        PHI_p(j,i,:)=prod(P);
    end
end
```

When entering the `for` loop at the ith and jth iteration, we first set `P=phi(j,:,:)` where P is a 3D array with N_p pages where each page consists of the row vector $\begin{bmatrix} \varphi^p_{j1} & \varphi^p_{j2} & \cdots & \varphi^p_{ji} & \cdots & \varphi^p_{jN_i} \end{bmatrix}$. Next, the page vector of P at the ith column is replaced by $\dot{\varphi}_{ji}$ using `P(1,i,:)=phi_p(j,i,:)`. Therefore, the new 3D array P has each page consisting of the row vector with the following components; $\begin{bmatrix} \varphi^p_{j1} & \varphi^p_{j2} & \cdots & \dot{\varphi}^p_{ji} & \cdots & \varphi^p_{jN_i} \end{bmatrix}$. Thus, the command

`PHI_p(j,i,:)=prod(P)` computes the product of each row vector for all the pages of P to produce the single page vector $\dot{\Phi}_{ji}$. For the next iteration, P is set back to `P=phi(j,:,:)`.

Next, let us state the following block of code:

```
for j=1:Nw
    for i=1:Ni
    EDmy=zeros(1,1,Np);
    EDdy=zeros(1,1,Np);
     for k=1:No
         p=squeeze(PHI_p(j,i,:));
         Dmy(k,:)=-(Woh(k,j)/d(j,i)).*p;
         zz=squeeze(z(j,i,:));
         Ddy(k,:)=Dmy(k,:).*zz';
         eDmy(1,1,:)=E(k,:)*u.*Dmy(k,:);
         eDdy(1,1,:)=E(k,:)*u.*Ddy(k,:);
         EDmy=EDmy+eDmy;
         EDdy=EDdy+eDdy;
     end
     Dm(j,i,:)=EDmy;
     Dd(j,i,:)=EDdy;
    end
end
```

At every kth iteration, for $k = 1, 2, \ldots, N_o$ we do the following:

(1) Use the command `p=squeeze(PHI_p(j,i,:))` to transform the page vector $\dot{\Phi}_{ji}$ into the column vector p.

(2) Define the row vector `Dmy(k,:)=-(Woh(k,j)/d(j,i)).*p` $= Dmy_k$ as stated below:

$$\begin{aligned} Dmy_k &= \left[\begin{array}{ccccccc} dmy_k^1 & dmy_k^2 & \cdots & dmy_k^p & \cdots & dmy_k^{N_p} \end{array}\right] \\ &= \left[\begin{array}{ccccccc} \frac{\partial \hat{y}_k^1}{\partial m_{ji}} & \frac{\partial \hat{y}_k^2}{\partial m_{ji}} & \cdots & \frac{\partial \hat{y}_k^p}{\partial m_{ji}} & \cdots & \frac{\partial \hat{y}_k^{N_p}}{\partial m_{ji}} \end{array}\right] \\ &= \left(-\frac{c_{kj}}{d_{ji}}\right).\dot{\Phi}_{ji} = \left(-\frac{c_{kj}}{d_{ji}}\right).p. \end{aligned} \tag{56}$$

(3) Use the command `zz=squeeze(z(j,i,:))` to transform the page vector Z_{ji} into a column vector zz. Similarly, let us define the row vector `Ddy(k,:)=Dmy(k,:).*zz'` $= Ddy_k$ as stated below:

$$\begin{aligned} Ddy_k &= \left[\begin{array}{ccccccc} ddy_k^1 & ddy_k^2 & \cdots & ddy_k^p & \cdots & ddy_k^{N_p} \end{array}\right] \\ &= \left[\begin{array}{ccccccc} \frac{\partial \hat{y}_k^1}{\partial d_{ji}} & \frac{\partial \hat{y}_k^2}{\partial d_{ji}} & \cdots & \frac{\partial \hat{y}_k^p}{\partial d_{ji}} & \cdots & \frac{\partial \hat{y}_k^{N_p}}{\partial d_{ji}} \end{array}\right] \\ &= \left(-\frac{c_{kj}}{d_{ji}}\right).\dot{\Phi}_{ji}.Z_{ji}^{\mathrm{T}} = \left(-\frac{c_{kj}}{d_{ji}}\right).p.zz^{\mathrm{T}} = Dmy_k.zz^{\mathrm{T}}. \end{aligned} \tag{57}$$

In the above equation the column vector zz has been transposed, just to be consistent with the code, since in Matlab programming, if element by element multiplication is carried out between two vectors, they must be of the same dimensions.

(4) Define the following:

```
eDmy(1,1,:)=E(k,:).*Dmy(k,:);
eDdy(1,1,:)=E(k,:).*Ddy(k,:);
```

where, `eDmy(1,1,:)` $= eDmy$ and `eDdy(1,1,:)` $= eDdy$ are page vectors that can be stated as:

$$eDmy = \left[\begin{array}{c} edmy^1 \\ edmy^2 \\ \vdots \\ edmy^p \\ \vdots \\ edmy^{N_p} \end{array}\right] = \left[\begin{array}{c} \mu e_k^1 dmy_k^1 \\ \mu e_k^2 dmy_k^2 \\ \vdots \\ \mu e_k^p dmy_k^p \\ \vdots \\ \mu e_k^{N_p} dmy_k^{N_p} \end{array}\right] = \mu E_k.Dmy_k, \tag{58}$$

and,

$$eDdy = \begin{bmatrix} eddy^1 \\ eddy^2 \\ \vdots \\ eddy^p \\ \vdots \\ eddy^{N_p} \end{bmatrix} = \begin{bmatrix} \mu e_k^1 ddy_k^1 \\ \mu e_k^2 ddy_k^2 \\ \vdots \\ \mu e_k^p ddy_k^p \\ \vdots \\ \mu e_k^{N_p} ddy_k^{N_p} \end{bmatrix} = \mu E_k.Ddy_k \tag{59}$$

In this case, μ has been set to 1 to ensure convergence of the MSE.

(5) Next, define the following page vectors:

$$EDmy = \begin{bmatrix} \sum_{k=1}^{N_o} \mu e_k^1 dmy_k^1 \\ \sum_{k=1}^{N_o} \mu e_k^2 dmy_k^2 \\ \vdots \\ \sum_{k=1}^{N_o} \mu e_k^p dmy_k^p \\ \vdots \\ \sum_{k=1}^{N_o} \mu e_k^{N_p} dmy_k^{N_p} \end{bmatrix}, EDdy = \begin{bmatrix} \sum_{k=1}^{N_o} \mu e_k^1 ddy_k^1 \\ \sum_{k=1}^{N_o} \mu e_k^2 ddy_k^2 \\ \vdots \\ \sum_{k=1}^{N_o} \mu e_k^p ddy_k^p \\ \vdots \\ \sum_{k=1}^{N_o} \mu e_k^{N_p} ddy_k^{N_p} \end{bmatrix}. \tag{60}$$

This is done using the following commands:

```
EDmy=EDmy+eDmy;
EDdy=EDdy+eDdy;
```

where `EDmy` and `EDdy` are fist initialized to zeros, using:

```
EDmy=zeros(1,1,Np);
EDdy=zeros(1,1,Np);
```

After exiting the `for` loop `k=1:No`, we set the page vectors:

