Summer Report

Continued progress on my undergraduate honours thesis study,

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

and other new work on the Liquid Xenon PET project.

aglwilson@gmail.com

Andrew G. Wilson

October 1, 2008

Contents

List of Tables

List of Figures

Chapter 1

Introduction

I assume the reader of this document is familiar with the material in my honours undergraduate thesis [1]. I present my continued progress in this direction – improved results, new figures, and completely new experiments; these experiments include adding reflectors to the chamber, and using photon sources. I also present tutorials for those wishing to further continue my work. And I describe work I have done which is not directly connected with my thesis – such as modelling solid angle or experimental work – but is relevant to the LXe PET project. Before and during your reading of this document, it will help to consult two presentations I have given on this material [2, 3]. In the second presentation, I thoroughly define some terminology first used (and also carefully defined) in my thesis, and frequently used throughout this document. Figure 1.1 shows an LXe chamber with the coordinate system I am using.

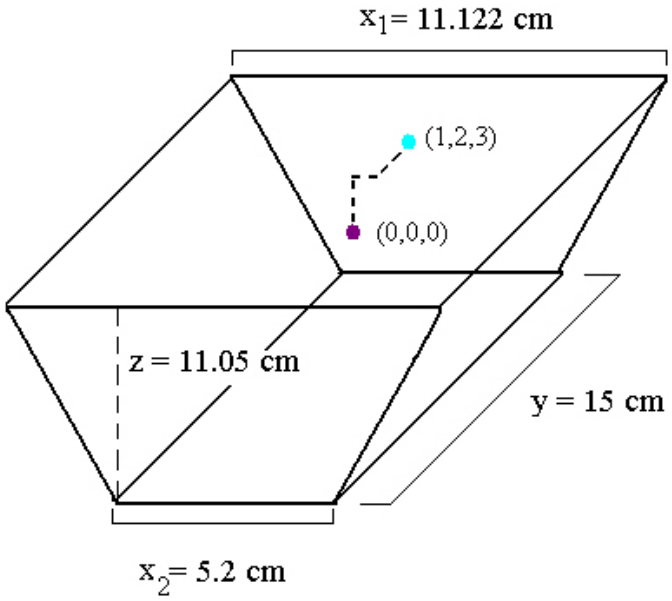

Figure 1.1: **XEPET Detection Chamber**. These chambers fit together to form a cylinder around the subject. 16 APDs are on each of the trapezoidal faces ($y = constant$), and an applied constant electric field runs from one $z = constant$ face to the other $z = constant$ face. Points are illustrated 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 report.

Chapter 2

Energy and Position Reconstruction: Stationary Sources

2.1 Introduction

All of the work in my thesis [1], involves reconstructing the positions and energies of **stationary** particles in an LXe chamber, from scintillation light. Further, all of these reconstructions are done assuming that the walls of the chamber are completely absorbing.

Since then, I have tried adding reflectors to the YZ surfaces, and the XY and YZ surfaces, to see the effect on position and energy reconstruction. With reflectors, one would expect the energy reconstruction to improve – there is more light data – and the position reconstruction to get worse, as there is more interference. Therefore the results may come as somewhat of a surprise: when typical APD noise is added to the photo-detector data, both position and energy reconstruction can improve by adding reflectors (up until a point)! This is because reflectors can increase the data at the photo-detectors such that noise has less of a relative effect when there are reflectors than when the walls are completely absorbing.

In this chapter, I first introduce many experiments (such as adding reflectors) intended to improve position and energy reconstruction. I briefly summarize the results of these experiments: for more details, please contact me. I then present the most accurate up-to-date results for reconstruction with stationary sources. You may wish to skip directly to this section. In a later chapter, I discuss reconstruction with photon sources. To avoid confusion, a **stationary particle or source** means *there is only **one** interaction point to reconstruct.* A **photon source** means *one **or more** interactions contribute to the light data used to reconstruct a position.* So often with photon sources, photo-detector data is generated from multiple interactions, but as before, the reconstruction algorithm will only return a single point – how this point relates to the interaction points is considered in this document.

Rather than writing large essays, where each paragraph discusses some new experiment, idea, or result, I have divided this chapter up into many extremely small subsections. This would not be ideal for a textbook or a thesis, but as a reference manual, it should save the reader time: just skim the index to immediately find what you may be looking for. Unless otherwise specified, all tests are done with 3811 points, spaced 0.41 cm apart, within the first three centimetres of the chamber.

2.2 Results and Experiments

2.2.1 Grids Generated

Table 2.1 shows grids I have used since my thesis, and Table 2.2 shows those grids I used for the work in my thesis. If you wish to use one of these grids, they are available in a package in the tutorial section, with

an easy means to convert the data into any format you please. This is highly encouraged over simulating similar data yourself, since it takes approximately 1.32 seconds to simulate each event – e.g. many of the grids here would take weeks or months of computing time to generate.

2.2.2 Changing network structure

The network I presented in my thesis has a [32-26-17-23-16-3] structure. When you saw this you likely thought, "why so many neurons and hidden layers?" Quite simply, this configuration was more accurate than all other configurations I had tested, and was competitive in training time. It was modelling this non-deterministic situation as well as any algorithm could, in many ways. And this is still the case. The next most accurate configuration I have tested has a [32-24-17-12-3] structure. It's also the fastest network I've trained (with an accuracy within 10% of the [32-26-17-23-16-3] network). Training using grids 1, 2, and 4, for instance, with a memory reduction factor of at least 100 (necessary in Matlab – see Matlab tutorial section for more details), and 200 epochs takes about 12 hours, as compared to the 16-20 hours it takes to train a [32-26-17-23-16-3] network. The radial error is usually only increased by about 1 mm, so to save time the smaller network ([32-24-17-12-3]) was used for all the tests I present in this document, unless otherwise specified.

A [32-64-3] network is one of the most obvious choices for this type of data – 64 neurons in the hidden layer with a tansig transfer function, and the 3 neurons in the final layer with a linear transfer function: a so-called 'rule of thumb' is to put twice as many neurons in one hidden layer as you have in the input layer, and leave it at that. This isn't near optimal in this situation. The network takes longer to train than even the [32-24-17-23-16-3] network (about 125% as long), and is about 0.2 cm worse in radial error. This ***is not*** necessarily the case when ***not*** using the Levenberg-Marquardt algorithm with the mean square error function: the network presented in the `C++/Root` tutorial section performs best with simpler structures like [32-64-3].

Finally, using Matlab implemented backpropagation with the Levenberg-Marquardt algorithm, most other structures tested had 0.4 cm worse radial reconstruction error than the [32-26-17-23-16-3] network; other structures tested had one or two hidden layers, each ranging from 5 to about 30 neurons, sometimes with one or more *logsig* transfer functions.

In conclusion, the [32-24-17-12-3] network was found best for testing reconstruction, and the [32-26-17-23-16-3] network was found best for performing reconstruction, when using Levenberg-Marquardt backpropagation with a mean square error function, and tansig transfer functions in all hidden layers. It's probably not worth your time experimenting with more structures when using the Levenberg Marquardt algorithm; at least, you should not spend more than a couple days testing new structures. I have a notebook that contains detailed information for each test of each structure I tried; re-writing the results in this document would likely cause confusion, and would fill up to a hundred pages. These results are available at your request.

2.2.3 Extracting data

In my thesis work, the way in which data was extracted from photo-detectors caused readings higher than '50' to be set to '0'. This became quite obvious when reflectors were added, increasing light output and therefore causing a very suspicious number of '0' readings. I changed the extraction method to avoid this issue, improving reconstruction accuracy by at least a few millimetres (most with reflecting surfaces), and energy resolution by at least a few percent (most with reflecting surfaces). In some cases this change did not make much of a difference – these cases are the ones explored in my thesis. In my new experiments, fixing this issue was helpful.

2.2.4 Changing error function

Since correcting the extraction issue, it proved beneficial with the levenberg marquardt algorithm to use a simple mean squared error function instead of a regularized mean square error function. Switching from RMSE to MSE improved radial reconstruction by about 2 mm. For full test results, please contact me.

2.2.5 Changing training data

In my thesis, I concluded that a mixture of multiple and single event training data is optimal – and I've confirmed this with subsequent tests. