Position and Energy Reconstruction from Scintillation Light in a Liquid Xenon Gamma Ray Detector designed for PET

by

Andrew Gordon Wilson

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Science

in

The Faculty of Science

(Honours Physics and Mathematics)

The University Of British Columbia

(Vancouver, Canada)

April, 2008

Abstract

Positron Emission Tomography (PET) is used to produce a three-dimensional map of functional processes in the body. In this thesis, I present techniques to localize the positions and energies of particles in a liquid xenon gamma ray detector designed for a promising new PET device. These techniques were developed strictly through modelling the isotropic scintillation light emission that follows when photons enter a liquid xenon chamber and interact with electrons at particular locations.

Specifically, given an arrangement of photo-detectors in a liquid xenon chamber, I used Geant3 Monte-Carlo simulations to generate data at photo-detectors when a specified number of photons is emitted in random directions from a specified point in the chamber. I developed reconstruction algorithms which take such photo-detector data as input, and return the energy and position of a particle in the chamber that could generate this data.

In order to estimate the energy of a particle, I compared photo-detector data corresponding to particles at the same positions but with different energies. For position reconstruction, I used *K Nearest Neighbour* and *Neural Network* algorithms. The neural network model presented is more successful, and was particularly developed and tested for regions of the detection chamber which are most critical for position reconstruction. I present statistics for how well this network reconstructs input from particles with energies ranging from 300 keV to 511 keV, with and without typical Avalanche Photo-diode (APD) noise added to the photo-detectors. Notably, the network provides position reconstruction to within a volume of 1 cm^3, 90% of the time, for typical photo-detector input with no added noise – that is, photo-detector data generated from a representative sample of particles with 511 keV of energy. This is close to a predicted minimum error volume of 0.9 cm^3 for such input.

The work in this thesis can be used in combination with position and energy measurements from charge collection, to produce a relatively inexpensive liquid xenon PET device with three to four times better spatial and energy resolution than the best conventional devices. This allows for great advances in medical research and diagnosis.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I am thankful to my supervisor, Douglas Bryman, for the opportunity to work on such a consequential project, which combines a remarkable selection of the specific interests I developed throughout my undergraduate years. I also thank him and his group for expertise and support that have been of great help in preparing this thesis.

Further, I thank Piotr Kozlowski and Alex Mackay for the kind introduction to their very interesting work in medical physics, which I hope to learn more about in the future. Similarly, thanks to the machine learning group in the computer science department at UBC for useful and interesting conversations.

I'm also thankful to Matthew Choptuik, who helped me develop skills that made much of the work in this thesis possible.

And thanks to my family and friends for their continued interest and support.

Dedication

For Laska.

Chapter 1

Introduction

1.1 Motivation

Positron Emission Tomography (PET) is an extremely important medical imaging technique. It is used to produce a three-dimensional map of functional (metabolic) processes in the body. It can also be combined with anatomical information provided by Magnetic Resonance Imaging (MRI), and Computed Tomography (CT). As a result, it has greatly advanced our understanding of functional processes, and has been used to locate, diagnose, monitor, treat, and study, cancers, metastasis, and cardiovascular disease, as well as neurological, psychiatric, and other critical medical conditions, where metabolic activity is of interest, and where no other medical imaging device would be of great use [1, 2]. Notably, as a consequence of metabolic information uniquely available through PET, brain and heart functions have been localized, and the intake and concentrations of drugs in our system as they are metabolized have been traced [2, 3]. Indeed, our understanding of functional physiological processes, the quality of health care in the future, and the prognosis of a substantial number of patients in critical conditions, will greatly depend on the continued implementation and development of this particular modality of medical imaging.

In response to the demand for improving PET, we are developing a new high resolution PET device, using liquid xenon as a detection medium. For image generation, this new device depends on reconstructing the position and energy of a particle from the scintillation light it emits. This new system, XEPET, has the potential to revolutionize functional imaging, supposing that the work on these reconstruction techniques is properly carried through; these reconstruction techniques are the primary subject of this thesis, and their potential when integrated with XEPET is the primary motivation for their development.

In order to conduct a PET scan, a radioactive tracer isotope, which decays by emitting a positron, is incorporated with a metabolically active molecule (e.g. a common sugar), and then usually injected into the bloodstream of a living subject. It follows that this tracer will accumulate and release positrons in metabolically active tissues, such as the brain. These positrons travel up to a few millimeters before encountering and annihilating with an electron, generating a pair of annihilation (gamma) photons, moving in almost opposite directions. In most cases, each photon in this pair will near-simultaneously hit scintillation material on opposite sides of the subject, generating a flash of light that can be detected by photomultiplier tubes or Silicon Avalanche Photodiodes (APDs). This annihilation event can then be localized along a line of response (LOR), which connects the region where one photon in the pair is detected with the opposite region where the other photon is detected. The places of greatest metabolic activity can then be found, in essence, by locating the regions with the greatest number of intersections between lines of response.

A PET detector can hypothetically be improved to the point where it is primarily limited

by the travel distance of a positron before annihilating with an electron (usually less than 1 mm), and the fact that the generated photons are not precisely collinear, since the positron and electron are not precisely at rest when they annihilate, and momentum must be conserved [1, 4, 5]. Therefore an ideal PET device would minimize all other notable limiting factors – it would have high sensitivity (fraction of annihilation events detected), high 3-D spatial resolution, low cost, low dead-time, good timing resolution, and good energy resolution.

Good energy resolution (percent error on energy detected) can be used to suppress data from photons that scatter within a subject, since any photon that does scatter in this way will have less than the 511 keV initially imparted to it from a positron-electron annihilation event. This preserves image resolution, since when a pair of gamma photons is generated, and one or both photons scatter within the subject, they do not travel in near opposite directions, and thus contribute poorly to data used in constructing lines of response, particularly since 30-50% of the annihilation events scatter in this way [1]; the probability that a photon will scatter within the patient depends on the thickness of bone and tissue it must traverse through. Energy resolution also helps suppress data from the rare annihilation events which do not produce two photons.

Good spatial resolution is also desired, since one would know more precisely where the gamma photons are detected when constructing lines of response. Moreover, with better time resolution one would better be able to determine which photons correspond to which annihilation events, reducing the corruption of data from "random coincidences". Further, the higher the (scintillation) efficiency, the greater the light output of the scintillation material, and so the better the energy resolution and sensitivity. And low dead-time allows for more data to be collected, ultimately resulting in more precise images. Dead-time and timing resolution are related to the decay time and coincidence time window of scintillation material used in a PET device; decay time is proportional to detector dead-time, since one must wait this long to detect an annihilation photon, and the coincidence time window is the error on the detection time of an annihilation photon. Figure 1.1 illustrates random and scattered coincidence in comparison to true coincidence.

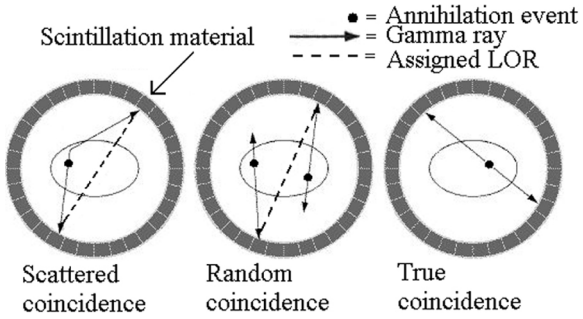

Figure 1.1: **Scattered, Random, and True Coincidence**. If two photons, generated from the same annihilation event within the subject, travel along a straight line path, and both interact with scintillation material almost at the same time, we have true coincidence and can assign an accurate LOR. Scattered coincidence occurs when one or both photons scatter within a subject, and so the assigned LOR will no longer be the same as the trajectories of these photons. Random coincidence occurs when photons from different annihilation events interact with scintillation material at nearly the same time. The assigned LOR for random coincidence in conventional devices is illustrated. Figure adapted from [6].

	XEPET Potential	LSO PET Devices
Scintillator decay time	3 ns (98%), 27 ns (2%) [10]	40 ns [10]
Scintillator cost[1]	US $3 per cc	US $50 per cc
Scintillator efficiency	$7.8 \cdot 10^4$ photons/MeV [10][2]	$3.2 \cdot 10^4$ photons/MeV [10]
Energy resolution[3]	5% (FWHM)* [9]	12-20% (FWHM) [11]
Spatial resolution	0.8 mm* [9]	2.5-4.0 mm [4]
Coincidence time window	3 ns [12]	6.0 ns [11]
Sensitivity	12%* [21]	3% [11]

Table 1.1: Comparison between XEPET and best modern PET devices. *Provided the position reconstruction of a particle from its scintillation light is successful. [1] Market prices as of 2007. [2] Values are known to vary by 10^4 photons/MeV in the literature. [3] at 511 keV. Unless otherwise specified, uncertainties are taken to be the last unlisted significant digit.

The most effective modern PET devices use segmented inorganic Lutetium Orthosilicate (LSO) crystals as scintillation material [7]. The primary difference between XEPET and these conventional devices is that XEPET uses liquid xenon as scintillation material. This choice of scintillation material allows XEPET to outperform all current devices, and likely all future device that use inorganic cystals. XEPET is free of the parallax errors that plague crystal devices [3], and has an energy resolution 10% ± 3% (FWHM) better and a spatial resolution 1 mm ± 0.2 mm less than crystal based PET devices are capable of having due to their intrinsic limitations [7]. Liquid xenon is also excellent for transporting charge, allowing for charge energy to be easily measured, resulting in higher energy resolution [9]. XEPET is also special in that it measures energy from both scintillation light, and ionized charge, which allows it to provide better energy resolution than if it were to measure energy from only one or the other [21]; the meaning of this will soon become clear. Further, liquid xenon can fill large detection volumes, resulting in the detection of a significantly larger fraction of annihilation photons than in conventional devices. And it is affordable, both to purchase, and maintain in its liquid state, since the melting point of xenon gas is 161 K, while the temperature of common liquid nitrogen is 63 K. These, and other advantages of XEPET, are quantitatively compared with the best modern devices in Table 1.1. Figure 1.2 illustrates the potential difference between the best modern PET device, microPET, in comparison to XEPET, provided position and energy reconstruction from scintillation light is successful.

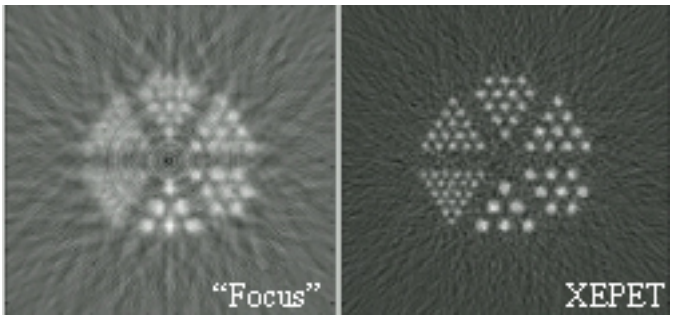

Figure 1.2: Simulated responses of a state of the art "Focus" Micro-PET (left), and XEPET, to a "Derenzo" water phantom. The smallest feature size, which has a 1 mm diameter, is clearly visible in the XEPET simulation. Figure taken from [21].

In order to conduct a scan with XEPET, a subject emitting annihilation photons is placed within a cylinder made from trapezoidal chambers filled with liquid xenon (figure 1.3). When a photon enters one of these chambers, it will interact with electrons and be photo-absorbed, or it will Compton scatter; in either case, the ionized electrons will be brought to the top of the chamber by an electric field, where their energies can be measured: using liquid xenon as scintillation material makes this possible. When these electrons are ionized, a flash of scintillation light is emitted. As noted in Table 1.1, localizing the position of where this electron is ionized from this scintillation light is essential to the operation of XEPET in many regards. In the next section, it will become clearer why localizing a particle from scintillation light is necessary, and why the electric field is particularly useful; first, some necessary background must be introduced. Essentially, this position reconstruction is necessary for generating accurate lines of response in situations where there is Compton scattering, or multiple photo-absorption events in one chamber. These events account for a substantial proportion of the data we will process, since it takes 40 microseconds for an electron to drift from one end of the chamber to the other, and we expect incoming gamma rays at a rate of 1 MHz, 78% of which will Compton scatter [21].

Finally, XEPET is expected to have a significant impact for the following reasons: the increased sensitivity could be used to lower the radiation dose associated with injecting a subject with a radioactive tracer – this is highly useful in research when needing to do repeated scans; this increased sensitivity would also allow the same dose to be used in clinical studies but with a shortened scan-time – increasing patient throughput, while reducing cost and waiting times; the improved energy resolution of XEPET would make the scattering of photons within the human brain, which is substantial due to the brain size, much less of an issue; the higher resolution of XEPET promises to make sources of motion, such as patient breathing, more easily detectable; blood flow studies can be used with tracers such as ^{15}O and ^{13}N, which have short half lives; investigations of regions with low signal such as extrastriatal regions, for studying neurodegenerative and psychiatric diseases are made possible; and investigations using new tracers with short half-lives are also made possible.

Further, the techniques employed in localizing a particle based on the light it emits will not only be immediately essential to the operation of XEPET, but could be used in detection systems in astronomy, particle physics, or other fields [13, 14].

In this section we have motivated the need for position reconstruction from scintillation light

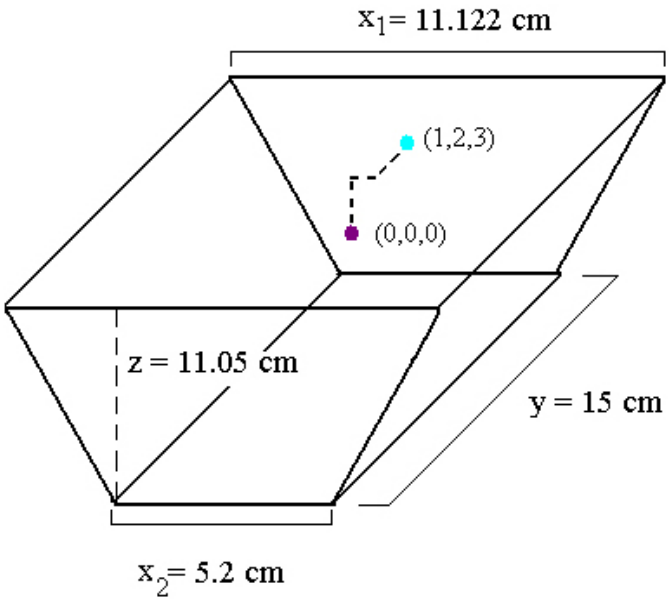

Figure 1.3: **XEPET Detection Chamber**. These chambers fit together to form a cylinder around the subject. 16 APDs are on each of the trapezoidal faces ($y = constant$), and an applied constant electric field runs from one $z = constant$ face to the other $z = constant$ face. Points are illustrated at the centre of the chamber (0,0,0), and at (1,2,3), in part to make use of a coordinate system that will be referred to throughout this thesis.

by assuming that it enables XEPET, and then demonstrating the importance of XEPET. In the next section, we assume XEPET is important, and motivate position reconstruction by demonstrating that it enables XEPET. In order to do this, we introduce necessary background theory, which also partly serves to clarify and re-iterate many of the explanations already presented.

1.2 Reconstruction from Scintillation Light

1.2.1 Gamma Ray Interaction with XEPET Chamber

If a photon interacts with an electron in a material, and its energy is greater than the binding energies of an electron in that material (the work function), then the electron may completely absorb the photon's energy and be freed from the material, or it may only absorb a portion of the photon's energy, in which case the photon will Compton scatter and travel off of at a deflection angle, θ_c. Photo-absorption is described by

$$h\nu = W + K \tag{1.1}$$

where h is Planck's constant, ν is the frequency of the incoming photon, W is the work function of the material, and K is the kinetic energy of the electron after this collision, while Compton scattering is described by the Klein Nishina formula,

$$cos(\theta_c) = 1 - \frac{E_1 mc^2}{E(E - E_1)} \tag{1.2}$$

where m is the mass of the electron, c is the speed of light, E_1 is the energy deposited in the first interaction and E is the energy of the incident photon.

Photons entering a XEPET chamber have several hundred keVs of energy (ideally 511 keV), well in excess of the binding energy of an electron in liquid xenon (less than 40 eV). Therefore when a photon enters a XEPET chamber, it will travel until it interacts with an electron, where it will either be photo-absorbed, or Compton scatter.

Photons entering a XEPET chamber have several hundred keVs of energy (ideally 511 keV), well in excess of the binding energy of an electron in liquid xenon (less than 40 eV). Therefore when a photon enters a XEPET chamber, it will travel until it interacts with an electron, where it will either be photo-absorbed, or Compton scatter. In either case, when a photon interacts with an electron in liquid xenon, it will emit a flash of scintillation light: $6.0 \cdot 10^4$ to $7.8 \cdot 10^4$ photons will be emitted per MeV of energy deposited at the electron. In essence, scintillation light follows when xenon atoms are either excited directly, or through the ionization of an electron: these excited atoms interact with neighbouring xenon atoms, which can also become excited, and subsequently lose this excitation energy by emitting scintillation light. Figures 1.4 and 1.5 respectively illustrate Compton scattering and photo-absorption in a XEPET chamber. A gamma ray entering a XEPET chamber has a 22%, 36%, 19%, 7%, and 6% chance of Compton scattering respectively 0, 1, 2, 3 or greater than 4 times [21]. This overall description, and figures 1.4 and 1.5, have been simplified in the sense that a localized cloud of electrons (several thousand) drift to the top of the chamber, rather than just the initially excited electron, which would actually travel erratically for approximately 1 mm, exciting xenon atoms in the process.

The flash of scintillation light emitted after photo-absorption or Compton scattering can be used to time when these events happen relative to when they are detected. After recording a flash of scintillation light, the timer is set zero. The 1-3 kV/cm electric field in a XEPET chamber will pull the freed electrons up to the top of the chamber, where their energies, x and y positions (fig 1.3), and times of arrival are measured. Knowing this electric strength, and the time it takes an electron to reach the top of the chamber from when it interacted with a photon, one can determine the z coordinate, and therefore reconstruct the full position of where the initial photon-electron interaction took place, and then assign an accurate line of response. However, in order to do this, we must associate each flash of scintillation light with the correct gamma-ray electron interactions. There are many cases in which this cannot be done without first localizing these interactions from scintillation light alone. These situations are illustrated and described in figures 1.4 and 1.5. In short, *one most localize gamma-ray electron interactions from scintillation light in order to assign accurate lines of response.* This is the specific motivation for the work in this thesis.

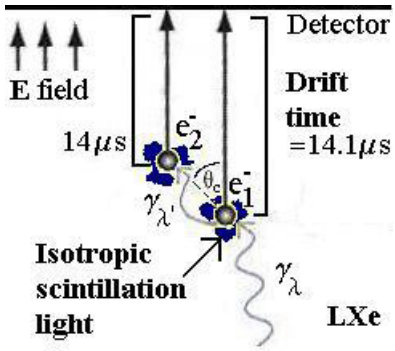

Figure 1.4: **One Compton Scattering Event in a XEPET chamber**. An incident photon with wavelength λ interacts with an electron. After depositing some of its energy at an electron, it leaves at an angle and with wavelength $\lambda' > \lambda$, where it is absorbed nearby. At each interaction a flash of scintillation light is emitted. However, without knowing where the electrons are after these flashes, it is impossible to determine which electron was excited first: illustrated above, the flash at e_1 is first, and e_2 arrives first, but the situation where the flash at e_2 is second, and e_2 arrives second, is also possible.

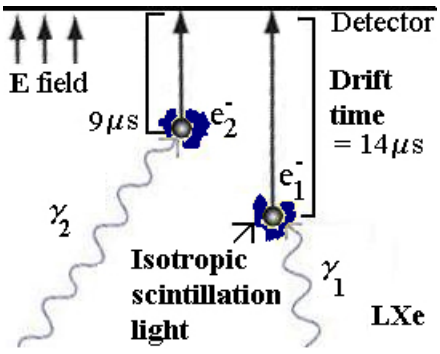

Figure 1.5: **Random Coincidence in a XEPET Chamber**. Two gamma rays are photo-absorbed within a time less than the time it takes an electron to drift across the whole chamber (40 μs). Localizing the positions of e_1 and e_2 from scintillation light allows us to determine which electron was excited by which gamma ray.

1.2.2 Position Reconstruction

On any given scintillation flash in a XEPET chamber, each photo-detector will sense a certain number of photons. Given data from the photo-detectors alone, *we wish to approximately determine the location of a point in a XEPET chamber that could emit a number of photons in random directions to generate such data*. The statistics in table 1.1 are reasonable if these points can be localized to within a sphere with a radius of approximately 1 to 2 cm [21].

In order to do this, it would be useful to know the answer to the following problem: we have a circular detector of radius R a distance D from a point emitting a sphere of light, and we wish to determine what fraction of this light will be incident on the detector. However, there is no known closed-form solution to this problem, and open form solutions can only be evaluated with computations so lengthy that they are not suitable for incorporation into models that require calculations for different positions of the detector [15].

But one can specify a number of photons being emitted at a point in a XEPET chamber, and then use Monte Carlo simulations to determine how many of these photons will hit each photo-detector. It turns out that for this application this is at least as useful as an analytical solution for this problem; after all, this is not a problem with a deterministic solution – a number of photons is emitted in *random* directions – and so if the number of photons is not exceedingly large, a pseudo-random computer model will need to be used for an accurate solution, regardless of the specific approach. In this case, the number of photons reaching photo-detectors is small enough that using a probabilistic model has practical advantages.

1.2.3 Energy Reconstruction

Energy reconstruction from scintillation light is explored in this thesis; indeed, it is included in the very title. But so far the discussion has focused more explicitly on the need for position reconstruction from scintillation light. This is because position reconstruction is necessary for energy reconstruction. And for accurate position reconstruction, we must first approximate particle positions from scintillation light.

The amount of scintillation light seen depends on how much energy an annihilation photon imparts to electrons in liquid xenon when it enters a XEPET chamber. So if we can reconstruct the position of particles emitting scintillation light, then we can simulate how much light would be seen by photo-detectors at these positions for particles with a specified energy (e.g. 511 keV). Then we can compare that photo-detector data to how much light was actually detected. This process provides a means of reconstructing the energy of particle from the scintillation light it emits.

The process of simulating how much light will reach each photo-detector after specifying a point in the chamber, and assigning an energy to that point, is a component of the process described in the section on position reconstruction: assigning an energy to a point specifies how many photons to emit in random directions from that point. Then Monte-Carlo simulations can be used to simulate how many of these photons would hit each photo-detector.

1.3 Outline of Thesis

I have now established and motivated the primary problem solved in this thesis: given only data from photo-detectors, where is the location of a point emitting a number of photons in random directions that could generate this data?

This problem can be approached using Monte-Carlo Simulations. I explain the Monte-Carlo simulations used as a first step towards solving this problem in Chapter 2. Essentially, the chamber introduced in figure 1.3 is split into a grid of points, and then Monte-Carlo simulations are used to generate examples of photo-detector data corresponding to points in the chamber.

In chapter 3, I use these examples to generate the mapping that solves the problem at hand – a mapping from photo-detector data to a point in the chamber that could generate this data. Two approaches are primarily considered – K Nearest Neighbours and Neural Networks. Some of the detail given in chapter 2 and the K Nearest Neighbour section of chapter 3 may appear tedious, but if that is the case, then it is in order for the Neural Network solution to fully be appreciated – it is unconventional, interesting (to those in a wide variety of fields), and successful. This neural network solution is not just described in chapter 3, but it is also tested in great detail: reconstruction statistics for particles with 300 keV, 450 keV, and 511 keV of energy are presented, with and without typical APD noise added to photo-detector data.

In chapter 4, I use the Monte-Carlo simulations discussed in chapter 2 to estimate the energy of a particle emitting light in liquid xenon. In doing this, I first assume perfect position reconstruction; to estimate energy resolution at a particular location, I compare the ratio of the number of photons detected from a particular point with 300 keV of energy to the number of photons detected from the same point with 511 keV of energy, to 300/511. Later in chapter 4, energy reconstruction is tested after position reconstruction from scintillation light. This is done to make some qualtitative judgements on how the error on position reconstruction will affect energy reconstruction; however, in practice this error will be significantly less, since we will also use fine position reconstruction from charge collection.

Further, chapter 5 serves to summarize my results. And appendix A provides a means to find most of the code I wrote for this thesis, while appendix B lists the file used for specifying parameters in the Monte-Carlo simulations presented in chapter 2. Overall, each chapter builds upon the previous one, even though all chapters (and sections) are independent where possible; for this reason, sometimes the same key equations or observations can be found in multiple chapters.

Finally, ever since reading George Orwell's 1946 essay, "Politics and the English Language", I've consciously avoided common writing practices that could obscure the clarity of my ideas. Since I am not bound to any convention in writing this thesis, I write it in the spirit of Orwell's essays – using language "as an instrument for expressing and not concealing or preventing thought".

My primary goal is to make my research understandable and engaging to you, the reader.

Chapter 2

Monte-Carlo Simulations

2.1 Introduction

To recap, a photon traveling along a line of response will likely interact with electrons in a XEPET chamber. At the sites of these interactions, a number of photons, depending on the amount of energy deposited at the electrons, will be emitted in random directions.

We therefore have two primary goals: to localize the position of a particle emitting light in random directions, and to estimate its energy. Succeeding with these goals allows us to assign accurate lines of response, which is necessary for producing quality PET images. Using Geant3 Monte-Carlo simulations, we can simulate the response of photo-detectors to a point emitting a specified number of photons within a XEPET chamber; in other words, the Monte-Carlo simulations serve as a mapping from points in space to photo-detector data. This information can be used to accomplish our goal of position reconstruction: we can use it to define an inverse mapping from photo-detector data to points in space. Both the inverse mapping and the forwards mapping can then be used to determine a particle's energy. First, the inverse mapping is used to localize the particle's position, which is then determined more precisely from charge collection. Then, the forwards mapping is used to generate potential photo-detector data from a particle at this position, which is scaled to determine the number of photons we would expect to detect if the particle has an energy we specify. Finally, we compare this to the number of photons we initially detected. Since the number of photons a particle emits is proportional to its energy, we can use this comparison to estimate the particle's energy.

From this explanation, it may seem that we would need to know the particle's energy in advance to do position reconstruction – after all, the number of photons we detect depends on the particle's energy. But this is not the case – we normalize the photo-detector data we use to construct the mappings, and we normalize the photo-detector data we use as input for these mappings. This way the same photo-detector data scaled by a constant factor will map to the same position in the chamber, and the value of that constant factor can be determined by scaling the forwards mapping until the scaled photo-detector data is reproduced.

In short, our two goals are connected – we use position reconstruction for energy reconstruction – and we can use Monte Carlo simulations as a step towards accomplishing both. In this chapter, I explain the Monte-Carlo simulations, and the data I generated with these simulations. Understanding the specific types of data I simulated is necessary to understand how it was used for position and energy reconstruction, in chapters 3 and 4 respectively.

2.2 The Simulations

The first step towards our goal of position and energy reconstruction is to generate examples of photo-detector data corresponding to points emitting isotropic scintillation light in the XEPET chamber; these examples can then be used to construct the mappings we need for position and energy reconstruction. Physically, the points in the chamber represent electrons, which have been excited by a photon traveling along a line of response. The amount of energy this photon deposits at a given electron will determine how many scintillation photons will be emitted.

So we wish to run a simulation in which we specify:
1) Points in the XEPET chamber
2) How much energy each point has, and therefore how many photons it will emit
3) The physical environment of the XEPET chamber

The simulation should then:
1) Choose a random direction for each photon at each specified point
2) Determine how many photons would hit each detector

These requirements are met by a Geant3 Monte-Carlo simulation I refer to as *ffcards*. A Monte-Carlo simulation is a computational algorithm with repeated random sampling. This is a very general definition, which is why the term "Monte Carlo" is used so often in different contexts. In an ffcards simulation, the direction of each photon is determined with two uniform pseudo random variables, θ and ϕ, which take values in $[0, \pi)$ and $[0, 2\pi)$, respectively. This is what makes ffcards a Monte-Carlo simulation. With the help of *Geant3* [17], a toolkit for modeling the passage of particles through matter, ffcards also simulates the effect of a liquid xenon environment on these photons as they travel towards photo-detectors – taking into account the possibility that these photons will be absorbed, scatter, or reflect from the walls of the chamber.

A large selection of parameters in an ffcards simulation can be specified in a file named `xepet.ffcards` (see appendix B). In order to generate data for position and energy reconstruction, I only use a subset of the functionality available in our simulation program (to see the full extent to which these simulations can be used, refer to the `xepet.ffcards` file). In my simulations, I vary the run number, the number of events, the photon yield per MeV, the random number seed, and the position of the particle emitting light in random directions, while keeping the other parameters as shown in the `xepet.ffcards` file in the appendix B. Parameters I choose to hold constant, but notably affect simulations, are the materials of the chamber (liquid xenon, stainless steel, white paint and black paint), the reflectivity of these materials, the number and positions of the photo-detectors (32 APDs, 16 on each of two faces), the dimensions of the chamber, and the type of particle we are placing in the chamber (an electron). The meaning of these parameters will be made clear in the example of the next section.

2.3 Sample run using ffcards

In this section I go through an example where I generate photo-detector data corresponding to the point (1,2,3) in the chamber. I include even mundane details so that this simulation could be

reproduced by the reader: the goal of this thesis is not only to present and explain the results of what was done, but to make it as easy as is practical for the reader to extend, continue, or make use of the work presented here.

An ffcards simulation uses a 3D Cartesian coordinate system, in which the origin, (0,0,0), is the centre of a XEPET chamber. This was illustrated in figure 1.3, with the dimensions of the chamber we are currently using in our design. Suppose we wish to place an electron with 511 keV of energy at position (1,2,3) in a XEPET chamber, as shown in figure 1.3. The `SORC` line in `xepet.ffcards` would read,

```
SORC -3 1.00 2.00 3.00 0.  0.  0.  0.  0.  0.  0.  0.511
```

The first argument, `-3`, specifies that the particle is an electron. The following three arguments specify the x, y, and z, position of the particle. These cannot simply be specified by the user as integers, or the simulation will not run properly; an integer coordinate should be assigned by an integer proceeded by a decimal point, with optionally trailing zeros. For instance, `4.` and `4.0` are acceptable equivalents, but `4` would not work as an input argument in the `SORC` line. Finally, the last argument of the `SORC` line specifies that the particle has 0.511 MeV of energy.

Our goal is to simulate data at the photo-detectors, so we must specify how many photons to generate per MeV of energy that the electron has. This number is determined by the scintillation yield and resolution scale of the material. The number of photons generated can be determined by a gaussian random variable with a mean equal to the product of the energy and scintillation yield, and standard deviation equal to the product of the resolution scale with the square root of the mean (see Geant3 or Geant4 reference manual for further information [17]). The scintillation yield in liquid xenon is approximately 40000 photons per MeV, and a resolution scale of 1.0 is also typical [21]. I use these values for all simulations. These values should be *representative* of what we expect from liquid xenon, but it is not important that they be extremely precise for the purpose of position reconstruction – in the end, position reconstruction should be somewhat independent of scintillation yield.

```
YILD 40000.0 1.0
```

In this case, approximately $\lfloor 0.511 * 40000 \rfloor$ photons will be sent in random directions from the point (1,2,3), and some of these photons will hit photo-detectors.

Now we specify the run number. This is to keep track of the simulation output. With a run number $j \in \mathbb{Z}$, the simulation output would be `runj.hbook`, and `runj.log` (where j is replaced by its numerical value). The following line

```
RUNG 1 0
```

sets the run number to `1`, and the event number to `0`. For our purposes, the event number will never need to be changed.

Next we specify the number of events,

TRIG 1

This means we run the simulation once. Each detector will sense an integer number of photons. However, when generating data for reconstruction, we may want to take the average of many simulations for a given run. We could take the average results of 20 events by specifying

TRIG 20

Now we run this simulation with the script `xepet.com`, which can be found in the appendix A. The file `run1.hbook` will be generated. This is intended to be interpreted by software called PAW [18], which is operated by Fortran commands. We convert this `hbook` file to a `root` file by typing

```
h2root run1.hbook
```

at the *nix command prompt. ROOT [19] is similar to PAW, except it is operated by C or C++ commands. At the ROOT command prompt, we can type

```
root run1.root
h999.Draw("pmt_energy[0]")
```

to see a histogram of the number of photons detected versus frequency for photo-detector 1 (the photo-detectors are indexed from 0 to 31). Figure 2.1 shows the results; it is exactly as presented by root, so there are no axes labels.

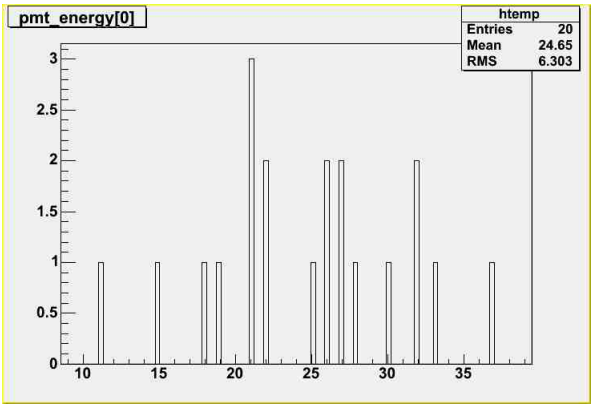

Figure 2.1: **ROOT Histogram of Data at a Photo-detector.** Histogram of the number of photons detected over 20 events at photo-detector 1, exactly as presented by ROOT. The y-axis is the frequency of the number of photons detected, and the x-axis is the number of photons detected.

Since the photons are being sent off in random directions for each simulation, we expect the results from simulation to simulation to differ. Figure 2.1 displays the results of 20 simulations. For instance, for two of the twenty events, 32 photons were detected at photo-detector one. For the purpose of reconstruction, we could take the average of these 20 results – for photo-detector 1, we could record 24.65 photons as having been detected. So we've generated possible example data we could eventually use for position or energy reconstruction:

Example Photo-detector Data
Number of events: 20
Location of point in chamber: (1.0,2.0,3.0)
Energy of particle: 0.511 MeV
Scintillation Yield: 40,000 photons per MeV

Average number of photons at each photo-detector:
Photo-detector 1: 24.65
Photo-detector 2: 41.9
...
Photo-detector 32: 39.7

2.4 Generating Simulation Data

My simulations are used to generate examples of photo-detector data that corresponds to points in the chamber that we specify. Three questions naturally arise when deciding how to generate this data:

1) How should I distribute the points I choose?
2) How many points in the chamber should I simulate photo-detector data at?
3) How many events should I simulate per point?

All three of these questions can only be reasonably answered through trial and error after making some preliminary insights. For a reliable inverse mapping, it may be more critical to collect data in certain regions of the chamber than in others, and it may not be necessary to generate data at a large number of points. Further, it is more important to the operation of XEPET for the position reconstruction to be reliable in certain regions of the chamber than it is in others.

2.4.1 Generating Grid Points

Without knowing the answer to the first question in great detail, I wrote the program `pointgen.cpp` in C++ (see appendix A), which allows one to evenly distribute points in a specified region of the XEPET chamber. This way the density of points can be increased in regions where it is most beneficial, and there is no a priori bias towards one part of the chamber or another.

For my discussion of how this was accomplished, I suggest you make reference to figure 1.3. Letting x vary while holding y and z constant, I generate $P_x = \lfloor \frac{11.122}{a} \rfloor$ points. Similarly, I have $P_y = \lfloor \frac{15}{a} \rfloor$ points for x and z constant, and $P_z = \lfloor \frac{11.05}{a} \rfloor$ points for x and y constant, where

a is the grid spacing in centimetres. The program effectively generates an expanding cube of evenly spaced points about the centre of the rectangle. This way, if the rectangle is cut through the centre by xy, xz and yz planes, the generated points are symmetric about each plane. This symmetry can be used advantageously with machine learning algorithms. To remove points in this grid that aren't in the chamber, I check the slope each point forms with the upper edge of the chamber in its yz plane. Each point is stored as a data structure with an x, y, and z component. To filter out all points but those within a specific region, the program loops through all points and excludes those that have an "out of range" x, y, or z value.

As an example, one can use `pointgen.cpp` to generate grid points with a spacing of 1.5 cm throughout the chamber, in a text file called `results.txt`, through executing the command:

```
./pointgen 1.5 results.txt
```

2.4.2 Generating Training Points

The training points are the points I use to reconstruct position and energy. In some sense, although sometimes quite indirectly, most position reconstruction algorithms would take as input photo-detector data, and then determine which photo-detector data corresponding to the training points it most resembles.

In answer to the question, "how many events should we simulate per point?", *sometimes*, (but not always), I will want to simulate photo-detector data at as many events as practical for training points, and take the average result. If we simulate photo-detector data for a single event at a given point, and repeat this experiment several times, we will find that the photo-detector data differs on each trial: this is expected since the photons are sent in random directions. If we were to compare at random the results of two of these trials with one another, they would most likely differ more greatly than if we chose one trial at random, and compared the results to the average photo-detector data for several runs simulated at the same location. This follows, since the more photo-detector data we simulate at a point, the more the average results correspond to the proportion of solid angle each photo-detector comprises from the perspective of that point. In fact, if we were to run an infinite number of simulations at a given point in the chamber, and average the results, the number of photons we would receive at each photo-detector divided by the total number of scintillation photons (assuming no photo-absorption or scattering), would be the proportion of solid angle the surface of the photo-detector represents for that point. Further, a forwards mapping from points in the chamber to solid angle will be smooth and one to one, and so a true inverse function exists. On the other hand, the guarantee of a one-one function is not extraordinarily valuable: the likelihood of running one event simulations at different points and producing the same photo-detector data is extremely small.

In any case, we may want to model the proportion of solid angle that a photo-detector would comprise for each grid point. Since we can only run simulations with a finite number of events, and the more events we run, the more time it takes for computations to complete, this becomes a cost benefit optimization problem, where we must optimize the difference between the accuracy of our solid angle approximation, and the cost of computations.

To estimate a solution to this problem, I simulated data at the centre of the chamber three times, for 10000 events, 5000 events, 1000 events, 500 events, 200 events, 100 events and 20 events,

using a 40000 photon scintillation yield per MeV, and a 0.511 MeV electron. We can define the diffence in photo-detector data between points A and B, $d(A, B)$ by,

$$d(A, B) = \frac{1}{32} \sum_{i=1}^{32} |PD_i^A - PD_i^B| \tag{2.1}$$

where PD_i^A, PD_i^B represent data at the i^{th} photo-detector for points A and B, respectively. This is the the average absolute difference between data at the photo-detectors for two simulations. Using this definition, the 10000 event simulations differed from one another by 0.01 (at the origin of the chamber). The 5000, 1000, 500, 200, 100, and 20 event simulations, on average, differed from the 10000 event simulations by 0.05, 0.15, 0.21, 0.41, 0.85 and 1.11, respectively. In the centre of the chamber, the photo-detectors will be most sensitive to a change in position on the whole. The difference between photo-detector data for 10000 events at (0,0,0), versus a point within 2 cm of (0,0,0) is, on average, 4.79. While for a change in position from (0,0,0) by 0.2 cm, the average difference between photo-detector data at 10000 events is approximately 0.78.

This means if we want to reconstruct the position of a particle emitting light in all directions, with an error of at least 0.2 cm, then we will need to average the data of at least 200 events per simulation. This only puts a lower bound on the error of reconstruction using this method of choosing training points. Near the boundaries of the chamber, the average photo-detector data appears erratic even with a large number (10000 or more events). And given the effects of Compton scattering and photo-absorption, the average-photo detector data will not converge with an arbitrarily high accuracy.

In order to properly take into account computing time in answering our third question, we must first have an idea about the answer to one of our questions: "how many points in the chamber should we simulate photo-detector data at?"

Photons interact in liquid xenon with attentuation $e^{(-\frac{7}{9})*(\frac{L}{3.1})}$, where L is the distance into the liquid xenon [21]. Therefore a large proportion of photons (likely at least 53%) interact in the first 3 centimetres of the chamber: (x,y,-5.525) to (x,y,-2.425) in ffcards coordinates. With this in mind, together with some experimentation using the K Nearest Neighbour approach to reconstruction in chapter 3.2, a grid with a spacing of 0.2 centimetres was chosen for the lower three centimetres of the chamber with 200 events per grid point, and a grid with 0.5 centimetre spacing with 500 events per grid point was chosen for the whole chamber. For testing purposes, a grid with a spacing of 1.0 cm and 1000 events per grid point was also generated. The statistics for these grids is given in table 3 in the summary of this chapter.

Reducing the spacing substantially increased the accuracy of the "K Nearest Neighbour" position reconstruction (chapter 3.2). However, the number of grid points is proportional to $\frac{1}{(spacing)^3}$, and therefore the number of grid points increases at an increasing rate as the grid spacing decreases. With a fixed computation time of approximately 1.32 seconds per event, decreasing the spacing between grid points becomes increasingly infeasible. Also, there will be a limit on the usefulness of decreasing grid spacing: below a certain spacing, the difference between photo-detector data from event to event will be greater than difference between the averaged photo-detector data for adjacent grid points. And increasing the number of grid points could result in longer waits for position and energy reconstruction.

Finally, *I did not always use average photo-detector data for training points.* While I exclusively used average photo-detector data for training with the K Nearest Neighbours approach, our

final Neural Network solution was trained on a mixture of average data and single event data. In this case, our network was "intelligent" enough to use the deviation in data from event to event to provide a superior model of the solution to our problem.

This section was necessarily tedious, but now we are nearly prepared for the more interesting content in chapter 3.3 onwards – position reconstruction with neural networks!

2.4.3 Generating Test Points and Noise

Data at test points should be representative of what we would expect in practice: the test points are chosen to be at different locations than the training points, and then given to a reconstruction program to see how well they are re-located. Therefore we only wish to simulate photo-detector data for 1 event per test point: the photo-detector we see in practice will not be the average of several events at the same point. Since this test data is intended to simulate the type of data we would expect in practice, we must also add additional noise.

We may use Silicon Avalanche Photo-diodes (APDs) as photo-detectors. When a photon is incident with an APD, there is only a 70% chance that it will be converted into a "photo-electron". When using APDs, there is also an error called "multiplication statistics": this additional noise is a Gaussian random variable with standard deviation equal to the square root of the number of photons detected. Finally, there is additional noise from the APDs that we model as a Gaussian random variable with mean 0 and standard deviation 6. These statistics for the expected noise from the APDs are explained in [35].

To summarize, suppose $PD1$ photons travel into photo-detector 1, and let $PD1'''$ be my estimate of the number of photons detected.
Step 1. Let $PD1' = Binomial(n = PD1, p = 0.7)$, the result of PD1 Bernoulli trials with a probability of detection, $p = 0.7$. This is the noise from photo-electron conversion.
Step 2. Let $PD1'' = Gaussian(\mu = PD1', \sigma = \sqrt{PD1'})$ (multiplication noise)
Step 3. $PD1''' = PD1'' + Gaussian(\mu = 0, \sigma = 6)$. (additional noise)
Step 4. Use $PD1'''$ as test data for photo-detector 1.

As seen in chapter 3, step 1 can result in large errors in reconstruction. But fortunately, this step can be omitted. Since roughly 40000 photons will be emitted by any point, and several thousand will be detected, we can account for quantum efficiency by assuming a 70% scintillation yield (p=*0.7*) in our test simulations; since N is large, the probability that the number of photo-electrons differs from Np by any substantial quantity is negligible, and so step 1 can be eliminated. Interestingly enough, this same reasoning is applied in statistical mechanics. If we have a gas (comprised of many atoms) in a container, it would violate the second law of thermodynamics to have the atoms congregate in a portion of the container. The probability that this will happen is a ratio of binomial probabilities, with the case of evenly distributed gas in the denominator (the average case). This probability is negligible for large N, which means there is no practical disagreement between statistical mechanics and the second law of thermodynamics.

Finally, we may use photo-tubes instead of APDs, in which case the noise on testing data is substantially reduced. So the results of reconstruction without adding noise to the test data is important.

Spacing	Points	Events	Energy	Region	Type
1.0 cm	1365	1000	511 keV	Whole chamber	Training
0.5 cm	11687	500	511 keV	Whole chamber	Training
0.2 cm	33825	200	511 keV	First 3 cm	Training
0.41 cm	3811	100	511 keV	First 3 cm	Testing
0.41 cm	19943	1	511 keV	Whole chamber	Testing
1.7 cm	297	1	511 keV	Whole chamber	Testing
0.23 cm	22165	1	511 keV	Whole chamber	Both
0.41 cm	3811	1	450 keV	First 3 cm	Testing
0.41 cm	3811	1	300 keV	First 3 cm	Testing

Table 2.1: **Simulated photo-detector data generated**. In all simulations, ffcards parameters not listed in this table were specified as in the appendix. In particular, a 40000 photon per MeV scintillation yield was used. "Both" means both used for training and testing.

The test grids generated are summarized in table 2.1.

2.5 Running the Simulations

The `ffcards` simulations were run with coordinates specified by the training and test grid points. For every grid point, the `xepet.ffcards` file needed to be edited to change the run number, the random seed, and the coordinates. In order to accomplish this, I wrote the Perl scripts `change`, `xecomaker`, and `xecorun`. See appendix A for more details.

2.6 Summary

Coordinates for grids of training and test points were generated in text files by `pointgen.cpp`. The coordinates are in euclidean 3 space, with (0,0,0) representing the centre of a XEPET chamber. These text files with grid point coordinates were read by a Perl script (`change.pl`) which generated corresponding `xepet.ffcards` files. The ffcard simulations were then initiated by other perl scripts (`xecomaker`, `xecorun`). The simulation output files were converted from `.hbook` to `.root`, and the relevant data in the `.root` files was extracted with a macro written in C++ (e.g. `nneural.C`). See the appendix A for information on how to obtain these files.

Table 2.1 summarizes the data I simulated. It took 1.32 seconds of computing time to simulate data at one point for 1 event. So for instance, the 200 event 0.2 cm spacing data took approximately 103 days of computing time to generate.

With the results of our simulations, we can now proceed to position and energy reconstruction. For position reconstruction, we want to analyze how photo-detector data maps to points in the chamber. For energy reconstruction, we use a mapping from points in the chamber to photo-detector data, after having re-constructed the position of a particle from its corresponding photo-detector data.

Chapter 3

Position Reconstruction

We want to run a regression on the *training data* – we want to correlate known inputs and outputs so that we can find reasonable outputs for new inputs. As described in chapter 2, the known inputs and outputs are referred to as the *training data*, and the new inputs (and the corresponding target outputs) are referred to as the *testing data*. The training data may represent something somewhat different than the testing data, for reasons explained in the previous chapter; essentially, the difference may result in a model that generalizes more successfully to fitting new training data. However, in both the training and testing sets, the input represents photo-detector data in some form, and the output is a point in the XEPET chamber; my ultimate goal is to localize the position of a point emitting light in random directions that generates the photo-detector data we observe. Further, both methods for reconstruction use the same coordinate-system as used by *ffcards*, described in chapter 2 – a 3D Cartesian system with (0,0,0) representing the origin of a XEPET chamber.

In this chapter, I discuss two models for position reconstruction – *K Nearest Neighbours* and *Neural Networks*. The K Nearest Neighbours model was my first attempt at position reconstruction, and it is described in the next section. The training data used in the K Nearest Neighbours (KNN) model is average photo-detector data, while the training data used in the neural network model is a mixture of scaled average photo-detector data, and scaled single-event photo-detector data. The K Nearest Neighbour model is conceptually simpler than the neural network model, and it is more successful than most neural network configurations. However, the final network model used is more successful than any approach I have attempted, and hence, it is more likely to be used for position reconstruction with XEPET than K Nearest Neighbours. As such, the neural network model is tested and presented in greater detail – it represents the most significant accomplishment in this thesis. Further, the majority of the analysis with neural networks is focused on the first three centimeters of the chamber, where the highest quality training data has been generated, and where the majority of reconstruction will need to be performed (see chapter 2 or further ahead). Despite this, it is still useful to present the K Nearest Neighbour model – it documents my progress on this project, it provides a meaningful comparison with the final neural network model, it further familiarizes the reader with this problem, and it provides others with an easy solution to their own multivariate regression problems.

3.1 K Nearest Neighbours

K Nearest Neighbours is a simple algorithm. Assume we have a training set of input vectors and corresponding output vectors. Then suppose the photo-detectors in a XEPET chamber sense that an interaction has taken place, and we want the K Nearest Neighbour algorithm to tell us roughly where this interaction was. We will have data at each of the 32 photo-detectors. We

input this test data, a 32 dimensional test vector, into our reconstruction program. The algorithm then measures the distance between this 32 dimensional test vector we used as input, and every 32 dimensional photo-detector vector in the training set. It finds the k photo-detector vectors in the training set with the smallest distance to the input vector. In my implementation, distance is taken to mean the Euclidean distance between two vectors in 32 space. The Euclidean distance between two vectors, $\mathbf{w}$ and $\mathbf{r}$, in N space is

$$d(\mathbf{w}, \mathbf{r}) = \sqrt{\sum_{i=1}^{N} (w_i - r_i)^2} \tag{3.1}$$

where w_i and r_i are the i^{th} elements of $\mathbf{w}$ and $\mathbf{r}$, respectively.

Each of the k nearest photo-detector vectors have corresponding output vectors in the training set, $a^{(1)}, \ldots, a^{(k)}$, representing positions in the XEPET chamber: each photo-detector vector in the training set was generated at a point, or "output vector", in the chamber. In my implementation, the average of these output vectors is used as the simulation output corresponding to an input test vector; the output vector $\mathbf{v}$ can be written as

$$\mathbf{v} = \frac{1}{k} \sum_{j=1}^{k} \mathbf{a}^{(j)} \tag{3.2}$$

The radial error on reconstruction is the Euclidean distance in 3 space between the simulation output vector and the target output vector (the test photo-detector data used as input is generated at a point of our choice, and that point is referred to as the target output vector; if the reconstruction program works perfectly, it would map the test input to this output).

The error on reconstructing the x, y, or z coordinate is taken as the absolute value of the difference between the simulated reconstructed coordinate and the target reconstructed coordinate. Given a set of test vectors, my simulation outputs error statistics for $k \in [1, \ldots, 10]$. This way k can be chosen to solve this particular problem in a way that minimizes the error of reconstruction.

It may seem that k should always be one – that by including other further neighbours we are diluting the accuracy of simulation output. However, noisy data may include training vectors that are nearer to the test input vector than training vectors that have output coordinates closer to the target output. Therefore by including several neighbours in determining simulation output, we can sometimes mitigate the effects of noise.

The simplicity of the algorithm is also a drawback. Although it may be easy to modify, the algorithm does not modify itself (like a machine learning algorithm), and there are sufficiently many physical processes underlying the training data that it would be difficult to determine a superior approach to (2) without much trial and error. And since this is a sophisticated problem, the simple model we are using can likely be adjusted to greatly improve reconstruction. This is the appeal of machine learning algorithms – these beneficial adjustments can be made with less difficulty. However, in practice, it is still difficult to adjust other algorithms to outperform KNN, and in many cases, other algorithms will be less successful without significant effort. For this reason, KNN is a competitive approach to regression for most practical applications.

Computation time is another drawback. If the training set is necessarily large (e.g. exceeds 100000 points), and reconstruction must be performed quickly (e.g. less than 0.1 seconds per reconstruction), then other approaches should be explored.

3.1.1 Implementation

This section is essentially a user's guide for my implementation of K Nearest Neighbours. See the appendix for a reference to the source code, `knn.cpp`. However, this section is not important to the analysis of the problem, and should be considered optional. In this thesis, the primary purpose of describing the KNN approach is to further familiarize the reader with the problem of position reconstruction, and to compare the results with the more successful neural network approach. For a review of the results, see the next section.

In order to use "knn.cpp", the training data and test data must be formatted in certain way. Presently, the data for any point should be in its own file. For instance, `train0.txt`, `train1.txt`, `train2.txt`, ... The first three lines of these text files should be the true spatial coordinates of a point in the chamber, and the next 32 lines should be the photo-detector data corresponding to this point. For example, if `train0.txt` contains photo-detector data for the point (1.0, -2.5, 3.0), then it should look something like this:

train0.txt:

```
1
-2.5
3.0
32.4
39.6
...
41.7
```

In order to easily generate these text files, follow this process:
1) Put all .root files for training or testing simulation data in an empty directory
2) Specify the name of the text files you want in the macro kphoto.cpp (see appendix)
3) Open one of the root files, and execute the macro `kphoto.cpp` by typing: `.x kphoto.cpp`

To run `knn.cpp`, execute the command

```
./knn [training data files] [test data files] [output statistics file] [target 1]
[target 2] [target 3] [target 4]
```

For instance, if the training data files are `train0.txt`, `train1.txt`, ..., and the test data files are `test0.txt`, `test1.txt`, ..., and we want to output the statistical results in `stats.txt`, and our targets are 4 cm, 2.5 cm, 1.5 cm, and 1.0 cm, we would type:

```
./knn train test stats.txt 4 2.5 1.5 1
```

Targets 1, 2, 3 and 4 must be listed in decreasing size. The following is an example of a possible `stats.txt`, supposing there are 11687 points in `train.txt`, and 297 points in `test.txt`. In this example, the results are only displayed for 1 nearest neighbour, although the actual `stats.txt` will display the results for 1,2,3,... up to 10 nearest neighbours.

```
Statistical Data for Nearest Neighbours

Number of examples (grid points):  11687
Number of test points (input):  297
Minimum target accuracy:  4 cm
Other targets:  2.5 cm, 1.5 cm, 1 cm

Statistics for 1 nearest neighbours
90.9091% outputs under largest target
80.4714% outputs under next largest target
52.1886% outputs under next largest target
27.6094% outputs under next largest target
Average distance between input and output:  1.93064

Maximum distance:  9.83972
Coords of input for this maximum:  (0, 3.4, 5.1)
Corresponding filename:  input145.txt
Coords of output for this maximum:  (0.5, 5.5, -4.5)

Minimum distance:  0.141421
Coords of input for this minimum:  (-3.4, 0, 3.4)
Corresponding filename:  input22.txt
Coords of output for this minimum:  (-3.5, 0, 3.5)

97.3064% output under largest x target
93.9394% output under next largest x target
82.8283% output under next largest x target
70.7071% output under next largest x target
99.6633% output under largest y target
92.9293% output under next largest y target
76.0943% output under next largest y target
60.6061% output under next largest y target
94.9495% output under largest z target
91.5825% output under next largest z target
84.5118% output under next largest z target
73.064% output under next largest z target

Average x distance:  0.885859
Average y distance:  0.960943
Average z distance:  0.970707
```

The above example reveals that the radial error on position reconstruction was less than 4 cm, 2.5 cm, 1.5 cm, and 1 cm, for 91%, 80%, 52% and 27% of the test data, respectively. Further, the

average radial error is given as 1.93 cm. This means, on average, that the results will be accurate to within a sphere of radius 1.93 cm. The same targets used to assess the radial accuracy are used in evaluating the accuracy of reconstructing each spatial coordinate. For instance, in the above example, 99.7% of the reconstructed output had a y coordinate with error less than 4 cm, while the average error on each reconstructed output is less than 1 cm.

Finally, the program generates a file named `compare.txt`, which lists every test coordinate, next to the corresponding reconstructed coordinate, with the distance between the two coordinates, for 1,2,...,10 nearest neighbours.

Since I configured neural networks that significantly outperform the KNN paradigm, I have not optimized the software described in this section for user convenience or to minimize computation time. Using a training-data set with 33825 points, this software presently takes approximately 0.3 seconds to reconstruct the position of an input photo-detector vector. I predict this can be improved to take on the order of 0.01 seconds if the software were modified so that all training data is in one file, and all test data is in is in one file. This would considerably reduce time-consuming file reading operations. If this change were made, the training file, for example `training.txt`, would likely be a concatenation of all training files use in the original implementation (e.g. `train0.txt`, `train1.txt`, ...), and the same for the test files. The program could then be used to produce the same results as above, with the possible instruction:

```
./knn training.txt testing.txt stats.txt 4 2.5 1.5 1
```

If the reader wishes to continue my work and to use "K Nearest Neighbours", I suggest these modifications be made. A reconstruction time of 0.3 seconds per photo-detector input is costly in this context, where at least 100000 reconstructions should be performed in under an hour. However, reconstruction time will not be noticeably improved by re-coding this paradigm in a different programming language – this software is presently compiled from efficient C++ code into a native binary executable.

3.1.2 Results with Discussion

The results of KNN reconstruction are extremely useful in that they provide an estimate of how accurately position reconstruction can be performed, and which factors will most affect the accuracy of reconstruction. As always with this KNN implementation, reconstructions are produced using *average* multiple event training data, and single event test data.

KNN was first tested with a training grid with a spacing of 1.0 cm (amounting to 1365 training points), averaged over 1000 events per point, the average radial error in reconstructing 297 noiseless test points (generated throughout the entire chamber, using a grid spacing of 1.7 cm). The average radial error of reconstruction monotonically decreased from 3.15 cm with one nearest neighbour, to 3.09 cm using the average of four nearest neighbours. The radial error then increased up to 3.25 cm using the average of ten nearest neighbours. With four nearest neighbours, 76%, 58%, 23%, and 10% of reconstructed outputs have a radial error less than 4.0, 2.5, 1.5, and 1.0 cm, respectively. The average x, y, and z, errors are 1.3, 1.6, and 1.7 cm, respectively. And 71%, 45%, and 63% of the x, y, and z errors are under 1.5 cm, respectively. The minimum radial error is 0.1 cm, the maximum is 8.23 cm.

For position reconstruction from scintillation light to be useful, the radial error on noiseless data should be less than this – it should be at least less than 2.5 cm on average. In order to improve the accuracy of reconstruction, the grid spacing was reduced to 0.5 cm (amounting to 11687 training points) averaged over 500 events per point. The same noiseless testing data is used as input into the KNN program. With this input, minimum radial error is achieved using the average of the first two nearest neighbours, though the average radial error only varies from this by at most 0.3 cm using other numbers of nearest neighbours. The average radial error using two nearest neighbours is 1.9 cm. The minimum radial error is 0.2 cm, while the maximum is 9.8 cm. With two nearest neighbours, 91%, 80%, 51%, and 29% of the reconstructed outputs have a radial error less than 4.0, 2.5, 1.5, and 1.0 cm, respectively. Further, the average x, y, and z, errors are 0.86 cm, 0.95 cm and 0.96 cm, respectively, while 87%, 75%, and 85% of the respective x, y, and z coordinates had an error less than 1.5 cm.

This is a dramatic improvement over reconstruction with a 1.0 cm grid spacing – the average radial error is reduced by 40%. This suggests that further reducing the grid spacing could greatly improve reconstruction accuracy.

Continuing with this 0.5 cm spacing training set, all of the noise discussed in chapter 2.4.3, including the photo-electron conversion error, was added to the 297 point testing set. The highest accuracy reconstruction took place when using the average of the results for 10 nearest neighbours (the maximum number attempted by this implementation). The average radial error is 4.8 cm, with only 41%, 17% and 5% of test points respectively under a 4.0 cm, 2.5 cm, and 1.5 cm radial error. The average error in reconstructing the x, y, and z, coordinates is 1.9 cm, 2.5 cm, and 2.6 cm, respectively. 48%, 35%, and 38% of the x, y, and z, coordinates respectively had an error of less than 1.5 cm. Finally, the worst reconstruction took place for one nearest neighbour, where there was on average a 5.3 cm radial error.

Adding noise substantially affected reconstruction accuracy – on average, reconstruction has an increased radial error of 3 cm. But, as discussed, we can avoid adding error for photo-electron conversation. Removing the photo-electron conversion noise, but keeping the rest, three nearest neighbours yields the most accurate reconstruction, with an average radial error of 3.3 cm. 73%, 53%, 26%, and 11% of the radial error was under 4.0 cm, 2.5 cm, 1.5 cm, and 1.0 cm, respectively. Further, the average x, y, and z, coordinate error is 1.5 cm, 1.5 cm, and 1.7 cm, respectively. 65%, 62% and 63% of x, y, and z, error is under 1.5 cm, respectively. Using 10 nearest neighbours the average radial error is at its worst, but not much different than with three nearest neighbours, at 3.4 cm.

Without removing noise due to photo-electron conversion, KNN could not be considered a viable approach to this problem. However, the photo-detector training data should be scaled, and scaling data can have a dramatically negative affect on this implementation. If the data is scaled by dividing the number of photons each photo-detector receives by the number of photons detected, the average radial error of increases by approximately 1.4 cm. And scaling data is necessary in this application – the number of photons we detect will depend on the energy of the particle, which is unknown. So we must be able to reconstruct position from a pattern at the photo-detectors, somewhat independent of the number of photons detected.

For the full results as output by the KNN reconstruction program, see the following files referred to in the appendix:

`stats.txt`: 1.0 cm grid, no noise
`cleanstats.txt`: 0.5 cm grid, no noise
`fullnoise.txt`: 0.5 cm grid, all noise
`somenoise.txt`: 0.5 cm grid, all noise except photo-electron conversion
`scaledstats.txt`: 0.5 cm grid, data scaled as described, no noise
`noisescale.txt`: 0.5 cm grid, data scaled as described, all noise
`somenoisescale.txt`: 0.5 cm grid, data scaled as described, some noise

3.1.3 Summary

As expected, K Nearest Neighbours provides information useful for understanding and better solving the position reconstruction problem. The observed results reinforce the notion that a larger number of nearest neighbours is more successful for reconstructing noisy data. The results also indicate that further reducing the training grid spacing will improve the accuracy of reconstruction. Reducing the grid spacing can be done, but only with difficulty – the 0.5 cm spacing grid that was used takes approximately 89 days of computing time to generate, and further decreasing grid spacing increases the number of training points cubically. And the speed at which K Nearest Neighbours performs reconstruction depends linearly on the number of training points used. With a 0.5 cm spacing grid, the number of training points is such that the reconstruction speed is already prohibitive.

While the KNN results are useful, they are not of high enough quality to be of great use for position reconstruction. Armed with a better understanding of this problem, I attempted to improve reconstruction using neural networks.

3.2 Overview of Neural Networks

It's exciting to have the opportunity to use neural networks. There is always great interest in projects outside of computer science where machine learning can be of use. Artificial intelligence is a rapidly growing and promising field, although in many ways it has been self-contained. I expect that projects like this thesis will increase awareness and open minds towards the potential machine learning has for application; in the future, artificial intelligence will hardly be self-contained. I believe that twenty-first century developments in machine learning will profoundly affect mankind – to the same extent electricity, transportation, or the Internet have affected those who lived in the twentieth century.

As opposed to conventional computing, which is based on explicit set of programmed instructions, neural networks learn the solution to a problem through exposure to a set of examples; this learning paradigm was inspired through studying connections between neurons in the human brain. In the context of position reconstruction for XEPET, the training examples are points in space corresponding to photo-detector data, and the solution is to generate a function that maps new photo-detector input to points in space where this data was likely generated.

In this case, the function is the neural network, and the solution is reached by adjusting the weights and biases that define this function in order to best fit the training examples. This process of adjusting weights and biases is referred to as *training*, and can be computationally intensive. However, once the weights and biases are fixed, the neural network processes new input very

rapidly. Networks are also known to be robust to noise.

It appears that neural networks are ideal for this problem. Fast reconstruction is necessary for high patient throughput and low cost scanning. And reconstruction must take place even with noisy input: noise will come from photo-detector inefficiency, reflection off the walls of the detection chamber, and unknown physical processes. Further, a network will model these unknowns without the need for someone to do a first principles analysis.

The next sections serve to introduce the neural network used to solve this regression problem, while including necessary theory along the way.

3.3 Neural Network Structure

I constructed a six layer feed-forward back-propagation neural network with the Levenberg-Marquardt training algorithm. The structure of this network is illustrated in figure 3.1. I refer to the first layer as the *input layer*. The following four layers are commonly referred to as *hidden layers*, and the final layer is the *output layer*. Layers 2, 3, 4, and 5 have 26, 17, 23, and 16 neurons, respectively. For this reason, this network could be referred to as a `[32-26-17-23-16-3]` network, although there is no consistent notation in the literature. The `32` comes from there being 32 initial network inputs, and the `3` from there being 3 final outputs. I built and tested this network in Matlab.

There are 32 inputs variables in the first layer, $x_1, \ldots, x_{32}$. Each input variable represents one of the 32 photo-detectors; for example, if 27 photons are recorded by photo-detector 4, then $x_4 = 27$. In the next layer there are 26 *neurons*. This means that the 32 inputs are mapped to 26 outputs, $z_1, \ldots, z_{26}$, by *weights*, *biases*, and a *transfer function*. Explicitly, the outputs of the first two layers are governed by the following equations:

$$z_j = g(a_j) = g(\sum_{i=1}^{32} w_{ji}^{(1)} x_i + b_j^{(1)}) \tag{3.3}$$

$$q_k = g(m_k) = g(\sum_{j=1}^{26} w_{kj}^{(2)} z_j + b_k^{(2)}) \tag{3.4}$$

where $j = 1, \ldots, 26$, and $k = 1, \ldots, 17$. The superscripts in (3.3) and (3.4) indicate a layer number. For instance, $w_{ji}^{(1)}$ and $b_j^{(1)}$ are weights and biases that transform photo-detector input to the outputs of the first layer. The outputs in the following two hidden layers are determined in a similar way.

The data is best modeled with a non-linear function. This is an intuitive assumption verified by trial and error. I therefore chose to use *tansig* transfer functions to map each layer to the next, except for the mapping from the penultimate layer to the final network output (the position in the chamber). For that, I used the linear transfer function $l(m) = m$.

The tansig function is a smooth non-linear function, bounded in (-1,1):

$$g(x) = tansig(x) = \frac{2}{1 + e^{-2n}} - 1 \tag{3.5}$$

It is illustrated in figure 3.2.

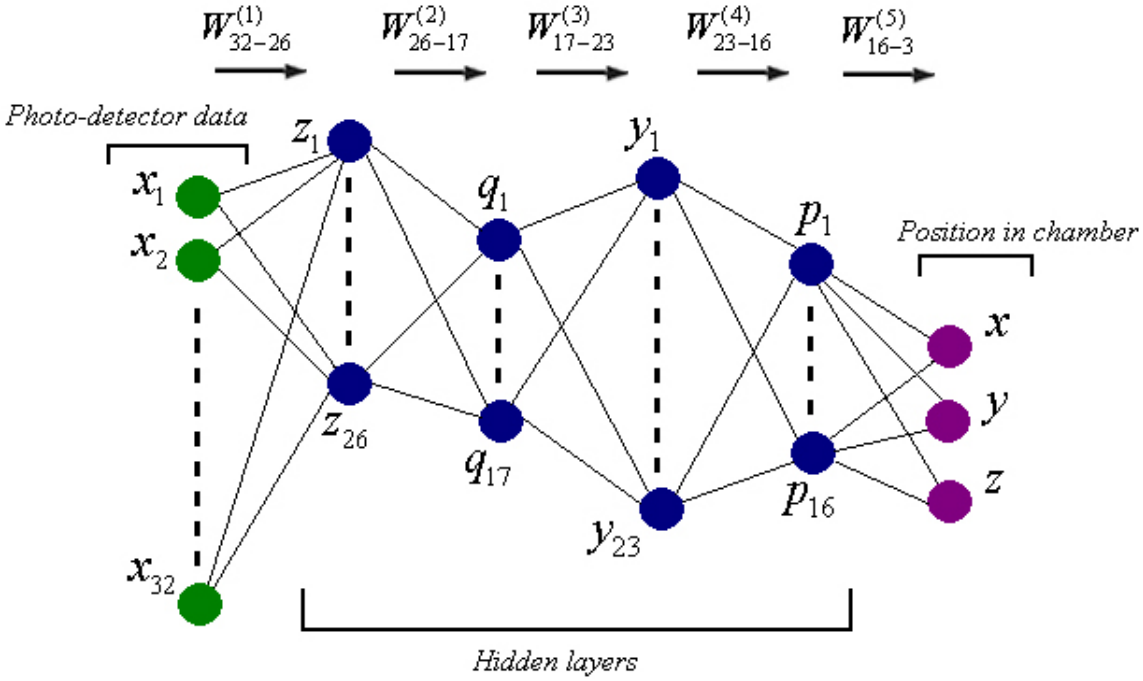

Figure 3.1: **Structure of the network used for position reconstruction.** The inputs are coloured green, the outputs purple, and the hidden layers blue. For those seeing this in black and white, the first layer is the input layer, the following four layers are hidden layers, and the final layer is the output layer. The 32 inputs are carried through to 26 outputs by a weight and bias matrix and a transfer function. These 26 outputs are then similarly carried through to 17 outputs, then to 23 outputs, then 16 outputs, and then finally, to the 3 outputs that represent the x, y, and z, coordinates of a position in a XEPET chamber.

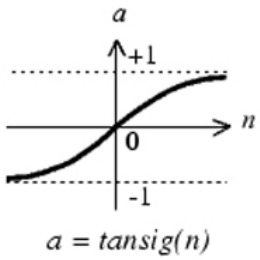

Figure 3.2: **Tansig transfer function used in all hidden layers except the last hidden layer.** This transfer function is smooth and bounded in (-1,1). Figure taken from [20].

The output of each neuron in the final layer cannot be bounded in (-1,1), since each of these outputs represents a spatial position in the chamber; these outputs are instead bounded by the dimensions of the chamber. The y output, for example, could be in the range [-7.5, 7.5]. Although a scaled tansig transfer function could be used, reconstruction is more successful using a linear transfer function, *l(m)=m*. This isn't surprising, since it often recommended that this transfer function be used for final layer output in regression problems [20]. The linear transfer function is successful in part because it is the simplest and least computationally intensive mapping that can provide the network with complete discretion over final layer outputs.

With this linear transfer function, the final network output – representing x, y, and z coordinates in space, corresponding to photo-detector input in the first layer – is determined by equations 3.6, 3.7, and 3.8:

$$x = \sum_{r=1}^{16} w_{1r}^{(5)} p_r + b_1^{(5)} \tag{3.6}$$

$$y = \sum_{r=1}^{16} w_{2r}^{(5)} p_r + b_2^{(5)} \tag{3.7}$$

$$z = \sum_{r=1}^{16} w_{3r}^{(5)} p_r + b_3^{(5)} \tag{3.8}$$

In the next section, I discuss how the network adjusts weights and biases when exposed to training examples. In the section following that, **Network Generalization**, I discuss how two concepts – regularization and early stopping – can be employed to improve how the network generalizes to new input data. This discussion will also clarify many questions that are likely in reader's mind; notably, how I decided on my network design – how I chose how many hidden layers to use, and how many neurons to use in each hidden layer. Cross-validation influenced these decisions.

3.4 Network LM Backpropagation

Backpropagation is a term used in neural computing literature to describe a number of different things [22]. In this thesis, I use its most well-known definition: backpropagation is a general paradigm feed-forward neural networks commonly use to choose weights and biases in regression problems. It was first introduced by Paul Werbos in 1974 [25], but became widely known after the work of Williams, Rumelhart, and Hinton in 1986 [26], leading to a revolution in machine learning. Backpropagation is referred to a supervised learning technique because it depends on having been provided correct output corresponding to examples of input.

As of April 3, 2008, *Wikipedia* provides a good elementary discussion of this subject, in the article *Backpropagation*. I paraphrase the general method discussed there, making alterations to reflect the context in which backpropagation is used in this thesis.

First, start the algorithm so that the current layer is the output layer (in this case, the layer that outputs a position in the chamber). Then input a set of training examples into the neural network. Then the algorithm:

- Compares the output of the current layer to the output desired for this layer, and assigns an error measure to the current layer. (The training examples contain the desired output of the final layer).
- Adjusts weights and biases of the previous layer to minimize this error, in a process that can involve assigning a different blame for this error to each neuron in the previous layer. The most blame goes to the neurons with the largest weights and biases.
- Repeats these steps for neurons at the previous layer.

The network determines the *error measure* by an error function. For this application, I use a regularized mean square error (RMSE) function. The RMSE function for N network outputs, $m_1, \ldots, m_N$, given N network output targets, $t_1, \ldots, t_N$, weights $w_1, ..., w_N$, and biases $b_1, .., b_N$, and a performance ratio γ, is

$$E = \frac{\gamma}{N}\sum_{i=1}^{N}(m_i - t_i)^2 + (1-\gamma)\frac{1}{N}\sum_{i=1}^{N}(w_i^2 + b_i^2) \tag{3.9}$$

The reasoning behind this error function (and the parameter γ) will become more clear in the next section. The network I used updates its weights and biases after having been exposed to all training examples – this is called *batch training* as opposed to *incremental training*.

Bishop describes back-propagation as fundamentally a two step process: first, compute the derivatives of the error function, then use the derivatives to compute adjustments to the weights and biases [22]. One of the most frequently used and simple backpropagation algorithms is called gradient descent. In very general terms, if $\mathbf{w}$ represents a vector containing weights, and E is the error function, then iteration k+1 reads,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - a_k \nabla E \tag{3.10}$$

where a_k is the learning rate at the k^{th} iteration. This process is described more precisely in [16], but the above equation essentially means that on each iteration the network updates the weights and biases in the direction that the error function decreases most rapidly (its negative gradient). Since I use *batch training* the weights and biases are updated only after the network is exposed to all training examples.

Sometimes it also useful to compute the Hessian matrix $\mathbf{H}$ for updating weights and biases. For instance, the Hessian is used in a procedure for quickly re-training a feed-forward network following a small change in training examples [24]. Also, many non-linear optimization algorithms for training neural networks rely on information about the second order-properties of the so-called "error surface", which is determined by the Hessian matrix [23].

The Hessian matrix is a matrix of second derivatives of a function. The i^{th} row and j^{th} column of the Hessian matrix is

$$\mathbf{H}_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{3.11}$$

where w_i is the i^{th} element of a weight vector, $\mathbf{w}$, and E is the error function. H_{ij} is evaluated at $\mathbf{w}$.

In this instance, the Jacobian matrix $\mathbf{J}$, is defined by

$$\mathbf{J}_{ij} = \frac{\partial e_i}{\partial w_j} \tag{3.12}$$

where e_i is the i^{th} element of a vector of errors $\mathbf{e}$, computed by the error function (see [20]).

Since we are using an error function which is a sum of squares [20], the Hessian matrix can be approximated as,

$$\mathbf{H} = \mathbf{J^T J} \tag{3.13}$$

This leads to the *Levenberg-Marquardt* training algorithm,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - [\mathbf{J^T J} + q\mathbf{I}]^{-1}\mathbf{J^T e} \tag{3.14}$$

where q is a constant, and $\mathbf{I}$ is the identity matrix. When q is zero, this iteration is just the multivariable Newton's method, with $\mathbf{J^T J}$ approximating the Hessian matrix. As q increases in size, this method approaches the gradient descent iteration, where $\mathbf{J^T e}$ approximates ∇E.

The Levenberg-Marquardt (LM) algorithm is what I used to train the network for position reconstruction. Owing in part to the Hessian approximation, and the flexibility of being able to vary from a Newton iteration to a gradient descent iteration, the Levenberg-Marquardt algorithm generally consumes less memory and decreases the regression error more quickly than most other training algorithms.

In this section, I have condensed theory which is usually discussed in over two hundred pages in a carefully organized textbook. My explanations are therefore necessarily vague in places, but I have included them (rather than just stating that the network uses Levenberg-Marquardt and providing a reference), because if nothing else, they will give you a vague understanding of how my network chooses weights and biases. For further reading, I recommend Bishop's popular book, *Pattern Recognition and Machine Learning* [22]. However, some of the detail on the conjugate-gradient and Levenberg-Marquardt methods provided here follows Matlab's reference manual [20], since there is some inconsistency in the way these algorithms are described in the literature, and I designed the network I used with Matlab.

3.5 Network Generalization

How many hidden layers should a network have? How many neurons should a hidden layer have? While the answers to these questions are rarely certain, *cross-validation* is one of the only means we have to approach them. Cross-validation also helps prevent the neural network from fitting the training data too well. That surely seems like a drawback at first glance, but *over-fitting* is a serious problem. First of all, the training data may not be perfect, and so if it is fit too well, these imperfections are carried through in the regression, preventing it from generalizing properly to new data. Also, in order to fit the data perfectly, counter-intuitive functions which have no hope for generalization can be adopted. Regularization and cross-validation prevent this from happening. Perhaps it is somewhat deceiving to categorically classify cross-validation as a tool that prevents training data from being fit too well; more precisely, it can be used to optimize the fit so that the error on validation data is at a minimum, effectively improving how the regression

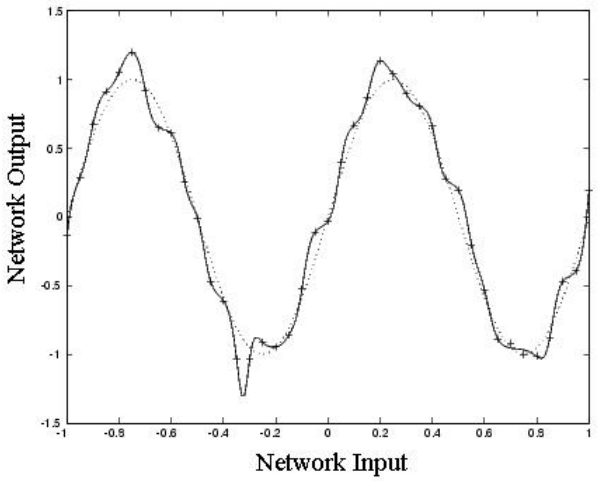

Figure 3.3: **Noisy sine curve fit with neural network.** Over-fitting a Noisy Sine Function. A neural network fits noisy sine data, marked by crosses. A dotted sine curve is plotted behind the fit. Though the model fits the data successfully, it does not have a clear meaning, and it will not generalize well to new data. [20]

generalizes to new data. All of the jargon in this paragraph is carefully defined in this section, and an example of over-fitting is shown in figure 3.3.

A neural network generalizes successfully when it accurately reconstructs input that wasn't used for training. Network generalization can be improved as simply as using a simple network, just large enough to adequately fit the training data. These networks may be unable to over-fit training data, since this requires complex functions that can only be afforded by larger networks. The more hidden-layers a network has, and the more neurons in each hidden layer, the more "intelligent" the network becomes – the more power the network has to model factors affecting training data – but also the more likely it is that the neural-network will over-fit. In the case of over-fitting, the network will likely have modeled meaningless artifacts in the training data. This makes the model less useful for mapping new data that does not contain these artifacts. In order to prevent over-fitting, I used cross-validation with early stopping.

Cross-validation is a term used in statistics. It is the practice of partitioning data into *training*, *validation*, and *testing* sets, and using the validation and testing sets to check analysis done on the training set. In chapter 2, I discussed generating training and testing grid data. The validation data consists of data taken from either the training data set, or the testing data set, or both.

Cross validation with early stopping is a simple mechanism to prevent over-fitting. The weights and biases of the network are updated to minimize the error on reconstructing training data. For instance, training photo-detector data was generated for points of my choice in a XEPET chamber. If this photo-detector data were used as input to my network, it would be mapped to an output point in the chamber. The network would compare this point to the point at which the input photo-detector data was generated, and adjust the weights and biases in the network to minimize this distance. This is a review of the material presented in section 3.5. The important point is that the network only uses the training data to update weights and biases. It does, however, measure the error on reconstructing the validation data. If the error on the

validation data increases as the network is trained, then training is stopped: this is where the name *early-stopping* comes from – the network training is stopped after fewer iterations than it would be otherwise. Since the error on the training data always decreases as training progresses, stopping training in this way results in a model that doesn't fit the training data as well as it would otherwise, but it likely also stops the network from over-fitting the training data: the model will generalize more successfully to the validation data set. The validation data set should be extensive for this procedure to be effective. As an extra check on the validation data, the error on another set, called the testing data set can be recorded. Training is not affected at all by the testing data set, but if the error in reconstructing the testing set mimics the error in the validation set, this indicates the validation data has been well-chosen. This extra check should not be performed at the expense of generating a large validation set – it is more important to have a large validation set than medium sized validation and testing sets. Finally, the *testing data set* discussed in this paragraph and the *test grid data* discussed in chapter 2 are not exactly the same. The testing data set could use test grid data, but test grid data was generated with the intent of testing the network independently of the training process. This isn't a major distinction, although it is worth noting.

When the network is trained or tested, the error on reconstruction is measured. During training, the "error" on reconstructing on training data can be taken as the average of the squared errors of reconstructing each coordinate; this is called the *mean squared error* (MSE). If there are N points in the training data set, and the i^{th} network output is m_i, and the i^{th} target output (the i^{th} point in the training set) is t_i, then

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(m_i - t_i)^2 \tag{3.15}$$

Instead of using this measure for the training error, I can use the *regularized mean square error* (RMSE), where

$$RMSE = \gamma MSE + (1-\gamma)\frac{1}{Q}\sum_{i=1}^{Q}(w_i^2 + b_i^2) \tag{3.16}$$

given that there are Q weights and biases, and the i^{th} weight is w_i, while the i^{th} bias is b_i. If the network is trained using this measure for error, then the weights and biases will be smaller, which causes the network model to be smoother and possibly less prone to over-fitting [20].

γ is a constant referred to as the performance ratio, and it is usually determined by trial and error. The higher γ, the smoother the network model, but the more poorly training data is fit. There are algorithms that optimize γ to minimize the error on reconstructing the validation set [27, 28]. These algorithms are often tied to particular training paradigms, so I did not make use of them. Through trial and error, I found a performance ratio of 0.9 to provide the most accurate reconstruction for this problem when using the Levenberg-Marquardt training algorithm.

In figure 3.4 is a regularized neural network fit of the noisy sine data in figure 3.3. The fit is now intuitive – it represents an obvious pattern. It's also smooth, and more likely to successfully model new data points.

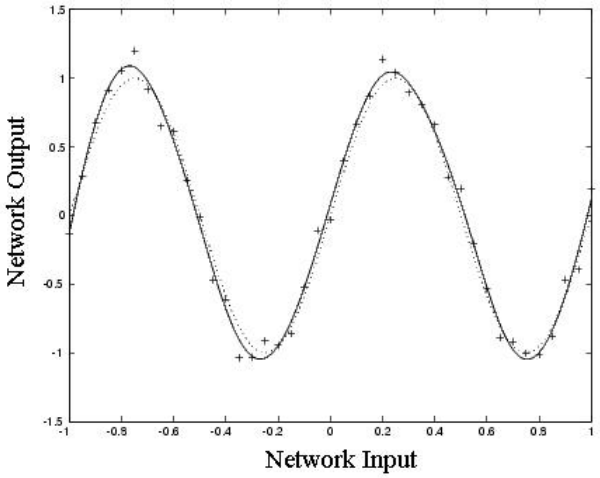

Figure 3.4: **Noisy sine curve fit with regularized neural network.** Noisy sine curve fit with a regularized neural network. Noisy data marked by crosses is fit by a regularized neural network. A dotted sine curve is shown in the background. The regularized fit is more intuitive, and more likely to generalize to new data points. Figure adapted from [20]

3.6 Training the Network

To recap, I am using a [32-26-17-23-16-3] network. It is a feed-forward back-propagation neural network, which uses the Levenberg-Marquardt training algorithm. All network layer outputs are determined by *weights*, *biases*, and *transfer functions*; a linear transfer function *l(w)=w* determines the final network output, but tansig transfer functions determine all other layer output. The network is trained using *regularized mean square error*, with a *performance ratio* of 0.9; this is a means to prevent over-fitting. The network also uses *cross-validation* with *early stopping* to prevent over-fitting.

As discussed in chapter 2, photons travel with an exponential attenuation described by

$$e^{-\frac{7}{9}\frac{L}{3.1}} \tag{3.17}$$

This means that likely at least 53% of the scintillation light reaching photo-detectors will come from the first three centimeters of the chamber. For this reason, I specialized my efforts in this region of the chamber. This way, I was more likely to realize the full potential of a neural-network solution, which could later be extended to the full chamber. It also allowed the network to provide superior reconstruction where superior reconstruction would be most beneficial. The smaller the region of interest, the easier it is improve reconstruction. It follows that the most time and effort should be devoted towards the regions of the chamber where accurate reconstruction is most needed.

Therefore the network presented here exclusively made use of training, validation and testing data generated in the first three centimeters of the chamber. With the coordinate system used in chapters 2 and 3.2, the "first three centimeters" means $z \in [-5.525, 2.525]$, while x and y can take any value within the chamber; this is a 3D cartesian coordinate system, where (0,0,0) is the centre of the chamber, as illustrated in figure 1.3.

The training data is comprised of a 22165 point, 0.23 cm spacing, 1 event grid, and 11660 points of a 33825 point, 0.2 cm spacing, 200 event grid. The remaining 22165 points of this 200 event grid were used as validation and testing data. Testing data in this context refers only as data used to check the validation data, not data exclusively used to test how accurately the network performs reconstruction – those tests are discussed and presented in the next section.

The training process is illustrated in figure 3.5. Training took approximately 2 hours. The error on reconstructing validation data consistently increased past 47 training iterations, and so training was stopped at this point. The error (y-axis) is on a logarithmic scale, for clarity. With this in mind, this figure displays the error on reconstructing validation and training sets as initially decreasing extremely rapidly. The error on reconstruction then decreases at a decreasing rate, until it appears almost asymptotic. The error on reconstructing the validation data and the test data closely mimic one another, which is a good sign. The reason the error on reconstructing the test data appears more erratic is simply because there aren't many points in this test data set – it is more important to generate a large validation data set than to have many test points. However, this data set does roughly follow the same pattern as the validation data, and has its minimum at the same place, which is reassuring. Whether or not the validation data has been well chosen is more rigorously tested in the next section.

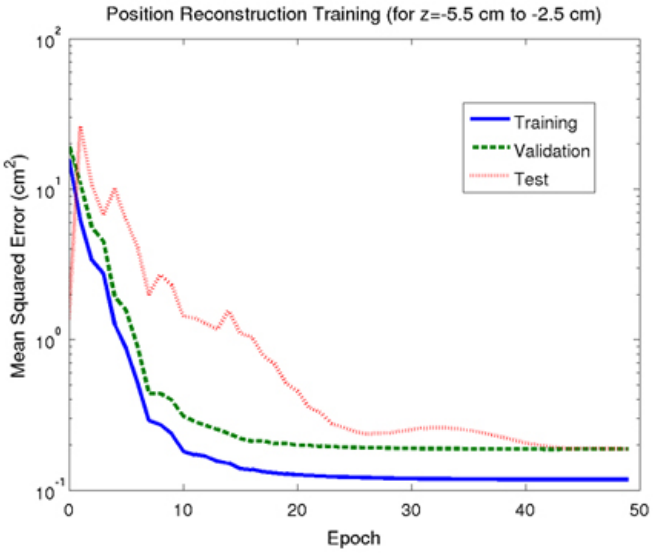

Figure 3.5: **Training the Network.** Training the neural network in the first three centimeters of the chamber. At 47 iterations, training is stopped, since the error on the validation data began to increase. In general, training and validation data error follow a similar pattern, and initially decrease rapidly. This decrease slows down, and the error appears to almost approach an asymptote.

The choice of training and validation data may seem unusual. Initially, I had trained the network on strictly multiple event data, and used strictly single event data for validation data. I reasoned that scaled multiple-event photo-detector data would represent something physically meaningful – the solid angle of each photo-detector from a point in the chamber – and it would be governed by a smooth one-one mapping. For these reasons, in addition to the reasons it was used for KNN, it seemed that multiple-event photo-detector data would be the most successful choice for training data. This isn't the case, perhaps because providing the network with some single event training data allows it to model the statistical variations we can expect from the input data (recall that input data generated at the same point will not be precisely the same from event to event). In the end, the network reconstructed new input most successfully with the above described mixture of single and multiple event training data, and multiple event validation data, of any configuration tested.

And many configurations were tested. I attempted dozens of configurations, with different validation data, training data, or a different number of neurons, or a different number of hidden layers, and with different performance ratios. This took about 90 hours (most networks took approximately 2 hours to train), and many of the configurations did not reconstruct nearly as well as "K Nearest Neighbours". On the other hand, the final configuration presented in this thesis and tested in the next section greatly outperformed "K Nearest Neighbours".

3.7 Network Results

The process of testing the neural network is as follows:

- Evenly distribute testing points in a XEPET chamber
- Simulate photo-detector data each of these points
- Use this photo-detector data as input to the network
- Compare the network output to the test points

I tested the network with six 3811 point, 0.41 cm spacing, single-event grids. Grids 1, 2, and 3, were generated for points that have 511 keV, 450 keV, and 300 keV of energy, respectively. To make grids 4, 5, and 6, noise was added to grids 1, 2, and 3. The added noise includes all factors discussed in chapter 2.4.3, except photo-electron conversion.

Reconstruction accuracy must be somewhat independent of how many scintillation photons are emitted – that is, the number of photons recorded by the detectors. This number depends on the energy of photons that enter the liquid xenon chamber and excite electrons, and the scintillation yield in liquid xenon. Values for the scintillation yield in liquid xenon vary greatly in the literature – by approximately 30% of the 40000 photons per MeV yield used in all simulations [9]. And the energy of photons entering the chamber will vary. These photons are initially imparted 511 keV of energy from a positron-electron annihilation event in the subject, but they may lose some of this energy from scattering in the patient or in the liquid xenon chamber. The energy of these photons entering the chamber will affect how many scintillation photons are emitted, and therefore how many photons are detected.

	90% (cm)	average (cm)	stdev (cm)
radial	1.062	0.6315	0.5086
x	0.809	0.3810	0.3399
y	0.496	0.2545	0.4227
z	0.641	0.2920	0.2609

Table 3.1: **Reconstruction statistics for 3811 particles (511 keV) with no noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured.

By testing reconstruction for particles with different energies, I am also testing how well the network can reconstruct data with different scintillation yields; for instance, simulating photo-detector data for particles with an energy of 300 keV and a scintillation yield of 40000 photons per MeV is the same as simulating photo-detector data for particles with 511 keV of energy and a scintillation yield of 23483 photons per MeV. What matters is how well the network can reconstruct photo-detector data with different numbers of scintillation photons.

Energy reconstruction, discussed in chapter 4, also uses position reconstruction. So a source of the error in energy resolution will be from reconstructing the position of a particle at some unexpected energy. This is another good reason to document how well the network can reconstruct the positions of particles with different energies. However, the source of error in energy reconstruction from position reconstruction is exaggerated in chapter 4, since I use position reconstruction from scintillation light alone, and not the fine position measurement we will get from charge collection. So the error from position reconstruction is considered in chapter 4 mostly to make some qualitative observations.

Finally, the results of reconstructing data with no noise are of particular interest if we wish to use photo-tubes instead of Avalanche Photo-diodes. With photo-tubes, most of the noise introduced would be eliminated.

In the following six sub-sections, I present the results of reconstructing each of these six grids with minimal comments: they serve as a direct presentation of the data. Tables and figures designed to evaluate the same reconstruction properties are given in each section, and therefore their descriptions are near identical. In the following section I comment on these results; what has to be written about each individual reconstruction is most meaningful in the context of having presented all reconstructions. Due to the repetitive nature of these subsections, I considered presenting some of their material in an appendix. I decided against this as the figures and tables shown are central to the proceeding discussion.

3.7.1 511 keV, no noise

Table 3.1 lists statistics for reconstructing a 3811 point, 0.41 cm spacing, 511 keV, single event, no noise data set. The radial error is the Euclidean 3 space distance between the network output and what it would be if it were 100% accurate. Error on reconstructing each spatial dimension is taken as the absolute value of the difference of what the network output with what it would be if it were 100% accurate.

One useful interpretation of the above table is that we can likely expect to reconstruct particles with 511 keV (when there is no APD noise) to within an ellipsoid with volume $\frac{4}{3}\pi(.809)(.496)(.641)$ $cm^3 = 1.07$ cm^3, *approximately* 90% of the time.

Figure 3.6 encapsulates many of the statistics in the above table. An interpretation of figure 3.6 is that 96% of photo-detector data from this sample can be reconstructed to within a sphere of radius 1.1 cm.

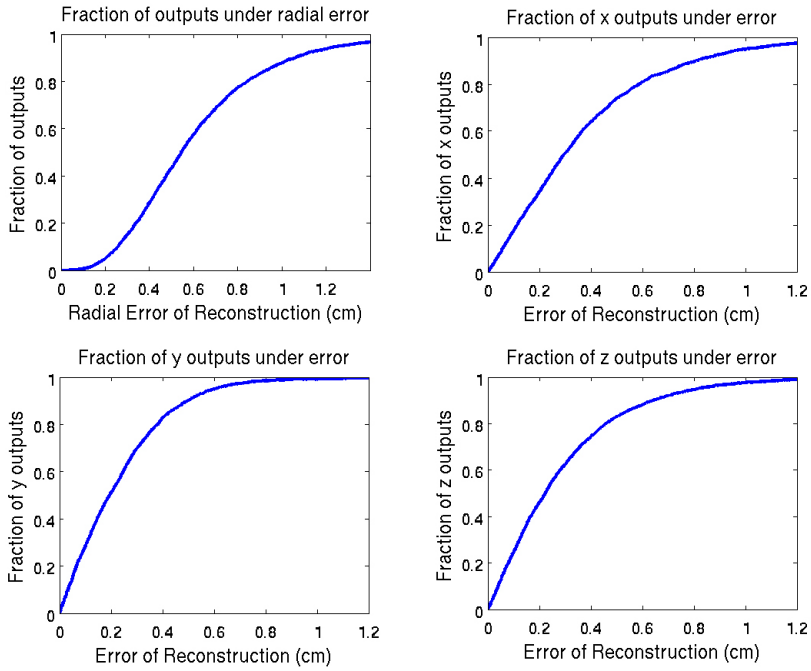

Figure 3.6: **Reconstruction error on 3811 single event, no-noise, testing points with 511 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

Figure 3.7 illustrates a regression on how well the x, y, and z spatial coordinates were reconstructed. It also provides a means to visualize how the variances in reconstructing coordinates changes as position in the chamber changes.

Figure 3.8 shows the radial error in reconstruction as a function of position in the chamber.

3.7.2 511 keV, noise

The same analysis presented in the last section is applied here, to the same testing data, except noise is added. All of the noise described in chapter 2.4.3 is added, except for noise from photo-electron conversion. Table 3.2 lists reconstruction statistics.

As seen in figure 3.9, noise clearly affected construction for the worse.

In figure 3.10, it is apparent that noise has greatly increased the variance in the error of reconstructed data at a given position; of course, this is expected.

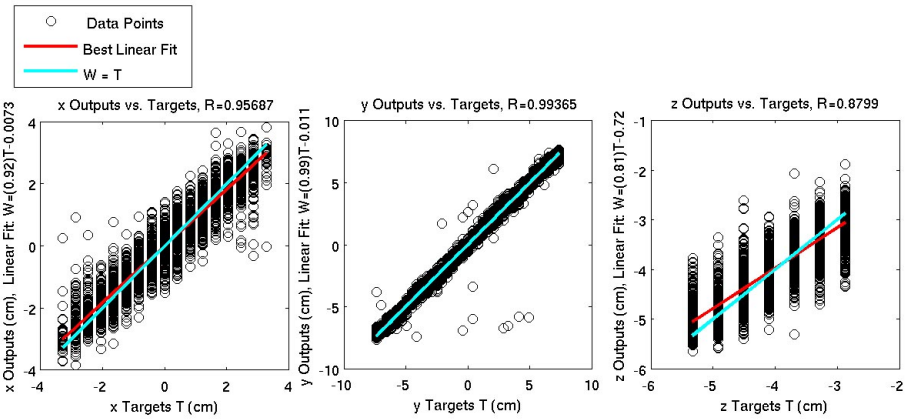

Figure 3.7: **Reconstructed spatial coordinate outputs as a function of target coordinate outputs for 3811 single event, no-noise, testing points with 511 keV of energy.** The y-axes represent reconstructed spatial outputs, and the x-axes represent target spatial outputs. The hollow circles represent data points. The red line is the best-linear fit through these points, and the blue line is the ideal fit, outputs = targets.

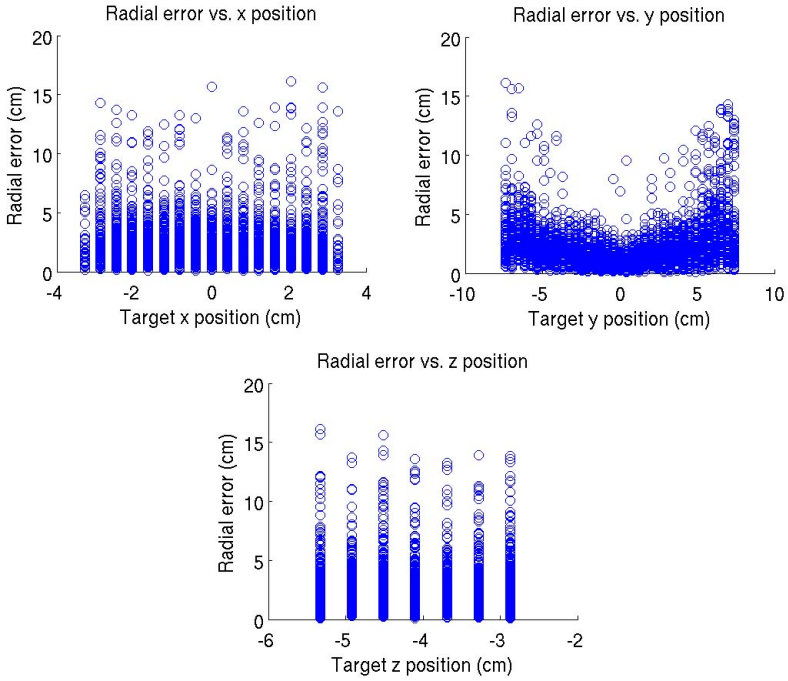

Figure 3.8: **Radial error vs. position**. Radial error on reconstruction versus spatial coordinate in the chamber for 3811 single event, no-noise, testing points with 511 keV of energy. The y-axes are the overall radial error of reconstruction for a specified coordinate in the chamber, on the x-axes.

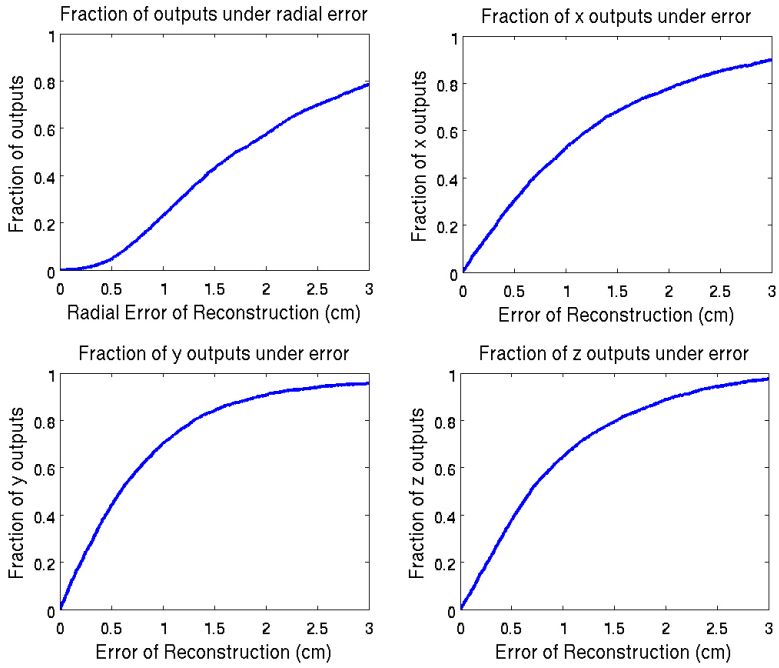

Figure 3.9: **Reconstruction error on 3811 single event, noisy testing points with 511 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

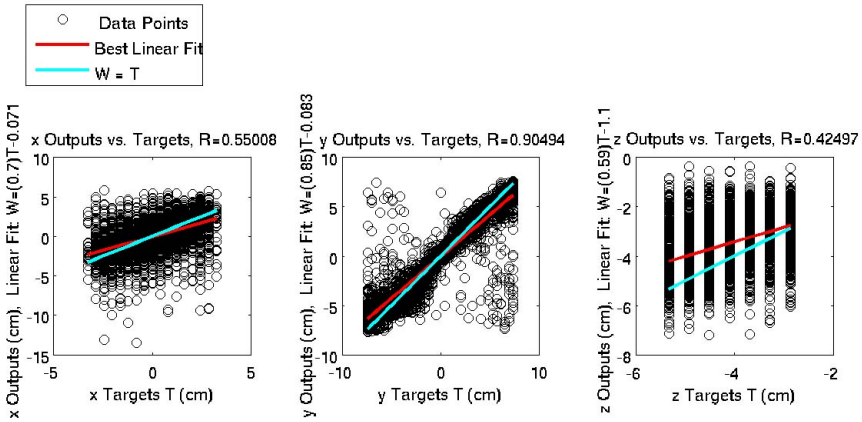

Figure 3.10: **Reconstructed spatial coordinate outputs as a function of target coordinate outputs for 3811 single event, noisy testing points with 511 keV of energy.** The y-axes represent reconstructed spatial outputs, and the x-axes represent target spatial outputs. The hollow circles represent data points. The red line is the best-linear fit through these points, and the blue line is the ideal fit, outputs = targets.

	90% (cm)	average (cm)	stdev (cm)
radial	$*^1$	2.2474	1.9232
x	2.996	1.3516	1.3868
y	1.937	1.0005	1.5795
z	2.095	0.9300	0.7992

Table 3.2: **Reconstruction statistics for 3811 particles (511 keV) with noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured. [1] Not measured. See figure 3.9 for an indication of the fraction of output under a given error.

Adding noise least affects the accuracy of reconstructing the y coordinate, and the y-coordinate in particular is not significantly affected by noise in the centre of the chamber.

Figure 3.11 shows the radial error of reconstructing coordinates as function of position in the chamber. As expected, there is greater variance between points for the radial reconstruction error at a given position, with noise, than without noise. This is last time this type of figure is displayed, as it doesn't reveal anything substantially new for reconstructions on points with 450 keV or 300 keV of energy.

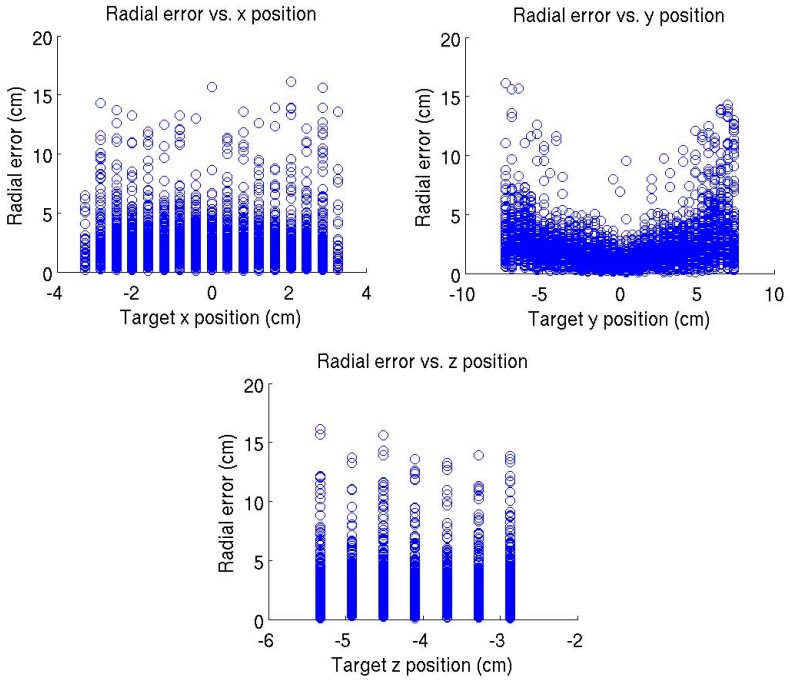

Figure 3.11: **Radial error vs position**. Radial error on reconstruction versus spatial coordinate in the chamber for 3811 single event, noisy testing points with 511 keV of energy. The y-axes are the overall radial error of reconstruction for a specified coordinate in the chamber, on the x-axes.

	90% (cm)	average (cm)	stdev (cm)
radial	1.263	0.7950	0.5267
x	0.879	0.4239	0.3694
y	0.653	0.3235	0.4461
z	0.862	0.4321	0.3208

Table 3.3: **Reconstruction statistics for 3811 particles (450 keV) without noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured.

3.7.3 450 keV, no noise

Table 3.3 lists statistics for reconstructing a 3811 point, 0.41 cm spacing, 450 keV, single event, no noise data set. The radial error is the Euclidean 3 space distance between the network output and what it would be if it were 100% accurate. Error on reconstructing each spatial dimension is taken as the absolute value of the difference of what the network output with what it would be if it were 100% accurate.

Figure 3.12 displays the now familiar curves for the fraction of reconstruction outputs with less than a given error. Thankfully, changing the energy of the test particles from 511 keV to 450 keV does not dramatically affect the error on reconstruction.

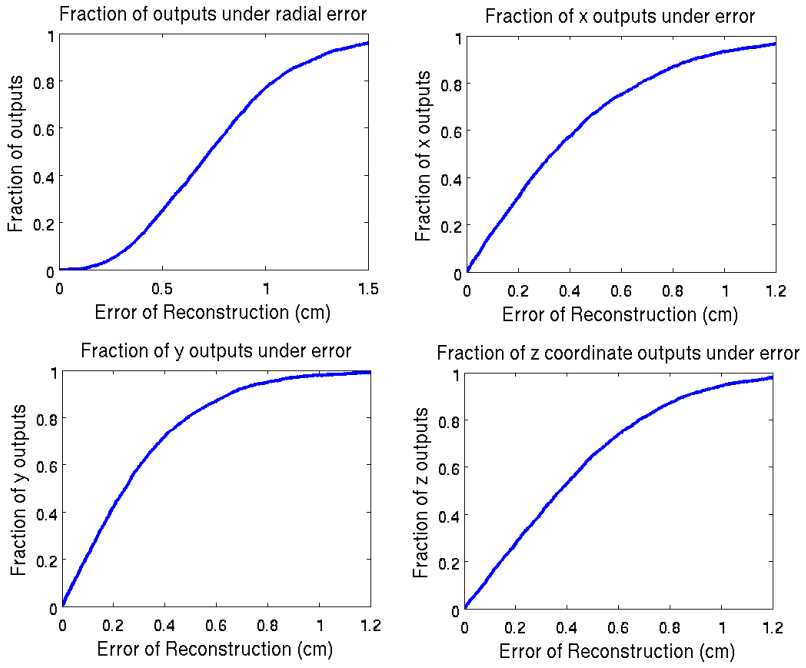

Figure 3.12: **Reconstruction error on 3811 single event, no-noise, testing points with 450 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

	90% (cm)	average (cm)	stdev (cm)
radial	$*^1$	2.3946	2.1578
x	2.99	1.4284	1.4673
y	2.128	1.144	1.8794
z	1.959	0.8815	0.7607

Table 3.4: **Reconstruction statistics for 3811 particles (450 keV) with noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured. [1] Not measured. See figure 3.14 for an indication of the fraction of output under a given error.

Finally, Figure 3.13 displays the now familiar regression of reconstructed spatial outputs versus target spatial outputs. As usual, the y-coordinate is reconstructed most successfully. Decreasing the energy has somewhat of a "noise" effect on the data, causing the variance in the error of reconstruction at a particular point in the chamber to increase. This is expected, since with fewer scintillation photons, statistical variations in photo-detector data generated at the same point in the chamber will increase. This photo-detector data is therefore more difficult to model.

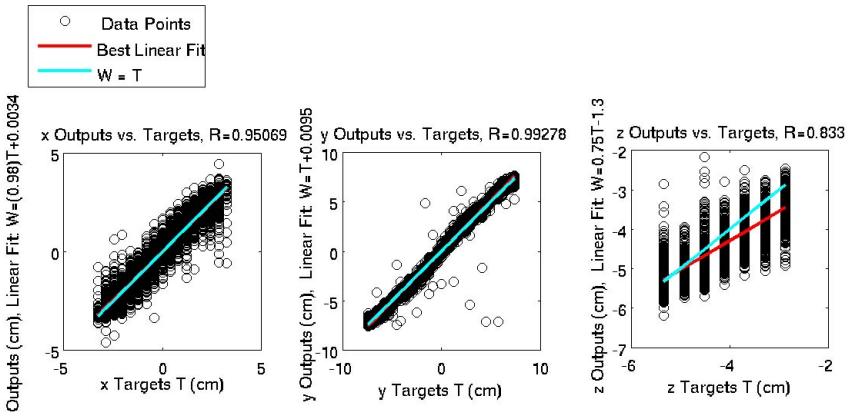

Figure 3.13: **Reconstructed spatial coordinate outputs as a function of target coordinate outputs for 3811 single event, no noise testing points with 450 keV of energy.** The y-axes represent reconstructed spatial outputs, and the x-axes represent target spatial outputs. The hollow circles represent data points. The red line is the best-linear fit through these points, and the blue line is the ideal fit, outputs = targets.

3.7.4 450 keV, noise

Table 3.4 lists statistics for reconstructing a 3811 point, 0.41 cm spacing, 450 keV, single event, noisy data set. The radial error is the Euclidean 3 space distance between the network output and what it would be if it were 100% accurate. Error on reconstructing each spatial dimension is taken as the absolute value of the difference of what the network output with what it would be if it were 100% accurate.

Figure 3.14 presents no surprises: the fraction of outputs under a given error is similar but somewhat less than it was for 511 keV points, with noise.

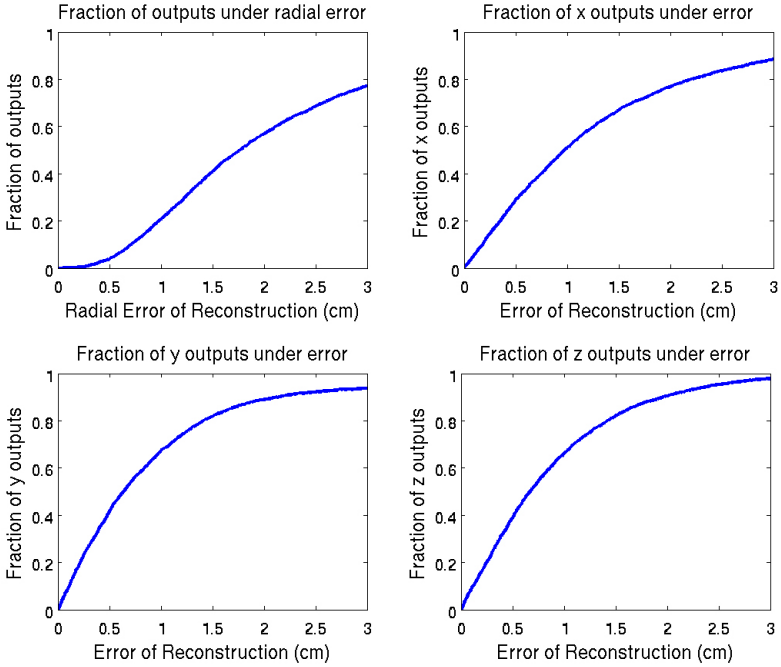

Figure 3.14: **Reconstruction error on 3811 single event, noisy testing points with 450 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

And the familiar regression pictures in Figure 3.15 are not surprising either: similar figures will not be shown in future sections.

3.7.5 300 keV, no noise

The energy of the simulated testing points is now significantly lower than the 511 keV energy initially imparted to a photon entering a XEPET chamber.

Table 3.5 lists statistics for reconstructing a 4811 point, 0.41 cm spacing, 300 keV, single event, no-noise data set. The radial error is the Euclidean 3 space distance between the network output and what it would be if it were 100% accurate. Error on reconstructing each spatial dimension is taken as the absolute value of the difference of what the network output with what it would be if it were 100% accurate.

Figure 3.16 shows the fraction of reconstructed outputs under a given error. Reconstruction is still surprisingly effective, even with a 40% reduced quantity of scintillation photons.

3.7.6 300 keV, noise

Since there are roughly 40% fewer scintillation photons for points with 300 keV than for points with 511 keV, noise has a much greater effect on the 300 keV data; this is because some of the noise is independent of how many photons are recorded at the detectors, and therefore accounts

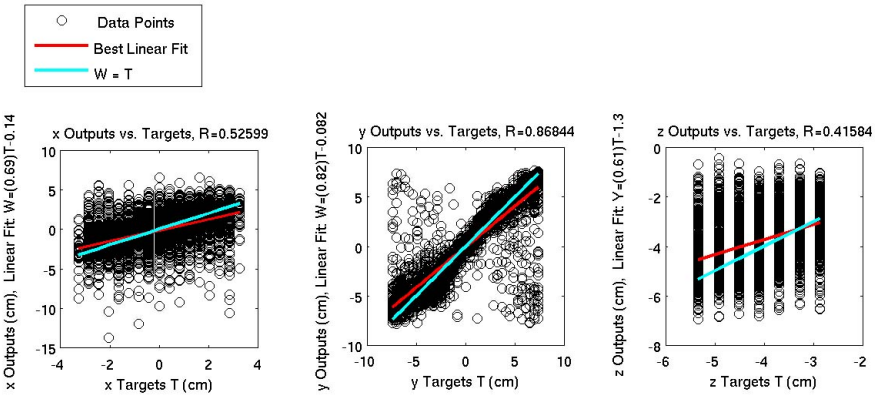

Figure 3.15: **Reconstructed spatial coordinate outputs as a function of target coordinate outputs for 3811 single event, noisy testing points with 450 keV of energy.** The y-axes represent reconstructed spatial outputs, and the x-axes represent target spatial outputs. The hollow circles represent data points. The red line is the best-linear fit through these points, and the blue line is the ideal fit, outputs = targets.

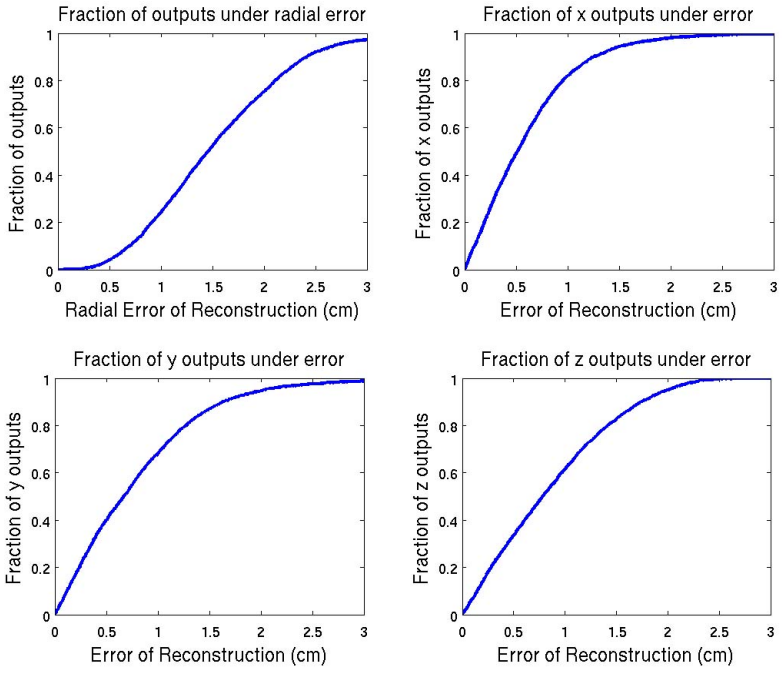

Figure 3.16: **Reconstruction error on 3811 single event, no noise testing points with 300 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

	90% (cm)	average (cm)	stdev (cm)
radial	2.418	1.5500	0.7855
x	1.249	0.6180	0.5143
y	1.632	0.8205	0.7613
z	1.743	0.8707	0.6016

Table 3.5: **Reconstruction statistics for 3811 particles (300 keV) without noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured.

	90% (cm)	average (cm)	stdev (cm)
radial	$*^1$	3.5411	2.8744
x	$*^2$	1.6477	1.7048
y	2.997	2.2313	2.8736
z	2.272	1.1414	0.8005

Table 3.6: **Reconstruction statistics for 3811 particles (300 keV) with noise.** 90% of outputs are under the specified error in the second column. The third column is the average error in reconstruction. The fourth column is the standard deviation in reconstruction error. Each row specifies what type of reconstruction error is being measured. [1] Not measured. [2] Not measured. See figure 3.17 for an indication of the fraction of output under a given error.

for a greater proportion of the total number of photons detected from points with less energy. This is apparent in table 3.6 and figure 3.17.

3.8 Discussion

First of all, the particular neural network presented in sections 3.3-3.8 greatly outperforms the K Nearest Neighbour (KNN) implementation in section 3.2. In the first three centimetres of the chamber, where reconstruction is most important, the radial reconstruction error is on average 317% greater when using KNN than when using the neural network, with similar training data and testing data. KNN suffers most in this comparison when reconstructing data with noise or with energies different from 511 keV: this is because scaling training and testing data substantially deteriorates the quality of KNN reconstruction.

The neural network also reconstructs input more rapidly than KNN. With a grid of 11687 training points, KNN takes approximately 0.2 seconds to reconstruct new input; at best, a fully optimized KNN implementation would take between 0.01 and 0.001 seconds to reconstruct new input with a 11687 point grid. With KNN, this reconstruction time increases linearly with grid size. On the other hand, the neural network reconstructs input in microseconds, regardless of the training grid size.

This is a reasonably fair comparison. The KNN implementation presented in this thesis uses a 0.5 cm training grid throughout the chamber, while the neural network presented uses an (approximately) 0.2 cm grid training grid in the first three centimetres. One might therefore feel

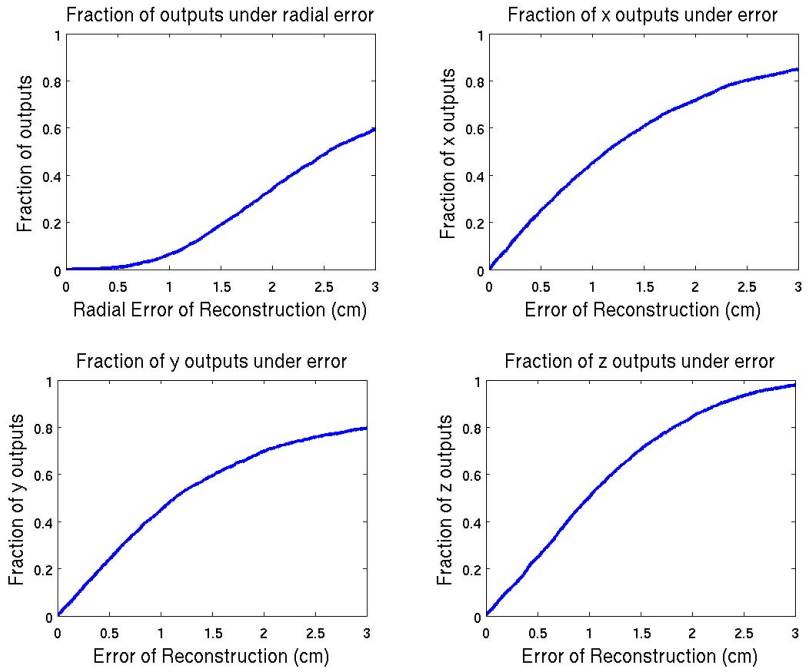

Figure 3.17: **Reconstruction error on 3811 single event, noisy testing points with 300 keV of energy.** The y-axes represent a fraction of outputs (percentage over 100) that have an error less than that specified on the x-axes.

that I am biased towards neural networks and object to my comparison on the grounds that I trained the network on different data than I used as training data for KNN. In fact, if I exclusively use the 0.5 cm multiple event grid as training data for both KNN and the network, K Nearest Neighbours provides slightly better radial reconstruction accuracy (about 5%) on tests where no noise has been added. But KNN would provide much worse reconstruction using the same 0.2 cm training grid the neural network uses: the "KNN" algorithm does not model extra statistical variations present in single event photo-detector data, these variations just deteriorate reconstruction accuracy. Indeed, comparing the "KNN" algorithm with neural network reconstruction using identical training data is as senseless as measuring the fuel efficiency of an aircraft by driving it across the continent. The point is that the particular neural network presented in sections 3.3-3.8 substantially outperforms the K Nearest Neighbour algorithm presented in 3.2, when using suitable training sets for each. It is possible to improve the success of both algorithms by generating new training data, but it is most practical to use the neural network solution.

This neural network solution reveals patterns common to the tables and figures presented in sections 3.8.1 through to 3.8.6. First, the y spatial coordinate reconstructs most successfully – it is generally reconstructed with less error than the other spatial coordinates, its reconstruction error is less effected by noise or decreased energy than the other spatial coordinates, and the network y-coordinate outputs correlate more closely with y-coordinate targets than x or z coordinate outputs correlate with x or z targets (as seen from the R values in figures 3.10, 3.13, and 3.15). This is particularly notable since the y-spatial dimension is the largest of the three, varying from -7.5 cm to 7.5 cm, as opposed to the x-dimension which varies from -5.561 cm to 5.561 cm, and

the z dimension, which varies from -5.525 cm to 5.525 cm (-5.525 cm to 2.525 cm for the first three centimetres); here I am using the familiar coordinate system where (0,0,0) is the centre of the chamber. There is also less variance in the reconstruction error for the y coordinate than for the other spatial coordinates.

Another important pattern emerges in the figures presented in the previous section: although the radial reconstruction error is at a minimum in the centre of the chamber, it is not much different at other places in the chamber. Thus reconstruction is not highly dependent on position.

Also, changing the energy of test particles does not affect reconstruction accuracy as much as I had expected. The average radial error of reconstruction changes from 0.63 cm for test points with 511 keV and no noise, to 0.795 cm for test points with 450 keV, no noise, and at the same location. Essentially, a 12% decrease in the number of photons detected results in a 26% increase in error, and for practical purposes, a radial error of 0.63 cm is not too different from a radial error of 0.795 cm. On the other hand, the average radial error for reconstructing particles with 300 keV of energy is 1.55 cm. So a 41% decrease in the number of photons results in a 246% increase in error. Therefore the error on reconstruction increases at an increasing rate as the number of photons detected decreases; this is intuitive – as the number of detected photons approaches zero, accurate reconstruction becomes impossible. With only a few photons detected, there is no obvious pattern for the neural network to associate with. Reconstruction with a 1.55 cm radial error is still useful, however.

On the other hand, data generated at lower energies is more negatively affected by noise than data generated at higher energies. This is expected: the lower the number of photons detected, the more adding a constant noise factor will deteriorate reconstruction. Not all of the noise is constant – some depends on the number of photons detected – but some of it is (see chapter 2). Future studies may wish to test reconstruction at even lower energies (partly to test lower scintillation yields).

On an important final note, radial error is used mostly to present my results succinctly. It is also useful to know the volume with which reconstruction will take place a given percentage of the time. For instance, using the values given in the tables of the last six subsections, 511 keV no noise data is located in an ellipsoid of volume 1.07 cm^2, *approximately* 90% of the time (a product of three 90% values are used in this calculation, but data will likely be in this ellipse more than $0.9^3 * 100$ percent of the time). An endless list of other values that may be of interest can also be calculated using the tables I presented.

3.9 Suggestions for Improvement

Other machine learning paradigms could be used to approach this problem. In a 2006 textbook, "Gaussian Processes for Machine Learning", Rasmussen and Williams detail a case study where Gaussian Process Regression is used to learn the "inverse dynamics of a seven degrees-of-freedom SARCOS anthropomorphic robot arm" [29]. In this case the mapping is from $\mathbb{R}^{21} \rightarrow \mathbb{R}^7$, as opposed to the $\mathbb{R}^{32} \rightarrow \mathbb{R}^3$ mapping in this reconstruction problem; (once position reconstruction from scintillation light is fully implemented, it would make for a valuable example in machine learning texts or journal articles – this is a relatively new and quickly growing field!). In the example, seven joint positions, velocities and accelerations are mapped to 7 joint torques. This example was also used to study other regression algorithms [30, 31, 32].

Support Vector Regression (a type of Support Vector Machine) is becoming a popular choice for multivariate regressions. I considered using it for this problem, but as of February 2008, there were no popular implementations that allowed the photo-detector data to be highly correlated to more than one output (in this case there are three outputs). The most well-documented and developed implementation of SVR I found is called "LIBSVM" [34]. I e-mailed the author, and he suggested that a future release would implement the facility to correlate input with multiple outputs in regression problems. For further reading on Support Vector Machines, one can refer to John Taylor and Nello Cristianini's book, "Support Vector Machines and other kernel-based learning methods" [33]. I suggest instead starting with the Wikipedia article "Support Vector Machines", and reviewing the references listed on that page.

On the other hand, exploring other machine learning approaches to this problem may not be extremely productive. Reconstruction can only be performed up to a certain accuracy. If a number of photons is sent in random directions at a given point in the chamber at one time, and a number of photons is sent in random directions at the same point in the chamber at another time, data at the photo-detectors will be different. Given this statistical variation, and the variation in the number of photons being detected by photo-detectors for any given point in the chamber, I expect reconstruction to have no better than a 0.3 cm average radial accuracy (for no noise 511 keV input). This estimate is based on the tests I described in Chapter 2, where I sent a typical number of photons off in random directions, and measured the average variations in photo-detector data at the same point, and for several nearby points. This isn't a rigorous estimate, but if it is in error, then I suspect my estimate is too optimistic, since these tests took place mostly in the centre of the chamber, where reconstruction appears to take place slightly more accurately (e.g. 5 to 10%) than in the most difficult regions of the chamber. This 0.3 cm average radial error is only 0.3315 cm away from the 0.6315 cm average radial error currently measured by the network (for no noise 511 keV input).

And this is a very optimally performing network – most of the sensible network configurations I tested had a 1.9 cm radial accuracy for reconstructing this (511 keV no noise) data. So I don't expect reconstruction on the noiseless testing grid data used in sections 3.8.1, 3.8.3, or 3.8.5, to improve by more than 0.1 cm in radial accuracy. Reconstruction could be improved to a greater extent on the noisy data in sections 3.8.2, 3.8.4, and 3.8.6: I estimate the radial error on reconstruction in the sections could be improved by up to 0.7 cm, and at least by 0.4 cm. This expected increase is greater because with noisy data there is more freedom in the composition I can choose for my training and testing data. Based on some preliminary work (as of April 12, 2008), I am quite confident that such increases in accuracy will be realized.

Generally, the accuracy of my network can be increased by generating new grid data (with a smaller grid spacing), and testing new training configurations. This will likely be more immediately productive than designing a model based on a completely different paradigm, although it would be interesting to see the results of a "Support Vector Machine" approach once this algorithm is more widely implemented.

Chapter 4

Energy Reconstruction

4.1 Introduction

In this short chapter, I briefly examine energy reconstruction from scintillation light; the subject of this thesis is primarily position reconstruction. As discussed in the first chapter, measuring both energy from scintillation light and energy from charges collected at the top of a XEPET chamber results in better energy resolution than if only one or the other measurement were used. To recap, having good energy resolution (percent error on energy detected) is important because it allows us to suppress data from photons which have scattered within the patient (thereby losing some of their energy). These scattered photons contribute poorly to data used for constructing lines of response, which are used to produce PET images.

Since the scintillation yield is directly proportional to the energy of a test particle (at least as far the Monte-Carlo simulations used in chapter 2 are concerned), then it is intuitive that the number of photons recorded by photo-detectors scales linearly with the energy that went into producing those photons. More specifically, if the photo-detectors record N photons from a 511 keV particle, it seems as good a guess as any that the photo-detectors would record $300/511 * N$ photons from a 300 keV particle in the same position.

This guess is tested in the next section. In the following section, I first reconstruct the position of a particle from scintillation light, and then reconstruct its energy from that reconstructed position. This will result in greater error than expected in practice, due to the lack of fine position reconstruction from charge collection. But it serves to make some qualitative observations.

All grids discussed in this section are within the first three centimetres of the chamber (z=-5.5 cm to z=-2.5 cm). Any noise discussed in this chapter refers to the noise for APDs discussed in chapter 2, with no photo-electron conversion noise.

4.2 Energy Reconstruction

I generated a 3811 point grid (0.41 cm spacing), for particles with 511 keV energy, using the average results of 100 simulations. Then to test energy resolution, I used a grid with same number of points in exactly the same locations, except with single event data, and 300 keV of energy. In both cases, I used a 40000 photons/MeV scintillation yield.

To make an estimate on the energy of a (300 keV) particle from the single event grid, I compared the absolute value of the sum of the number of photons detected for that particle, a, with the sum of the photons detected for a particle in the same position on the average 100 event 511 keV grid, w. My estimate of the energy for this particle was $a/w * 511 = y$. I then took the absolute value of the difference between the this estimate and the actual energy of the particle, 300 keV: $m = |y - 300|$. The percentage error on energy detected was then taken as $m/300$.

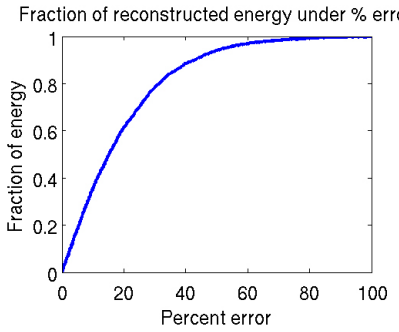

Figure 4.1: **Energy reconstruction error on 3811 single event, no noise, testing points with 300 keV of energy.** The y-axes represent a percentage of energy outputs that have an error less than the percentage specified on the x-axes.

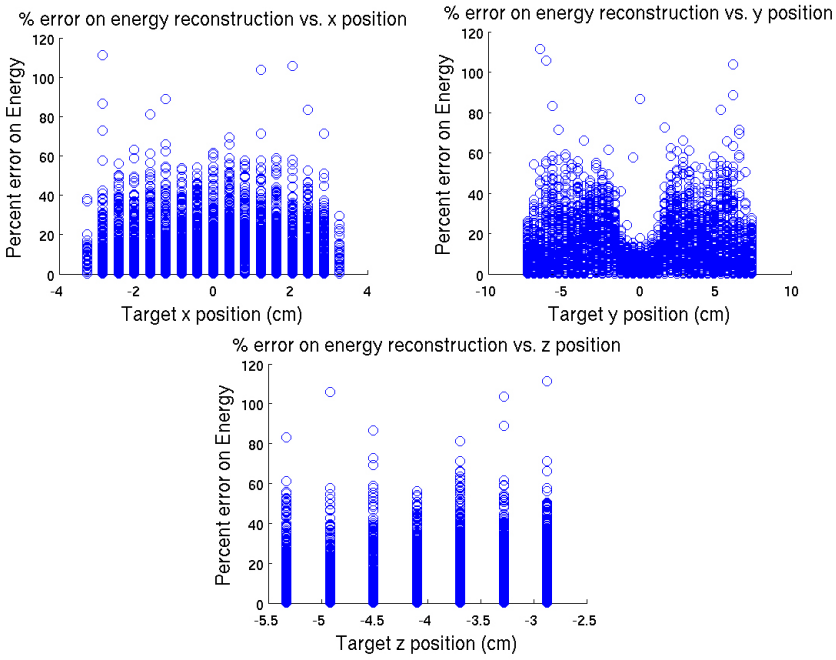

Figure 4.2: **Energy Reconstruction Error as a function of Position (300 keV no noise).** The y-axes represents the percent error in energy reconstruction as a function of position in the chamber (x-axes). Error is noticeably smaller when near y=0.

The reason I used the absolute value of the total photons detected for a 300 keV particle is because I also tested energy resolution by adding the same type of noise to the photo-detector data I added in sections 3.8.2, 3.8.4, and 3.8.6. It is therefore possible (though very unlikely – e.g. close to a 5% probability) for "negative" photons to be recorded by a detector in any one of these simulations. (So the percent error on energy reconstruction from this simulated noise is marginally greater than it would be in practice – on average about 1% greater, based on preliminary tests).

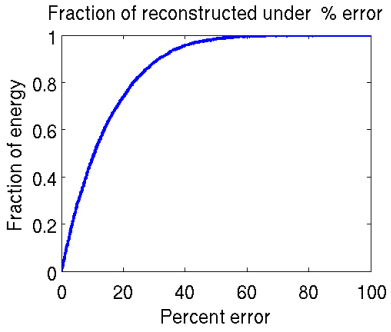

Figure 4.3: **Energy reconstruction error on 3811 single event, noisy testing points with 300 keV of energy.** The y-axes represent a percentage of energy outputs that have an error less than the percentage specified on the x-axes.

Following this procedure, I found the average error on reconstructing the energy of noiseless 300 keV particles to be 42.6 keV (14.2% error). The standard deviation on this reconstruction is 57.1 keV. The fraction of particles with a reconstructed error under a given percentage is shown in figure 4.1. In figure 4.2, I show the percentage error in energy reconstruction as a function of position in the chamber.

The average error on reconstructing the energy of 300 keV particles (with noise added to photo-detector data) is 58.9 keV (19.6%). The standard deviation on this reconstruction is 77.9 keV. These two numbers are both larger than their counter-parts for no-noise data, as expected. Figures 4.3 and 4.4 are the counterparts to figures 4.1 and 4.2, for photo-detector data which has had the APD noise (discussed chapter 2) added to it.

There are four notable observations to make. First, the average error on reconstructed energy is much smaller than the difference between the energies of the two grids used for reconstruction – regardless of whether or not there is noise added to the photo-detectors. 42.6 keV and 58.9 keV are both much smaller than 211 keV (= 511 keV - 300 keV). Also, the error in reconstructing the energy of the y coordinate is noticeably smaller near the center of the chamber. And, adding noise does not substantially affect the average accuracy of energy reconstruction (e.g. 42.6/211 = 0.2, while 58.9/211 = 0.28). Finally, similar statements can be made about any given fraction of inputs below 90%, as seen in figures 4.1 and 4.3.

4.3 Position then Energy Reconstruction

In this section, I first reconstruct the position of particles from a 3811 point (0.41 cm spacing), single event, 300 keV grid using photo-detector data. I then reconstruct the energies of these

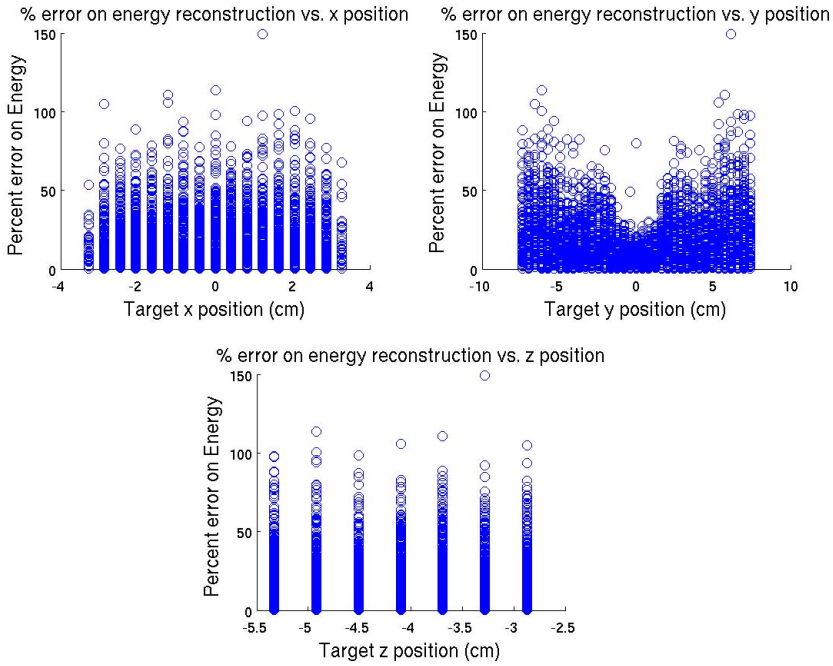

Figure 4.4: **Energy Reconstruction Error as a function of Position (300 keV noise).** The y-axes represents the percent error in energy reconstruction as a function of position in the chamber (x-axes). Error is still noticeably smaller when near y=0.

particles by finding the closest points on a 33825 point, 200 event, 511 keV grid. I then compare photo-detector in exactly the same way as in the last section.

This will be only useful for making a few qualitative observations – the error in these results will be much higher than expected in practice, since I am only reconstructing position from scintillation light. With the fine position measurement from charge collection, the error in energy reconstruction will be similar to what was presented in the previous section.

Adding noise does not add value to this particular qualitative discussion – the observations are the same. Without noise, the energy of this grid was reconstructed with an average 59% error, and 34% standard deviation. These numbers are large (as expected per the above discussion) but something quite interesting is happening to this error as a function of the y coordinate position in the chamber, as shown in figure 4.5. The error in energy reconstruction for a y coordinate near the centre of the chamber is relatively small (on average at most 20%). This error sharply increases away from the origin and then decreases in a linear fashion to a minimum at y = -4 cm and y = 4 cm. It then increases in a linear fashion in the same way. For most y positions, the spread in the error of energy reconstruction is quite small (between 10% and 20%). Since the error increases and decreases in a systematic fashion (as a function of the y position in the chamber), I could model these increases and decreases, to provide an average error of about 20% (30% respectively with noise), using position reconstruction from scintillation light (for 300 keV particles).

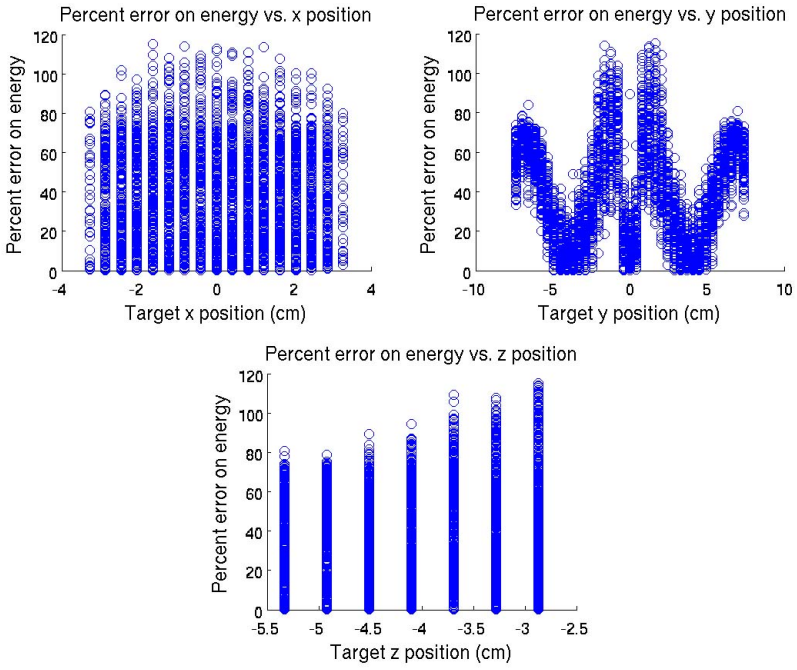

Figure 4.5: **Energy Reconstruction Error (after position reconstruction) as a function of Position (300 keV no noise).** The y-axes represents the percent error in energy reconstruction as a function of position in the chamber (x-axes). Energy error appears to vary with y coordinate position in a systematic way.

Chapter 5

Summary

In this thesis, I have developed and tested methods to localize the positions and energies of particles in a liquid xenon gamma ray detector designed for PET. This position and energy localization is done strictly through modeling the scintillation light emission that follows when photons enter liquid xenon chambers and interact with electrons.

Assuming no error from position reconstruction, the average accuracy of energy reconstruction from scintillation light is expected to be less than 14% (20% with APD noise) when the energy of particles is greater than 300 keV. Specific information to make confidence intervals is found in figures 4.1 and 4.3. Overall, this energy reconstruction from scintillation light is likely accurate enough to provide better energy resolution in combination with charge collection than if energy measurements from charge collection were used alone. To improve energy resolution, it is worth experimenting with having white reflectors on the slanted walls of the chamber; this will improve light collection, but will probably also deteriorate position reconstruction.

The y spatial coordinate (see fig 1.3) is particularly interesting. It provides useful information on energy reconstruction where the other spatial coordinates do not. For instance, energy reconstruction is most successful when the y coordinate is within a few centimeters of $y = 0$ – this is clear from figures 4.4 and 4.2. However, these figures do not show anything of great interest as far as reconstructing the energy of the x and z coordinates go (except, perhaps, that this energy resolution may be independent of distance). The y coordinate also easily allows one to salvage energy reconstruction when first reconstructing position from scintillation light; this may also be the case when using charge collection for position reconstruction.

And even though a XEPET detection chamber is longest in the y direction, the y spatial coordinate of a particle can generally be localized to within a smaller range than the other coordinates. The accuracy of reconstructing the position of the y coordinate is also least affected by noise.

I primarily approached position reconstruction with two methods: K Nearest Neighbours and Neural Networks. The neural network configuration was ultimately more successful, and therefore was more thoroughly tested. The network presented in this thesis was developed for use to within the "first three centimeters" (z=-5.5 to z=-2.5 cm in fig 1.3) of a XEPET detection chamber: this is where the majority photon-electron interactions take place.

Overall, based on (though not limited to) the results presented in Chapter 3, I expect for 90% of photo-detector input, neural networks can be used to reconstruct the position of a particle (anywhere in a XEPET chamber) from scintillation light

- To within a sphere with radius less than 2 cm for particles with less than 300 keV
- To within a sphere with radius less than 1 cm for particles within 40 keV of 511 keV
- To within a volume of 1 cm^3 for particles within 40 keV of 511 keV

When typical APD noise is added to photo-detector data, the radial error in these figures will increase by approximately 1.2 cm (once reconstruction with the network is fine-tuned). However, "noiseless" data is also of interest, since most of the APD noise is removed when photo-tubes are used. Also, I expect that position reconstruction accuracy is essentially independent of position in the chamber, given the results presented in Chapter 3.

Finally, I have shown how neural networks can be useful in subtle multivariate regression problems: the application of neural networks to the consequential problem in this thesis would be of interest to the machine learning community. However, I hope my work will help others, in any field whatsoever, who are wishing to perform their own regressions.

More specifically, I have demonstrated how the positions and energies of particles can be reconstructed from scintillation light. This makes possible a promising new liquid xenon PET device, which allows for great advances in medical research and diagnosis.

Bibliography

[1] T.J. Ruth, *Nuclear Physics Review* **16(3)**, 31-33 (2006)

[2] V. Sossi, *Nucl. Instr. Meth.* **510(1-2)**, 107-115 (2003)

[3] See e.g., J. Wang and L. Maurer, *Current Topics in Medicinal Chemistry* **5(11)**, 1053-1075 (2005)

[4] C. Levin and E. Hoffman *Phys. Med. Biol.* **44(3)**, 781-799 (1999)

[5] P. Colombino, B. Fiscell, and L. Trossi. *Nuovo Cimento* **38(10)**, 707-723 (1965)

[6] University of Washington, Division of Nuclear Medicine. "Introduction to PET Physics". http://depts.washington.edu/nucmed/IRL/pet_intro

[7] M. Korzhik et. al, *Nucl. Instr. Meth. A* **271(1-2)**, 122-125 (2007)

[8] Y. Wang et. al, *Journal of Nuclear Medicine* **47(11)**, 1891-1900 (2006)

[9] E. Aprile et. al, *Phys. Rev. B* **76(1)** 1891-1900 (2007)

[10] S. Jan, J. Collot, and E. Toumefier. IEEE NSS Conf. (2000)

[11] J. Su Kim et. al, *Journal of Nuclear Medicine* **48(9)**, 1527 (2007)

[12] F. Nishikido et. al, IEEE NSS/MIC Conf. (2002)

[13] E. Aprile and M. Suzuki, *IEEE Trans. Nucl. Sci* **36(1)** 311-315 (1989)

[14] T. Doke, et. al, *Nucl. Instr. Meth. A* **505(1-2)**, 199-202 (2003)

[15] R.P. Gardner and K. Verghese, *Nucl. Instr. Meth.* **93(1-2)**, 163-167 (1971)

[16] C.M. Bishop, *Review of Scientific Instruments* **65(6)**, 1803-1832 (1994)

[17] European Organization for Nuclear Research (CERN). "Geant3 User's Guide". http://wwwasdoc.web.cern.ch/wwwasdoc/geant_html3/geantall.html

[18] European Organization for Nuclear Research (CERN). "PAW Reference Manual". http://wwwasd.web.cern.ch/wwwasd/paw/reference_manual/tutref.ps.gz

[19] European Ogranization for Nuclear Research (CERN). "ROOT User's Guide". http://root.cern.ch/root/doc/RootDoc.html

[20] Matlab, Neural Network Toolbox User's Guide. http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf

[21] D.A. Bryman et. al, to be published.

[22] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[23] C.M. Bishop, and I.T. Nabney. *Pattern Recognition and Machine Learning: A Matlab Companion*. Springer, 2008. In preparation.

[24] C.M. Bishop. *International Journal of Neural Systems* **2(3)**, 229-236 (1991).

[25] P.J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University doctoral dissertation, 1974.

[26] Williams, Ronald J., David E. Rumelhart, and Geoffrey E. Hinton. *Learning internal representations by error propagation*. In Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1. Cambridge: MIT Press/Bradford Books, 1986.

[27] D.C. Mackay. *Neural Computation*. **4(1)**, 415-447 (1992).

[28] D. Foresee, F. Hagan. *International Conference on Neural Networks*. **3(1)**, 1930-1935, (1997).

[29] C.E. Rasmussen and C.K. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[30] S. Vijayakumar, A. D'Souza, and S. Schaal. *Neural Computation*. **17(12)**, 2602-2634, (2005).

[31] S. Vijayakumar, et. al. *Autonomous Robot*, **21(1)**, 2002.

[32] S. Vijayakumar and S. Schaal. Proc. of the Seventeenth International Conference on Machine Learning. pp. 1079-1086. (2000)

[33] J. Taylor and N. Cristianini. *Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[34] C. Chang and C. Lin. LIBSVM: a library for support vector machines (Updated March 2008). Software available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm

[35] R. J. Mcintyre. IEEE Transactions on Electron Devices, **13(1)**, 1966.

Appendix A

Files used in solution

It would fill too many pages to paste all source files referenced in this text into an appendix. I have therefore made these sources available at *http : //laplace.physics.ubc.ca/People/agwilson/xesources.tar.gz* and *http : //laplace.physics.ubc.ca/People/agwilson/xesources.zip*. If any source file is missing, or if you have any questions about any source, or about this thesis in general, please contact me at either *agwilson@physics.ubc.ca* or *aglwilson@gmail.com*. These sources are not presently refined, since many of them are still in development. I will post updated copies at the above addresses.

Here is a brief description of the some of the files:

For Running Simulations
`pointgen.cpp`: Distribute points in chamber, output coords in text file
`change.pl`: Generate ffcards files for these points
`xecomaker.pl`: Generate xepet.com files
`xecorun.pl`: Run xepet.com files
`photo.cpp`: Extract photo-detector data from root files, for use with KNN
`photonoise.cpp`: Same as `photo.cpp`, but adds APD noise
`neural.cpp`: Extract photo-detector data from root files, for use with neural nets
`nneural.cpp`: Same as `neural.cpp`, except APD noise is added

K Nearest Neighbour Results
`stats.txt`: 1.0 cm grid, no noise
`cleanstats.txt`: 0.5 cm grid, no noise
`fullnoise.txt`: 0.5 cm grid, all noise
`somenoise.txt`: 0.5 cm grid, all noise except photo-electron conversion
`scaledstats.txt`: 0.5 cm grid, data scaled as described, no noise

`knn.cpp`: K Nearest Neighbour Reconstruction Program

Appendix B

xepet.ffcards

```
xepet.ffcards

LIST
C
C **** Geant FFKEYs: see GEANT manual for more details ****
C
C ========== RUNG: IDRUN IDEVT [1,0]
C == IDRUN ==   User run number
C == IDEVT ==   User event number
C
RUNG 1 0
C
C
C ========== TRIG: NEVENT ==========
C
C == NEVENT ==  Number of events to be processed
C
TRIG 20
C
C ========== TIME: TIMINT TIMEND ITIME    ==========
C == TIMINT  == Time used for initialization
C               NOTE: FFCARD input for TIMINT is ignored/overwritten
C == TIMEND  == Time required for termination [10.]
C == ITIME   == Test every ITIME events
C               NOTE: User must optimize TIMEND/ITIME so that ITIME is
C                     as large as is save! - Program termination is
C                     initiated as soon as the time left on a particular
C                     queue is smaller than TIMEND.
TIME 0.0 100. -1
C
C
C
C ========== RNDM: NRNDM(1) NRNDM(2) ==========
C
C ==NRNDM==   Initial value of random number seeds NRNDM(1), NRNDM(2).
C          If NRNDM(2) is 0, the independent sequence NRNDM(1) is used.
```

```
C          If NRNDM(1) is 0, the default sequence is used. (9876, 54321)
C
RNDM 159432  142731
C
C ========= AUTO: IGAUTO =========
C
C          1 = Automatic computation of STMIN,STEMAX,DEEMAX,TMAXFD (default)
C          0 = Tracking media parameters taken from the argument list of GSTMED
C
C AUTO 0
C
C
C
C ========== HADR: IHADR ==========
C
C
C
C          0 = no hadron interactions effect
C          1 = hadron interactions with generation of secondaries (default)
C          2 = same without generation of secondaries
C
C          GHEISHA hadronic shower code if IHADR = 1
C          FLUKA   hadronic shower code if IHADR = 4
C          FLUKA/MICAP had. shower code if IHADR = 5
C
HADR 0
C
C ========== MULS: IMULS ==========
C
C          0 = no multiple scattering
C          1 = Moliere or single Coulomb scattering
C          2 = same as 1
C          3 = Gaussian scattering with Rossi formula
C
MULS 1
C
C
C
C ========== LOSS: ILOSS ==========
C
C          0 = no energy loss effect
C          1 = delta ray and reduced Landau fluctuations (default)
C          2 = full Landau fluctuations and no delta rays
C          3 = same as 1
```

```
C          4 = average Energy loss and no fluctuations
C
LOSS 1
C
C
C
C ========== STRA: ISTRA ==========
C
C          0 = Urban model for energy loss for thin layer (default)
C          1 = PAI model    "    "     "    "    "    "
C          2 = ASHO model for 1<delta<=30
C
C STRA 0
C
C ========== MUNU: IMUNU ==========
C
C          0 = no muon  nuclear interaction effect
C          1 = muon nucl. inter. with gen. of secondaries (default)
C          2 = same without generation of secondaries
C          Note: (IMUNU .NE. 0) only for GHEISHA!
C
C MUNU 1
C
C ========== ANNI: IANNxepet.ffcards_10001I ==========
C
C          0 = no positron annihilation effect
C          1 = positron annihilation with generation of secondaries
C          2 = same without generation of secondaries
C
ANNI 1
C
C ========== PFIS: IPFIS ==========
C
C          0 = no resonant photon absorption/photonfission
C          1 = photonfission with generation of secondaries
C          2 = resonant photon absorption/photonfission without secondaries
C
PFIS 2
C
C ========== SCNT: ISCNT ==========
C
C          0 = no scintillation process enabled
C          1 = scintillation process enabled
C          2 = (limited) scintillation process [1% of yield]
```

```
C
SCNT 1
C
C
C ========== YILD: PHOTON_YIELD RESOLUTION_SCALE ==========
C
C          photon_yield     = scintillation photons/MeV deposited energy
C          resolution_scale = > 1.0 => resolution worse than statistical
C
YILD 40000.0 1.0
C
C ========== THLD: TOT_THRSHLD PMT_THRSHLD ==========
C
C          tot_thrshld = threshold on the total number of photons
C                        detected in all PMTs
C          pmt_thrshld = threshold on the number of photons detected
C                        in each PMT (only the PMTs above pmt_thrshld
C                        contribute to the sum to which tot_thrshld is
C                        applied, and only those PMTs are used in the
C                        reconstruction)
C
THLD 0.0 0.0
C
C
C *** The ENERGY RANGE of the cross section and energy loss tables can
C     be fixed by the user with the new data card :
C                    'ERAN'   EKMIN  EKMAX    NKBIN
C     which defines nkbin bins from Ekmin to Ekmax in a logarithmic scale.
C     The default is, as before, 90 bins from 10 Kev to 10 Tev but in
C     logarithmic scale. NKBIN must be 50<NKBIN<200.
C
ERAN 0.00001 10.0 180
C
C
C ========== CUTS: CUTGAM CUTELE CUTNEU CUTHAD CUTMUO
C
C *** Low energy cutoffs - no tracking below these values [GeV] ***
C
C     CUTGAM   Kinetic energy cut for gammas [10 keV]
C     CUTELE   Kinetic energy cut for electrons [10 keV]
C     CUTHAD   Kinetic energy cut for hadrons [500 keV]
C     CUTNEU   Kinetic energy cut for neutral hadrons [500 keV]
C     CUTMUO   Kinetic energy cut for muons [10 keV]
C
```

```
CUTS 0.00001 0.00001 0.0005 0.0005 0.00001
C
C *** GEANT 3.21 global Cerenkov photon production flags
C
C ========== CKOV: ITCKOV ==========
C
C == CKOV = 0 No Cerenkov photon production [0]
C == CKOV = 1 Sequential parent particle tracking
C == CKOV = 2 Interrupted parent particle tracking
C
CKOV 2
C
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C         **** Geant User FFKEYs for debugging purposes ****
C
C ========== DEBU: IDEMIN IDEMAX ITEST ==========
C == IDEMIN  == First event to debug. If negative the debug flag IDEBUG
C               is set for the initialization phase
C == IDEMAX ==  Last event to debug.
C == ITEST  ==  Print control frequency (for all events!)-prints seeds
C
DEBU -1 100000 1000
C
C    *** The convention for GDEBUG is followed (see GEANT manual) ***
C
C == ISWIT(1) = 2: the content of the temporary stack for secondaries in
C                  the common /GCKING/ is printed;
C == ISWIT(2) = 1: the current point of the track is stored in the JDXYZ
C                  bank via the routine GSXYZ;
C             = 2: the current information on the track is printed via
C                  the routine GPCXYZ;
C             = 3: the current step is drawn via the routine GDCXYZ;
C             = 4: the current point of the track is stored in the JDXYZ
C                  bank via the routine GSXYZ. When the particle stops
C                  the track is drawn via the routine GDTRAK and the
C                  space occupied by the track in the structure JDXYZ
C                  released;
C             = 5: print GEANT vertex information via GPVERT at the end
C                  of the event (in GUOUT)
C == ISWIT(3) = 1: the current point of the track is stored in the JDXYZ
C                  bank via the routine GSXYZ;
C == ISWIT(4) = 0: individual rays as input
C             = 1: distribution of rays (origin and direction)
```

```
C == ISWIT(5) = 1: the current track is drawn via the routine GDTRAK when
C                  a photon hits a PMT (also requires ISWIT(2) = 1)
C == ISWIT(6) = 0: no output of hits, etc
C             = 1: ntuple output : nt# 997, 998, 999
C
C == ISWIT(7) = 0: no RZ file for input or output
C             = 1: write HITS RZ file
C             = 2: read  HITS RZ file
C
CSWIT 0 2 0 1 0 1 1
SWIT 0 0 0 1 0 1 1
C
C
C
C
C ========== HSTA: LHSTA
C == LHSTA ==   NHSTA names of required standard histograms
C
C HSTA 'TIME' 'SIZE' 'MULT' 'NTRA' 'STAK'
C
C
C ========== PRIN: LPRIN
C == LPRIN ==   NPRIN names of GEANT data structures to be printed
C
C PRIN 'PART' 'MATE' 'TMED' 'VOLU' 'SETS'
C
C ========== RGET: LRGET
C == LRGET ==   NRGET names of GEANT data structures to fetch from RZ
C
C
C RGET 'INIT'
C
C
C ========== RSAV: LRSAV
C == LRSAV ==   NRSAVE names of GEANT data structures to fetch from RZ
C
C
C RSAV 'INIT'
C
C
C
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C             ***** TEST Photon Detector Run directives *****
```

```
C
C ========== SORC: ISORC PSORC(14) ==========
C
C     ISORC     = Number of photons at initial vertex (if > 0)
C                 GEANT particle type (if < 0)
C     PSORC( 1) = Particle gun origin x [cm]
C     PSORC( 2) = Particle gun origin y [cm]
C     PSORC( 3) = Particle gun origin z [cm]
C     PSORC( 4) = Particle gun aim - rotation around x-axis (deg)
C     PSORC( 5) = Particle gun aim - rotation around y-axis (deg)
C     PSORC( 6) = Particle gun aim - rotation around z-axis (deg)
C     PSORC( 7) = sigma of x position (cm)
C     PSORC( 8) = sigma of y position (cm)
C     PSORC( 9) = horizontal emittance [mr]
C     PSORC(10) = vertical   emittance [mr]
C     PSORC(11) = Particle energy [MeV]
C     PSORC(12) = 1 +- deltaE/E
C     PSORC(13) = RAYTRACE theta [mr]
C     PSORC(14) = RAYTRACE phi   [mr]
C
SORC -3 1.00 2.00 3.00 0. 0. 0.  0. 0.   0.  0.  0.511 1.0  0. 0.
C
C ========== DMAT: N_DETMATE N_XY N_YZ N_XZ ==========
C
C     12 == LXe
C     18 == stainless steel
C     14 == white paint
C     19 == black paint
C
C DMAT 12 18 14 19
DMAT 12 19 19 19
C
C
C
C ========== DETE: TRD1_DIM(4) D_MTL D_IRT ==========
C
C             trd1_dim(1) == x1-dimension of scintillator [cm]
C             trd1_dim(2) == x2-dimension of scintillator [cm]
C             trd1_dim(3) == y -dimension of scintillator [cm]
C             trd1_dim(4) == z -dimension of scintillator [cm]
C
C             d_mtl  == metal/steel sheet thickness [cm]
C             d_irt  == inert LXe   sheet thickness [cm] ! 0.4 is
the min. allowed
```

```
C
DETE   11.122 5.2  15.  11.05  0.0635 0.4
C
C ========== MPMT: MTYPE_PMT ==========
C
C             mtype_pmt == 1 : circular
C             mtype_pmt == 2 : square
MPMT 1
C
C ========== PMTR: PMT_SIZE PMT_RING ==========
C
C             pmt_size   == half size or radius of PMT [cm]
C             pmt_ring   == the radial width of PMT housing ring [cm]
C
PMTR 0.8  0.3
C
C ========== NPMT: NX_FB NZ_FB NY_LR NS_LR NX_BT NY_BT NX_TP NY_TP ==========
C
C             nx_fb == number of PMTs along the front/back x-dimension
C             nz_fb ==   "    "   "     "    "  front/back z-dimension
C             ny_lr ==   "    "   "     "    "  left/right y-dimension
C             ns_lr ==   "    "   "     "    "  left/right s-direction
C             nx_bt ==   "    "   "     "    "  bottom     x-dimension
C             ny_bt ==   "    "   "     "    "  bottom     y-dimension
C             nx_tp ==   "    "   "     "    "  top        x-dimension
C             ny_tp ==   "    "   "     "    "  top        y-dimension
C
CNPMT 4 6 8 4 4 8 2 8
NPMT 4 5  0  0 0 0 0 0
C
C
C
C ========== BLKA: BULK_ABSORPTION(1-3) ==========
C
C             bulk_absorption(1) == in Scintillator [cm]
C             bulk_absorption(2) == in Glass and/or Lucite [cm]
C             bulk_absorption(3) == in WLS fiber [cm]
C
BLKA 75.0 420.0 500.0
C
C
C
C
C ========== REFL: STEEL_  WHITE_PAINT_  BLACK_PAINT_ABSORPTION ==========
```

```
C
C             steel_absorption       == (1-reflectivity)
C             white_paint_absorption == (1-reflectivity)
C             black_paint_absorption == (1-reflectivity)
C
REFL 0.2 0.0 1.0
C
C ========== PLSH: POLISHED(1) POLISHED(2) POLISHED(3) POLISHED(4) ==========
C
C     polished == the GLISUR polish model parameter
C                 1. -> perfect smoothness
C                 0. -> maximum roughness, with effective plane
C                       of reflection distributed as cos(alpha)
C                       where alpha is the angel between the
C                       unit normal to the effective plane of
C                       reflection and the normal to the nominal
C                       medium boundary.
C                 Between zero and one, the surface is modelled
C                 by a bell-shaped distribution in alpha, with
C                 limits defined by: sin(alpha) = +/- (1-polished)
C
C                 polished(1) - surface: Scintillator/LXe and Steel
C                 polished(2) - surface: Scintillator/LXe and Glass
C                 polished(3) - surface: Scintillator/LXe and MGO (White Paint)
C                 polished(4) - surface: Scintillator/LXe and Black Paint
C
PLSH 1.0 1.0 0.0 0.0
C
STOP
```