```
Dm(j,i,:)=EDmy;
Dd(j,i,:)=EDdy;
```

where, `Dm(j,i,:)` $= Dm_{ji}$ and `Dd(j,i,:)` $= Dd_{ji}$ can be stated as:

$$Dm_{ji} = \begin{bmatrix} dm_{ji}^{1} \\ dm_{ji}^{2} \\ \vdots \\ dm_{ji}^{p} \\ \vdots \\ dm_{ji}^{N_p} \end{bmatrix}, Dd_{ji} = \begin{bmatrix} dd_{ji}^{1} \\ dd_{ji}^{2} \\ \vdots \\ dd_{ji}^{p} \\ \vdots \\ dd_{ji}^{N_p} \end{bmatrix}. \tag{61}$$

When we exit the `for` loop `j=1:Nw`, we would have constructed the 3D arrays, **Dm** and **Dd** with N_p pages where each page consists of the matrices:

$$\mathrm{Dm}^{p} = \begin{bmatrix} dm_{11}^{p} & dm_{12}^{p} & \cdots & dm_{1i}^{p} & \cdots & dm_{1N_i}^{p} \\ dm_{21}^{p} & dm_{22}^{p} & \cdots & dm_{2i}^{p} & \cdots & dm_{2N_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dm_{j1}^{p} & dm_{j2}^{p} & \cdots & dm_{ji}^{p} & \cdots & dm_{jN_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dm_{N_w1}^{p} & dm_{N_w2}^{p} & \cdots & dm_{N_wi}^{p} & \cdots & dm_{N_wN_i}^{p} \end{bmatrix}, \tag{62}$$

and,

$$\mathrm{Dd}^{p} = \begin{bmatrix} dd_{11}^{p} & dd_{12}^{p} & \cdots & dd_{1i}^{p} & \cdots & dd_{1N_i}^{p} \\ dd_{21}^{p} & dd_{22}^{p} & \cdots & dd_{2i}^{p} & \cdots & dd_{2N_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dd_{j1}^{p} & dd_{j2}^{p} & \cdots & dd_{ji}^{p} & \cdots & dd_{jN_i}^{p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ dd_{N_w1}^{p} & dd_{N_w2}^{p} & \cdots & dd_{N_wi}^{p} & \cdots & dd_{N_wN_i}^{p} \end{bmatrix}. \tag{63}$$

Next, let us define the matrices, `Dm_aver` $= \mathrm{D}\bar{\mathrm{m}}$ and `Dd_aver` $= \mathrm{D}\bar{\mathrm{d}}$ as follows:

$$\mathrm{D}\bar{\mathrm{m}} = \begin{bmatrix} d\bar{m}_{11} & d\bar{m}_{12} & \cdots & d\bar{m}_{1i} & \cdots & d\bar{m}_{1N_i} \\ d\bar{m}_{21} & d\bar{m}_{22} & \cdots & d\bar{m}_{2i} & \cdots & d\bar{m}_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{m}_{j1} & d\bar{m}_{j2} & \cdots & d\bar{m}_{ji} & \cdots & d\bar{m}_{jN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{m}_{N_w1} & d\bar{m}_{N_w2} & \cdots & d\bar{m}_{N_wi} & \cdots & d\bar{m}_{N_wN_i} \end{bmatrix}, \tag{64}$$

and,

$$\mathrm{D}\bar{\mathrm{d}} = \begin{bmatrix} d\bar{d}_{11} & d\bar{d}_{12} & \cdots & d\bar{d}_{1i} & \cdots & d\bar{d}_{1N_i} \\ d\bar{d}_{21} & d\bar{d}_{22} & \cdots & d\bar{d}_{2i} & \cdots & d\bar{d}_{2N_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{d}_{j1} & d\bar{d}_{j2} & \cdots & d\bar{d}_{ji} & \cdots & d\bar{d}_{jN_i} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ d\bar{d}_{N_w1} & d\bar{d}_{N_w2} & \cdots & d\bar{d}_{N_wi} & \cdots & d\bar{d}_{N_wN_i} \end{bmatrix}. \tag{65}$$

The matrices $\mathrm{D}\bar{\mathrm{m}}$ and $\mathrm{D}\bar{\mathrm{d}}$ are computed using the following code:

```
for j=1:Nw
    for i=1:Ni
        Dm_aver(j,i)=mean(Dm(j,i,:));
        Dd_aver(j,i)=mean(Dd(j,i,:));
    end
end
```

In matrix form equations (52) and (53) can be stated as follows:

$$\Delta\mathrm{m}\,(l) = \mathrm{D}\bar{\mathrm{m}} + \gamma\Delta\mathrm{m}\,(l-1)\,, \tag{66}$$

and,

$$\Delta\mathrm{d}\,(l) = \mathrm{D}\bar{\mathrm{d}} + \gamma\Delta\mathrm{d}\,(l-1)\,, \tag{67}$$

where,

$$\Delta \mathrm{m}\left(l\right) = \begin{bmatrix} \Delta m_{11}\left(l\right) & \Delta m_{12}\left(l\right) & \cdots & \Delta m_{1i}\left(l\right) & \cdots & \Delta m_{1N_i}\left(l\right) \\ \Delta m_{21}\left(l\right) & \Delta m_{22}\left(l\right) & \cdots & \Delta m_{2i}\left(l\right) & \cdots & \Delta m_{2N_i}\left(l\right) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta m_{j1}\left(l\right) & \Delta m_{j2}\left(l\right) & \cdots & \Delta m_{ji}\left(l\right) & \cdots & \Delta m_{jN_i}\left(l\right) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta m_{N_w1}\left(l\right) & \Delta m_{N_w2}\left(l\right) & \cdots & \Delta m_{N_wi}\left(l\right) & \cdots & \Delta m_{N_wN_i}\left(l\right) \end{bmatrix}, \quad (68)$$

and,

$$\Delta \mathrm{d}\left(l\right) = \begin{bmatrix} \Delta d_{11}\left(l\right) & \Delta d_{12}\left(l\right) & \cdots & \Delta d_{1i}\left(l\right) & \cdots & \Delta d_{1N_i}\left(l\right) \\ \Delta d_{21}\left(l\right) & \Delta d_{22}\left(l\right) & \cdots & \Delta d_{2i}\left(l\right) & \cdots & \Delta d_{2N_i}\left(l\right) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta d_{j1}\left(l\right) & \Delta d_{j2}\left(l\right) & \cdots & \Delta d_{ji}\left(l\right) & \cdots & \Delta d_{jN_i}\left(l\right) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Delta d_{N_w1}\left(l\right) & \Delta d_{N_w2}\left(l\right) & \cdots & \Delta d_{N_wi}\left(l\right) & \cdots & \Delta d_{N_wN_i}\left(l\right) \end{bmatrix}. \quad (69)$$

Using `Deltam` $= \Delta \mathrm{m}\left(l\right)$ and `Deltam_old` $= \Delta \mathrm{m}\left(l-1\right)$, equation (66) can be computed as follows:

```
Deltam=Dm_aver+gamma*Deltam_old
```

Similarly, using `Deltad` $= \Delta \mathrm{d}\left(l\right)$ and `Deltad_old` $= \Delta \mathrm{d}\left(l-1\right)$, equation (67) can be computed as follows:

```
Deltad=Dd_aver+gamma*Deltad_old
```

The parameter of `m` and `d` are then updated as follows:

```
m=m+Deltam;
d=d+Deltad;
```

Then for the next epoch we have,

```
Deltam_old=Deltam;
Deltad_old=Deltad;
```

After completing the desired number of epochs or iterations, a plot of the MSE is generated as follows:

```
loglog(MSE,'*')
xlabel('Iterations')
ylabel('MSE')
grid on
```

Appenndix C: DAQ Code

```
/*********************************************************************
*
* ANSI C Example program:
*    Acq-IntClk.c
*
* Example Category:
*    AI
*
* Description:
*    This example demonstrates how to acquire a finite amount of data
*    using the DAQ device's internal clock.
*
* Instructions for Running:
*    1. Select the physical channel to correspond to where your
*       signal is input on the DAQ device.
*    2. Enter the minimum and maximum voltages.
*    Note: For better accuracy try to match the input range to the
*          expected voltage level of the measured signal.
*    3. Select the number of samples to acquire.
*    4. Set the rate of the acquisition.
*    Note: The rate should be AT LEAST twice as fast as the maximum
*          frequency component of the signal being acquired.
*
* Steps:
*    1. Create a task.
*    2. Create an analog input voltage channel.
*    3. Set the rate for the sample clock. Additionally, define the
*       sample mode to be finite and set the number of samples to be
*       acquired per channel.
*    4. Call the Start function to start the acquisition.
*    5. Read all of the waveform data.
*    6. Call the Clear Task function to clear the task.
*    7. Display an error if any.
*
* I/O Connections Overview:
*    Make sure your signal input terminal matches the Physical
*    Channel I/O Control. For further connection information, refer
*    to your hardware reference manual.
*
*********************************************************************/
```

```
#include <C:\Program Files\Microsoft Visual Studio\VC98\Include\stdio.h>
#include <C:\Program Files\Microsoft Visual Studio\VC98\Include\stdlib.h>
#include <C:\Program Files\Microsoft Visual Studio\VC98\Include\string.h>
#include <C:\Program Files\Microsoft Visual Studio\VC98\Include\cstring>
#include <NIDAQmx.h>

#define DAQmxErrChk(functionCall)
if( DAQmxFailed(error=(functionCall)) ) goto Error; else
int main(void)
{
    int32       error=0;
    TaskHandle  taskHandle=0;
    TaskHandle  task=0;
    float64     a[4];
    float64     photo[1000][4];
    int32       value;
    uInt8       data[8]={0,0,0,0,0,0,0,0};
    char        errBuff[2048]={'\0'};
    int         i,k,m,n;
    FILE*fin;

    /*********************************************/
    // DAQmx Configure Code (Voltage Acquisition)
    /*********************************************/
    DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
    DAQmxErrChk (DAQmxCreateAIVoltageChan(taskHandle,
    "Dev2/ai0,Dev2/ai1,Dev2/ai2,Dev2/ai3","",
    DAQmx_Val_Diff,-5.0,5.0,DAQmx_Val_Volts,NULL));

    /*********************************************/
    // DAQmx Configure Code (Writing Digital)
    /*********************************************/
    DAQmxErrChk (DAQmxCreateTask("",&task));
    DAQmxErrChk (DAQmxCreateDOChan(task,"Dev2/port0/line0:7",
    "",DAQmx_Val_ChanForAllLines));

    /*********************************************/
    // DAQmx Start Code (Writing Digital)
    /*********************************************/
    DAQmxErrChk (DAQmxStartTask(task));


    /*********************************************/
    // DAQmx Start Code (Voltage Acquisition)
    /*********************************************/
    DAQmxErrChk (DAQmxStartTask(taskHandle));


    /*********************************************/
    // Create and open file PD.txt
    /*********************************************/
    fin=fopen("PD.txt","w");
    /*********************************************/
    // DAQmx Read Code and Write code
    /*********************************************/
        //Setting Duty cycle to 50% (114 counts) via
        //PORTB of the microcontroller
        data[7]=0;
```

```
            data[6]=1;
            data[5]=1;
            data[4]=1;
            data[3]=0;
            data[1]=1;
            data[0]=1;
            DAQmxErrChk (DAQmxWriteDigitalLines(task,1,1,10.0,
            DAQmx_Val_GroupByChannel,data,NULL,NULL));

        for(k=0;k<75;k++){

            DAQmxErrChk (DAQmxReadAnalogF64(taskHandle,1,10.0,
            DAQmx_Val_GroupByChannel,a,4,&value,NULL));
            DAQmxErrChk (DAQmxWriteDigitalLines(task,1,1,10.0,
            DAQmx_Val_GroupByChannel,data,NULL,NULL));
            photo[k][0]=a[0];
            photo[k][1]=a[1];
            photo[k][2]=a[2];
            photo[k][3]=a[3];
            data[0]=1-data[0];
            for(m=0;m<50000;m++){
                for(n=0;n<10000;n++){
                    m=m+0;
                }
            }

        }
          data[7]=0;
            data[6]=0;
            data[5]=0;
            data[4]=0;
            data[3]=0;
            data[1]=0;
            DAQmxErrChk (DAQmxWriteDigitalLines(task,1,1,10.0,
            DAQmx_Val_GroupByChannel,data,NULL,NULL));

        /*********************************************/
        // Save results in the file PD
        /*********************************************/
        for(i=0;i<75;i++){
            fprintf(fin,"%f     %f      %f      %f\n",
            photo[i][0],photo[i][1],photo[i][2],photo[i][3]);

        }
        /*********************************************/
        // Close the file
        /*********************************************/
        fclose(fin);

    Error:
        if( DAQmxFailed(error) )
            DAQmxGetExtendedErrorInfo(errBuff,2048);
        if( taskHandle!=0 )  {
            /*********************************************/
            // DAQmx Stop Code (Reading function)
            /*********************************************/
            DAQmxStopTask(taskHandle);
            DAQmxClearTask(taskHandle);
```

```
        }
        if( task!=0 )  {
            /*********************************************/
            // DAQmx Stop Code (Writing function)
            /*********************************************/
            DAQmxStopTask(task);
            DAQmxClearTask(task);
        }

        if( DAQmxFailed(error) )
            printf("DAQmx Error: %s\n",errBuff);
        printf("End of program, press Enter key to quit\n");
        getchar();
        return 0;
    }
```

Appenndix D: Microcontroller Code

```
/////////////////////////////main.c/////////////////////////////////

#include <mc9s12dg256.h>       /* Derivative information */
#include "lcd.h"               /* LCD header */
#include "t2i.h"               /* Input Capture header */
#include "pll.h"               // Defines _BUSCLOCK, sets bus
#include "sci1.h"              // frequency to _BUSCLOCK MHz */


#pragma LINK_INFO DERIVATIVE "mc9s12dg256b"

void  show_result(void);      /*Show_result function prototype*/

////////////////////////Declarations//////////////////////////////
   char x,y;

//////////////////////PWM_Initialization/////////////////////////
//  PWMCTL : PWM Control Register                                //
//         'CONxy=0': Concatenation Disabled (8-bit PWM)         //
//         'PSWAI=0': Continue while in wait mode                //
//         'PFRZ =0': PWM will continue while in freeze mode     //
//  PWMPOL : PWM Polarity Register                               //
//         'PPOL1=1': PWM channel 1 is high at the beginning     //
//  PWMPRCLK: PWM Prescale Clock Select Register                 //
//         'PCKA2=0': Clock A (No Prescaler Division)            //
//         'PCKA1=0': Clock A (No Prescaler Division)            //
//         'PCKA0=0': Clock A (No Prescaler Division)            //
//  PWMCLK : PWM Clock Select Register                           //
//         'PCLK1=1': Clock SA is the clock source for PWM CH1//
//  PWMSCLA: PWM Scale A Register                                //
//  PWMCAE : PWM Center Align Enable Register                    //
//         'CAE1=0':CH1 operates in Left Aligned Output Mode     //
//  PWMPER1: PWM Channel 1 Period Register                       //
//  PWMDTY1: PWM Channel 1 Duty Register                         //
//  PWME   : PWM Enable Register                                 //
//         'PWME1=1': Pulse Width channel 1 is enabled           //
////////////////////////////////////////////////////////////////////
```

```
void init_PWM(void) {
  //PWM signal is generated at PP0 (Motor X) and PP1 (Motor Y) pins

  DDRB=0x00;
  y=PORTB;
  x=y & 0xFA;
  PWMCTL   = 0x00;     /*8-bit mode*/
    PWMPOL   = 0xFF;     /*High polarity mode*/
  PWMPRCLK = 0x04;
  PWMCLK   = 0x03;    //clock source is clock SA
    PWMSCLA  = 10;
  PWMCAE   = 0x00;    //output is left aligned
  PWMPER0  = 225;     //PWM freq. =330Hz,
  PWMPER1  = 225;     //PWM freq. =330Hz,
  PWMDTY0  = 0;       //Duty cycle is 50% of the CW direction
  PWMDTY1  = 0;       //Duty cycle is 50% of the CW direction
  PWME = 0x03;        //Enable PWM channel 0 and channel 1

}


////////////////////////////MAIN/////////////////////////////////


void main(void) {

  /* set system clock frequency to _BUSCLOCK MHz (24 or 4) */
  PLL_Init();

  EnableInterrupts;

  t2_Init();

  init_PWM();


  ADC_Init();

  SCI1_Init(BAUD_115200);

  /* initialize LCD display */
  initLCD();
  //setting PORTA as output,pin 0 BRK for Y, pin 2 DIR for Y,
  //pin 4 BRK for X, pin 6 DIR for X
  DDRA=0xFF;
  PORTA= 0x10;

  for(;;){
  DDRB=0x00;
  y=PORTB;
  x=y & 0xFA;
  PWMDTY0  = x;
  PWMDTY1  = x;
  show_result();
  }

}
```

```
//////////////////////////Show_Result//////////////////////////////

void  show_result() {

  lcd_goto(Line1, 0);
  lcd_puts("   ");
  put_signed_num(countx2);
  lcd_puts("   ");
  put_signed_num(countx1);
  lcd_puts("   ");

  lcd_goto(Line2, 0);
  lcd_puts("   ");
  put_signed_num(county2);
  lcd_puts("   ");
  put_signed_num(county1);
  lcd_puts("   ");

 }


//////////////////////////THE_END//////////////////////////////////
```

```
//////////////////////isr_vectors.c////////////////////////////

extern void near _Startup(void);       /* Startup routine */
/* declarations of interrupt service routines */
extern __interrupt void OV_F_ISR(void);
extern __interrupt void TIC2ISRX(void);
extern __interrupt void TIC2ISRY(void);
extern __interrupt void TICISR(void);
extern __interrupt void SCI1_isr(void);

#pragma CODE_SEG __NEAR_SEG NON_BANKED /* Interrupt section
for this module. Placement will be in NON_BANKED area. */
__interrupt void UnimplementedISR(void) {

   /* Unimplemented ISRs trap.*/
   asm BGND;
}

typedef void (*near tIsrFunc)(void);
const tIsrFunc _vect[] @0xFF80 = {     /* Interrupt table */
UnimplementedISR,      /* vector 63 : (reserved) */
UnimplementedISR,      /* vector 62 : (reserved) */
UnimplementedISR,      /* vector 61 : (reserved) */
UnimplementedISR,      /* vector 60 : (reserved) */
UnimplementedISR,      /* vector 59 : (reserved) */
UnimplementedISR,      /* vector 58 : (reserved) */
UnimplementedISR,      /* vector 57 : PWM emergency shutdown */
UnimplementedISR,      /* vector 56 : PORT P */
UnimplementedISR,      /* vector 55 : MSCAN4 - transmit */
UnimplementedISR,      /* vector 54 : MSCAN4 - receive */
UnimplementedISR,      /* vector 53 : MSCAN4 - errors */
UnimplementedISR,      /* vector 52 : MSCAN4 - wakeup */
UnimplementedISR,      /* vector 51 : MSCAN3 - transmit */
UnimplementedISR,      /* vector 50 : MSCAN3 - receive */
UnimplementedISR,      /* vector 49 : MSCAN3 - errors */
UnimplementedISR,      /* vector 48 : MSCAN3 - wakeup */
UnimplementedISR,      /* vector 47 : MSCAN2 - transmit */
UnimplementedISR,      /* vector 46 : MSCAN2 - receive */
UnimplementedISR,      /* vector 45 : MSCAN2 - errors */
UnimplementedISR,      /* vector 44 : MSCAN2 - wakeup */
UnimplementedISR,      /* vector 43 : MSCAN1 - transmit */
UnimplementedISR,      /* vector 42 : MSCAN1 - receive */
UnimplementedISR,      /* vector 41 : MSCAN1 - errors */
UnimplementedISR,      /* vector 40 : MSCAN1 - wakeup */
UnimplementedISR,      /* vector 39 : MSCAN0 - transmit */
UnimplementedISR,      /* vector 38 : MSCAN0 - receive */
UnimplementedISR,      /* vector 37 : MSCAN0 - errors */
UnimplementedISR,      /* vector 36 : MSCAN0 - wakeup */
UnimplementedISR,      /* vector 35 : FLASH */
UnimplementedISR,      /* vector 34 : EEPROM */
UnimplementedISR,      /* vector 33 : SPI2 */
UnimplementedISR,      /* vector 32 : SPI1 */
UnimplementedISR,      /* vector 31 : IIC bus */
UnimplementedISR,      /* vector 30 : DLC */
UnimplementedISR,      /* vector 29 : SCME */
UnimplementedISR,      /* vector 28 : CRG lock */
UnimplementedISR,      /* vector 27 : Pulse accumulator B overflow */
```

```
UnimplementedISR,      /* vector 26 : Modulus down counter underflow */
UnimplementedISR,      /* vector 25 : PORT H */
UnimplementedISR,      /* vector 24 : PORT J */
UnimplementedISR,      /* vector 23 : ATD1 */
UnimplementedISR,      /* vector 22 : ATD0 */
SCI1_isr,              /* vector 21 : SCI1 (TIE, TCIE, RIE, ILIE) */
UnimplementedISR,      /* vector 20 : SCI0 (TIE, TCIE, RIE, ILIE) */
UnimplementedISR,      /* vector 19 : SPI0 */
UnimplementedISR,      /* vector 18 : Pulse accumulator input edge */
UnimplementedISR,      /* vector 17 : Pulse accumulator A overflow */
OV_F_ISR,              /* vector 16 : Timer Overflow (TOF) */
UnimplementedISR,      /* vector 15 : Timer channel 7 */
UnimplementedISR,      /* vector 14 : Timer channel 6 */
UnimplementedISR,      /* vector 13 : Timer channel 5 */
UnimplementedISR,      /* vector 12 : Timer channel 4 */
TIC2ISRY,              /* vector 11 : Timer channel 3 */
TIC2ISRX,              /* vector 10 : Timer channel 2 */
TICISR,                /* vector 09 : Timer channel 1 */
UnimplementedISR,      /* vector 08 : Timer channel 0 */
UnimplementedISR,      /* vector 07 : Real-Time Interrupt (RTI) */
UnimplementedISR,      /* vector 06 : IRQ */
UnimplementedISR,      /* vector 05 : XIRQ */
UnimplementedISR,      /* vector 04 : SWI */
UnimplementedISR,      /* vector 03 : Unimplemented Instruction trap */
UnimplementedISR,      /* vector 02 : COP failure reset*/
UnimplementedISR,      /* vector 01 : Clock monitor fail reset */
_Startup               /* vector 00 : Reset vector */
  };
```

```
//////////////////////////////pll.h//////////////////////////////////

/*************************************************************
*   boosts the CPU clock to 48 MHz                           *
*************************************************************/

// modified to define _BUSCLOCK
// PLL now running at 48 MHz to be consistent with HCS12 Serial Monitor
// fw-07-04


#ifndef _PPL_H_
#define _PLL_H_

/* Define the desired bus clock frequency:
no PLL (crystal) -> SYSCLOCK = 4 MHz   -> BUSCLOCK = 2 MHz
PLL on           -> SYSCLOCK = 48 MHz  -> BUSCLOCK = 24 MHz
This is used by sci0.c and/or sci1.c to determine the baud rate divider */
#define _BUSCLOCK 24
//********* PLL_Init ***************
// Set PLL clock to 48 MHz, and switch 9S12 to run at this rate
// Inputs: none
// Outputs: none
// Errors: will hang if PLL does not stabilize
void PLL_Init(void);

#endif  /* _PLL_H_ */
```

```
////////////////////////////pll.c/////////////////////////////////

/*************************************************************
*   boosts the CPU clock to 48 MHz                           *
*************************************************************/

// modified to make PLL_Init depend on _BUSCLOCK (defined in pll.h)
// PLL now running at 48 MHz to be consistent with HCS12 Serial Monitor
// fw-07-04

#include <hidef.h>              /* common defines and macros */
#include <mc9s12dp256.h>        /* derivative information */
#include "pll.h"                /* macro _BUSCLOCK */

//********* PLL_Init ***************
// Set PLL clock to 48 MHz, and switch 9S12 to run at this rate
// Inputs: none
// Outputs: none
// Errors: will hang if PLL does not stabilize
void PLL_Init(void){

  /* ensure we're running the controller at an appropriate clock speed */
  #if (_BUSCLOCK != 24 && _BUSCLOCK != 4)
  #error pll.h: _BUSCLOCK has to be set to 4 (MHz) or 24 (MHz)
  #endif

  /* set PLL clock speed */
  #if _BUSCLOCK == 24
  SYNR = 0x05;      // PLLOSC = 48 MHz
  #else
  SYNR = 0x00;      // PLLOSC = 8 MHz
  #endif

  REFDV = 0x00;

  /* PLLCLK = 2 * OSCCLK * (SYNR + 1) / (REFDV + 1)
       Values above give PLLCLK of 48 MHz with 4 MHz crystal.
       (OSCCLK is Crystal Clock Frequency)                    */

  CLKSEL = 0x00;

  /*Meaning for CLKSEL:
  Bit 7: PLLSEL = 0 Keep using OSCCLK until we are
                          ready to switch to PLLCLK
  Bit 6: PSTP   = 0 Do not need to go to Pseudo-Stop Mode
  Bit 5: SYSWAI = 0 In wait mode system clocks stop.
  But 4: ROAWAI = 0 Do not reduce oscillator amplitude in wait mode.
  Bit 3: PLLWAI = 0 Do not turn off PLL in wait mode
  Bit 2: CWAI   = 0 Do not stop the core during wait mode
  Bit 1: RTIWAI = 0 Do not stop the RTI in wait mode
  Bit 0: COPWAI = 0 Do not stop the COP in wait mode
  */


  PLLCTL = 0xD1;

  /*Meaning for PLLCTL:
```

```
  Bit 7: CME   = 1; Clock monitor enable — reset if
                       bad clock when set
  Bit 6: PLLON = 1; PLL On bit
  Bit 5: AUTO  = 0; No automatic control of bandwidth,
                       manual through ACQ
  But 4: ACQ   = 1; 1 for high bandwidth filter (acquisition);
                        0 for low (tracking)
  Bit 3:            (Not Used by 9s12c32)
  Bit 2: PRE   = 0; RTI stops during Pseudo Stop Mode
  Bit 1: PCE   = 0; COP diabled during Pseudo STOP mode
  Bit 0: SCME  = 1; Crystal Clock Failure –> Self Clock mode NOT reset.

  */

  while((CRGFLG&0x08) == 0){      // Wait for PLLCLK to stabilize.
  }
  CLKSEL_PLLSEL = 1;  // Switch to PLL clock
}
```

```
////////////////////////////t2i.h////////////////////////////////

/*************************************************************
**  In this header file we are using the input capture.  **
**  Bus Clock = 24 MHz ,  Prescaler = 1                  **
*************************************************************/

#include <hidef.h> /* common defines and macros */
#include <mc9s12dg256.h> /* derivative information */

#define MAX 4

extern __interrupt void OV_F_ISR(void);
extern __interrupt void TIC2ISRX(void);
extern __interrupt void TIC2ISRY(void);
extern __interrupt void TICISR(void);
//Global variables
  float   CounterX=0,CounterY=0;
  int countx1=0,countx2=0,county1=0,county2=0;

//Initialization function
void t2_Init()
{
  //connect emcoder to pt2
  //TC7   = 0x0000; //Channel 7 compare register is set to 0
  TSCR2 = 0x0A;   //TCRE bit is set so that free running counter
  // is set to 0000, prescale of 4
  TIE   = 0x0E;   //Channel 1,2,3 interrupt enabled
  TSCR2 = 0x80;   //TOI inhibited and prescaler factor = 1
  TCTL4 = 0x5C;   //Captures on rising edges only for pins 2 and 3,
  // both edges for pin 1
  TIOS  = 0x00;   //Channel 1,2,3 acts as input capture
  TSCR1 = 0x80;   //Timer is enabled and normal flag clearing
  DDRB  = 0x00;   //setting PORTB as input,pin 0 UP/DWN for X,
  // pin 2 UP/DWN for Y
}

//Interrupt subroutine
#pragma CODE_SEG NON_BANKED
#pragma TRAP_PROC

void TIC2ISRX()

{     if((PORTB & 0x01)==0x00) {
       if (countx1==0x00) {
      countx1=0xff;
      countx2--;
     }
      else{
        countx1--;
      }
      }
      if((PORTB & 0x01)==0x01) {
        if (countx1==0xff) {
      countx1=0x00;
      countx2++;
     }
```

```
      else{
        countx1++;
      }
      }

  TFLG1 |= 0x04;    //pin2 CLKX
}

void TIC2ISRY()
{
      if((PORTB & 0x04)==0x00) {
       if (county1==0x00) {
      county1=0xff;
      county2--;
     }

      else{
        county1--;
      }
      }
      if((PORTB & 0x04)==0x04) {
        if (county1==0xff) {
      county1=0x00;
      county2++;
     }
      else{
        county1++;
      }
      }

  TFLG1 |= 0x08;   //pin3 CLKY
}

void TICISR()

{
   SCI1_OutChar(0x0d);
   //SCI1_OutUDec(countx2);
  // SCI1_OutChar(' ');
  // SCI1_OutUDec(countx1);
  // SCI1_OutChar(' ');
   SCI1_OutUDec(county2);
   SCI1_OutChar(' ');
   SCI1_OutUDec(county1);

  TFLG1 |= 0x02;    //pin1
}

void OV_F_ISR() {   /*Overflow Interrupt*/

  TFLG2 = 0x80;
}
#pragma CODE_SEG DEFAULT
```

```
////////////////////////////lcd.h/////////////////////////////////

/* Dragon-12 LCD Header file  */

#ifndef _LCD_H_
#define _LCD_H_


/* declare public functions */
void initLCD(void);               // must be called first to init LCD
void lcd_puts(char *string);      // write a string to LCD
//write single char as command or data to LCD
lcd_write(unsigned char x, unsigned char rs);
void put_num(unsigned int no);    // write a number to LCD
void put_signed_num(int num);     // write signed nu
//Move cursor to specific Line and offset location
void lcd_goto(unsigned char line, unsigned char offset);
#define Line1 0x80            // Line1 address in LCD
#define Line2 0xc0            // Line2 address in LCD
#endif /* _LCD_H_ */
```

```
//////////////////////////////sci1.h////////////////////////////////

// filename  ******************* sci1.h *************************
// Jonathan W. Valvano 1/29/04

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Brooks-Cole, copyright (c) 2000,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Brooks-Cole,
//    copyright (c) 2002

// Copyright 2004 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains
// Modified by EE345L students Charlie Gough && Matt Hawk
// Modified by EE345M students Agustinus Darmawan + Mingjie Qiu
//
// adapted to the Dragon12 board using SCI1  --  fw-07-04

// define labels for baudrates
// (necessary 'coz 115200 isn't a 16-bit number anymore  --  fw-08-04)
#define BAUD_300      0
#define BAUD_600      1
#define BAUD_1200     2
#define BAUD_2400     3
#define BAUD_4800     4
#define BAUD_9600     5
#define BAUD_19200    6
#define BAUD_38400    7
#define BAUD_57600    8
#define BAUD_115200   9


// standard ASCII symbols
#define CR   0x0D
#define LF   0x0A
#define BS   0x08
#define ESC  0x1B
#define SP   0x20
#define DEL  0x7F

//------------------------SCI1_Init-------------------------
// Initialize Serial port SCI1
// Input: baudRate is tha baud rate in bits/sec
// Output: none
extern void SCI1_Init(unsigned short baudRate);

//------------------------SCI1_InStatus-------------------------
// Checks if new input is ready, TRUE if new input is ready
// Input: none
// Output: TRUE if a call to InChar will return right away with data
//         FALSE if a call to InChar will wait for input
extern char SCI1_InStatus(void);

//------------------------SCI1_InChar-------------------------
// Wait for new serial port input, busy-waiting synchronization
```

```
// Input: none
// Output: ASCII code for key typed
extern char SCI1_InChar(void);

// Reads in a String of max length
extern void SCI1_InString(char *, unsigned short);

//------------------------SCI1_InUDec-----------------------------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 16 bit unsigned number
//     valid range is 0 to 65535
// Input: none
// Output: 16-bit unsigned number
// If you enter a number above 65535, it will truncate without an error
// Backspace will remove last digit typed
extern unsigned short SCI1_InUDec(void);

//------------------------SCI1_InULDec-----------------------------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 32 bit unsigned number
//     valid range is 0 to 4294967296
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967296, it will truncate
// without an error
// Backspace will remove last digit typed
extern unsigned long SCI1_InULDec(void);

//------------------------SCI1_InSDec-----------------------------------
// InSDec accepts ASCII input in signed decimal format
//     and converts to a 16 bit signed number
//     valid range is -32768 to +32767
// Input: none
// Output: 16-bit signed number
// If you enter a number outside +/-32767, it will truncate
// without an error
// Backspace will remove last digit typed
extern signed int SCI1_InSDec(void);

//------------------------SCI1_InSLDec-----------------------------------
// InSLDec accepts ASCII input in signed decimal format
//     and converts to a 32 bit signed number
//     valid range is -2,147,483,648 to +2,147,483,647
// Input: none
// Output: 32-bit signed number
// If you enter a number outside +/-2147483648, it will truncate
// without an error
// Backspace will remove last digit typed
extern signed long SCI1_InSLDec(void);

//------------------------SCI1_InUHex-----------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 16-bit unsigned number
// Just enter the 1 to 4 hex digits
// It will convert lower case a-f to uppercase A-F
//     and converts to a 16 bit unsigned number
//     value range is 0 to FFFF
```

```
// If you enter a number above FFFF, it will truncate
// without an error
// Backspace will remove last digit typed
extern unsigned short SCI1_InUHex(void);

//-------------------------SCI1_OutStatus-----------------------------
// Checks if output data buffer is empty, TRUE if empty
// Input: none
// Output: TRUE if a call to OutChar will output and return right away
//         FALSE if a call to OutChar will wait for output to be ready
extern char SCI1_OutStatus(void);

//-----------------------------SCI1_OutChar---------------------------
// Wait for buffer to be empty, output 8-bit to serial port
// busy-waiting synchronization
// Input: 8-bit data to be transferred
// Output: none
extern void SCI1_OutChar(char);

//-------------------------SCI1_OutUDec-------------------------
// Output a 16-bit number in unsigned decimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1-5 digits with no space before or after
extern void SCI1_OutUDec(unsigned short);

//---------------------------SCI1_OutString---------------------------
// Output String (NULL termination), busy-waiting synchronization
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
extern void SCI1_OutString(char *pt);

//---------------------------SCI1_OutUHex----------------------------
// Output a 16 bit number in unsigned hexadecimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1 to 4 digits with no space before or after
extern void SCI1_OutUHex(unsigned short);
```

```
////////////////////////////sci1.c////////////////////////////////

/* ****************************  sci1.c ******************************
**
** This module implements interrupt driven background communications
** using SCI1; a single interrupt service routine is used to service
** both incoming as well as outgoing data streams.
**
** fw-02-05
*/


#include <mc9s12dp256.h>        /* derivative information */
#include <string.h>             /* strlen() */
#include "sci1.h"
#include "pll.h"                /* macro _SYSCLOCK */
#include "rb_.h"                /* ring buffer macros */


#define      MAX_BUFLEN 128
static char  outbuf[2*MAX_BUFLEN];  /* memory for ring buffer #1 (TXD) */
static char  inbuf [2*MAX_BUFLEN];  /* memory for ring buffer #2 (RXD) */

/* define o/p and i/p ring buffer control structures */
/* static struct { ... } out; -> global to this file */
static RB_CREATE(out, char);
/* static struct { ... } in;  -> global to this file */
static RB_CREATE(in,  char);


/*
** ---------------------------------------------------------------------
** interrupt handler
** ---------------------------------------------------------------------
*/

#define RDRF 0x20   // Receive Data Register Full Bit
#define TDRE 0x80   // Transmit Data Register Empty Bit

__interrupt void SCI1_isr(void) {

  /* determine cause of interrupt */
  if((SCI1SR1 & RDRF) != 0) {

    /* Receive Data Register Full -> fetch character and store */
    if(!RB_FULL(&in)) {

    /* store the value of SCI1DRL in the ring buffer */
       *RB_PUSHSLOT(&in) = SCI1DRL;
       RB_PUSHADVANCE(&in);      /* next write location */
    }

    PORTB ^= 0x01;

  } else if((SCI1SR1 & TDRE) != 0) {

    /* Transmission Data Register Empty -> send... */
```

```
    if(!RB_EMPTY(&out)) {

    /* start transmission of next character */
       SCI1DRL = *RB_POPSLOT(&out);
    /* remove the sent character from the ring buffer */
       RB_POPADVANCE(&out);

    } else {

      /* buffer empty -> disable TX interrupt */
       SCI1CR2 &= ~0x80;
      /* ... otherwise the system 'hangs' (continous interrupts) */
    }

  }

} /* SCI1_isr */



/*
** ------------------------------------------------------------------------
** communication interface
** ------------------------------------------------------------------------
*/

/* O/P : send single character */
//------------------------------SCI1_OutChar------------------------------
// Wait for buffer to be empty, output 8-bit to serial port
// busy-waiting synchronization
// Input: 8-bit data to be transferred
// Output: none
void SCI1_OutChar(char data) {

  /* wait until there's space in the ring buffer */
  while(RB_FULL(&out));
  /* place character to be sent in the buffer */
  *RB_PUSHSLOT(&out) = data;
  /* set write position for the next character to be sent */
  RB_PUSHADVANCE(&out);

  SCI1CR2 |= 0x80;      /* (re-)enable interrupt  */

} /* SCI1_OutChar */


/* O/P : send entire string */
void SCI1_OutString(char *pt) {

  while(*pt) {

    SCI1_OutChar(*pt);
    pt++;

  }

} /* SCI1_OutString */
```

```
/* I/P : get single character */
char SCI1_InChar(void) {

char c;

  /* wait until there's data in the ring buffer */
  while(RB_EMPTY(&in));
    /* get character off the buffer */
  c = *RB_POPSLOT(&in);
  /* set write position to the next free slot */
  RB_POPADVANCE(&in);

  return c;

} /* SCI1_InChar */



//------------------------------SCI1_Init------------------------------
// Initialize Serial port SCI1
// Input: baudRate is tha baud rate in bits/sec
// Output: none
void SCI1_Init(unsigned short baudRate) {

  /* set-up input and output ring buffers */
  RB_INIT(&out, outbuf, 255);      /* set up TX ring buffer */
  RB_INIT(&in, inbuf, 255);        /* set up RX ring buffer */


  /* check if bus frequency has been boosted to 24 MHz (fw-07-04) */
  #if _BUSCLOCK == 24

  /* 24 MHz bus frequency (PLL is used, SYNR = 2, REFDV = 0 -> factor 6)
     Baud rate generator:
      SCI1BDL/H = (24e6/16)/baudrate = 1.5e6/baudrate */
  switch(baudRate){
    case BAUD_300:
      SCI1BDH=19;
      SCI1BDL=136;
      break;
    case BAUD_600:
      SCI1BDH=9;
      SCI1BDL=196;
      break;
    case BAUD_1200:
      SCI1BDH=4;
      SCI1BDL=226;
      break;
    case BAUD_2400:
      SCI1BDH=2;
      SCI1BDL=113;
      break;
    case BAUD_4800:
      SCI1BDH=1;
      SCI1BDL=56;
      break;
    case BAUD_9600:
```

```
    SCI1BDH=0;
    SCI1BDL=156;
    break;
  case BAUD_19200:
    SCI1BDH=0;
    SCI1BDL=78;
    break;
  case BAUD_38400:
    SCI1BDH=0;
    SCI1BDL=39;
    break;
  case BAUD_57600:
    SCI1BDH=0;
    SCI1BDL=26;
    break;
  case BAUD_115200:
    SCI1BDH=0;
    SCI1BDL=13;
    break;
}

#else

/* 4 MHz bus frequency (PLL not used, SYNR = REFDV = 0 -> factor 2)
   Baud rate generator:
   SCI1BDL/H = (4e6/16)/baudrate = 250000/baudrate */
switch(baudRate){
  case BAUD_300:
    SCI1BDH=3;
    SCI1BDL=64;
    break;
  case BAUD_600:
    SCI1BDH=1;
    SCI1BDL=160;
    break;
  case BAUD_1200:
    SCI1BDH=0;
    SCI1BDL=208;
    break;
  case BAUD_2400:
    SCI1BDH=0;
    SCI1BDL=104;
    break;
  case BAUD_4800:
    SCI1BDH=0;
    SCI1BDL=52;
    break;
  case BAUD_9600:
    SCI1BDH=0;
    SCI1BDL=26;
    break;
  case BAUD_19200:
    SCI1BDH=0;
    SCI1BDL=13;
    break;
}

#endif /* _BUSCLOCK */
```

```
  SCI1CR1 = 0;
  /* bit value meaning
    7   0    LOOPS, no looping, normal
    6   0    WOMS, normal high/low outputs
    5   0    RSRC, not appliable with LOOPS=0
    4   0    M, 1 start, 8 data, 1 stop
    3   0    WAKE, wake by idle (not applicable)
    2   0    ILT, short idle time (not applicable)
    1   0    PE, no parity
    0   0    PT, parity type (not applicable with PE=0) */

  SCI1CR2 = 0xAC;      /* enable both RX and TX interrupts */
  /* bit value meaning
    7   0    TIE, transmit interrupts on TDRE
    6   0    TCIE, no transmit interrupts on TC
    5   1    RIE, receive interrupts on RDRF
    4   0    ILIE, no interrupts on idle
    3   1    TE, enable transmitter
    2   1    RE, enable receiver
    1   0    RWU, no receiver wakeup
    0   0    SBK, no send break */

}


//------------------------------SCI1_InStatus-------------------------------
// Checks if new input is ready, TRUE if new input is ready
// Input: none
// Output: TRUE if a call to InChar will return right away with data
//         FALSE if a call to InChar will wait for input

char SCI1_InStatus(void) {

  return(SCI1SR1 & RDRF);

}

//-------------------------SCI1_OutStatus-----------------------------------
// Checks if output data buffer is empty, TRUE if empty
// Input: none
// Output: TRUE if a call to OutChar will output and return right away
//         FALSE if a call to OutChar will wait for output to be ready
char SCI1_OutStatus(void) {

  return(SCI1SR1 & TDRE);

}


//-------------------------SCI1_InString------------------------------
// This function accepts ASCII characters from the serial port
//    and adds them to a string until a carriage return is inputted
//    or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//    and the backspace is echoed
// InString terminates the string with a null character
```

```
// -- Modified by Agustinus Darmawan + Mingjie Qiu --
void SCI1_InString(char *string, unsigned short max) {
int length=0;
char character;
  character = SCI1_InChar();
  while(character!=CR){
    if(character==BS){
      if(length){
        string--;
        length--;
        SCI1_OutChar(BS);
      }
    }
    else if(length<max){
      *string++=character;
      length++;
      SCI1_OutChar(character);
    }
    character = SCI1_InChar();
  }
  *string = 0;
}


//#ifdef ERASE

//-------------------------SCI1_InUDec-----------------------------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 16 bit unsigned number
//     valid range is 0 to 65535
// Input: none
// Output: 16-bit unsigned number
// If you enter a number above 65535, it will truncate
// without an error
// Backspace will remove last digit typed
unsigned short SCI1_InUDec(void) {

unsigned short number=0, length=0;
char character;

  character = SCI1_InChar();

  while(character!=CR) {

    // accepts until carriage return input
    // The next line checks that the input is a digit, 0-9.
    // If the character is not 0-9, it is ignored and not echoed
    if((character>='0') && (character<='9')) {
      // this line overflows if above 65535
      number = 10*number+(character-'0');
      length++;
      SCI1_OutChar(character);
    }

// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
    else if((character==BS) && length) {
```

```
      number /= 10;
      length--;
      SCI1_OutChar(character);

    }

    character = SCI1_InChar();

  }

  return number;

}

//------------------------SCI1_InULDec-----------------------------------
// InULDec accepts ASCII input in unsigned decimal format
//     and converts to a 32 bit unsigned number
//     valid range is 0 to 4,294,967,296
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967296, it will truncate
// without an error
// Backspace will remove last digit typed
unsigned long SCI1_InULDec(void) {

unsigned long number=0, length=0;
char character;

  character = SCI1_InChar();

  while(character!=CR) {

    // accepts until carriage return input
    // The next line checks that the input is a digit, 0-9.
    // If the character is not 0-9, it is ignored and not echoed
    if((character>='0') && (character<='9')) {
      // this line overflows if above 4294967296
      number = 10*number+(character-'0');
      length++;
      SCI1_OutChar(character);
    }

// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
    else if((character==BS) && length) {

      number /= 10;
      length--;
      SCI1_OutChar(character);

    }

    character = SCI1_InChar();

  }

  return number;
```

```
}


//------------------------SCI1_InSDec-------------------------------------
// InSDec accepts ASCII input in signed decimal format
//     and converts to a 16 bit signed number
//     valid range is -32768 to +32767
// Input: none
// Output: 16-bit signed number
// If you enter a number outside +/-32767, it will truncate
// without an error
// Backspace will remove last digit typed
signed int SCI1_InSDec(void) {

signed int number=0, length=0;
char sign = 0;  // '0': pos, '1': neg
char character;

  character = SCI1_InChar();

  while(character!=CR) {

    // accepts until carriage return input
    // The next lines checks for an optional sign character ('+' or '-')
    // and then that the input is a digit, 0-9.
    // If the character is not 0-9, it is ignored and not echoed
    if(character=='+') {
      SCI1_OutChar(character);
    } else if(character=='-') {
      sign = 1;
      SCI1_OutChar(character);
    } else if((character≥'0') && (character≤'9')) {
      // this line overflows if above 4294967296
      number = 10*number+(character-'0');
      length++;
      SCI1_OutChar(character);
    }

    // If the input is a backspace, then the return number is
    // changed and a backspace is outputted to the screen
    else if((character==BS) && length) {

      number /= 10;
      length--;
      SCI1_OutChar(character);

    }

    character = SCI1_InChar();

  }

  if(sign == 1) return -number;
  else          return  number;

}
```

```
//------------------------SCI1_InSLDec-----------------------------------
// InSLDec accepts ASCII input in signed decimal format
//     and converts to a 32 bit signed number
//     valid range is -2,147,483,648 to +2,147,483,647
// Input: none
// Output: 32-bit signed number
// If you enter a number outside +/-2147483648, it will truncate
// without an error
// Backspace will remove last digit typed
signed long SCI1_InSLDec(void) {

signed long number=0, length=0;
char sign = 0;  // '0': pos, '1': neg
char character;

  character = SCI1_InChar();

  while(character!=CR) {

    // accepts until carriage return input
    // The next lines checks for an optional sign character
    // ('+' or '-')
    // and then that the input is a digit, 0-9.
    // If the character is not 0-9, it is ignored and not echoed
    if(character=='+') {
      SCI1_OutChar(character);
    } else if(character=='-') {
      sign = 1;
      SCI1_OutChar(character);
    } else if((character>='0') && (character<='9')) {
      // this line overflows if above 4294967296
      number = 10*number+(character-'0');
      length++;
      SCI1_OutChar(character);
    }

    // If the input is a backspace, then the return number is
    // changed and a backspace is outputted to the screen
    else if((character==BS) && length) {

      number /= 10;
      length--;
      SCI1_OutChar(character);

    }

    character = SCI1_InChar();

  }

  if(sign == 1) return -number;
  else          return  number;

}

//------------------------SCI1_OutUDec-----------------------------
// Output a 16-bit number in unsigned decimal format
```

```
// Input: 16–bit number to be transferred
// Output: none
// Variable format 1–5 digits with no space before or after

void SCI1_OutUDec(unsigned short n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n ≥ 10){
    SCI1_OutUDec(n/10);
    n = n%10;
  }
  SCI1_OutChar(n+'0'); /* n is between 0 and 9 */
}

//———————————————————SCI1_InUHex———————————————————————
// Accepts ASCII input in unsigned hexadecimal
// (base 16) format
// Input: none
// Output: 16–bit unsigned number
// Just enter the 1 to 4 hex digits
// It will convert lower case a–f to uppercase A–F
//     and converts to a 16 bit unsigned number
//     value range is 0 to FFFF
// If you enter a number above FFFF, it will truncate
// without an error
// Backspace will remove last digit typed

unsigned short SCI1_InUHex(void){
unsigned short number=0, digit, length=0;
char character;
  character = SCI1_InChar();
  while(character!=CR){
    digit = 0x10; // assume bad
    if((character≥'0') && (character≤'9')){
      digit = character–'0';
    }
    else if((character≥'A') && (character≤'F')){
      digit = (character–'A')+0xA;
    }
    else if((character≥'a') && (character≤'f')){
      digit = (character–'a')+0xA;
    }
// If the character is not 0–9 or A–F,
// it is ignored and not echoed
    if(digit≤0xF ){
      number = number*0x10+digit;
      length++;
      SCI1_OutChar(character);
    }
// Backspace outputted and return value changed
// if a backspace is inputted
    else if(character==BS && length){
      number /=0x10;
      length––;
      SCI1_OutChar(character);
    }
    character = SCI1_InChar();
  }
```

```
  return number;
}

//------------------------------SCI1_OutUHex----------------------
// Output a 16 bit number in unsigned hexadecimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1 to 4 digits with no space before or after

void SCI1_OutUHex(unsigned short number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
  if(number≥0x10)   {
    SCI1_OutUHex(number/0x10);
    SCI1_OutUHex(number%0x10);
  }
  else if(number<0xA){
    SCI1_OutChar(number+'0');
  }
  else{
    SCI1_OutChar((number-0x0A)+'A');
  }
}

//#endif /* ERASE */
```

VITA

Yasmine Ahmed El-Ashi was born on December 31st, 1984, in Khartoum, Sudan. She completed her IGCSE O-level and A-level examinations in Unity High School, a missionary private secondary school in Khartoum, in 2001. She earned a Bachelor of Science degree in Electrical Engineering with a MagnaCumlaude honor (3.8 GPA) from the American University of Sharjah, in 2006. In addition, she completed a minor in Applied and Computational Mathematics.

She enrolled in the Mechatronics Masters program in the American University of Sharjah as a graduate assistant, in 2007. Furthermore, she worked on a project funded by AUS Research Grant on Modeling and Analysis of a Wavelet Network Based Optical Sensor for Vibration Monitoring.

Published conference papers:

- Y. El-Ashi, R. Dhaouadi, and T. Landolsi, Design of a Novel Optical Vibrometer Using Gaussian Beam Analysis, Proc. of 5th International Symposium on Mechatronics and its Applications (ISMA08), Amman, Jordan, May 2008.
- Y. El-Ashi, R. Dhaouadi, and T. Landolsi, Accuracy of a Gaussian Beam Optical Vibrometer with a Quad Photodetector Spatial Separation, Proc. of 3rd International Conf. on Modeling,Simulation and Applied Optimization, Sharjah, UAE, January 2009.

Published journal paper:

- Y. El-Ashi, R. Dhaouadi, and T. Landolsi, Position Detection and Vibration Monitoring System Using Quad-cell Optical Beam Power Distribution, Journal of the Franklin Institute, April 2010.

Submitted journal paper:

- Y. El-Ashi, R. Dhaouadi, and T. Landolsi, Modeling and Analysis of a Wavelet Network Based Optical Sensor for Vibration Monitoring, IEEE Transactions on Sensors, April 2010.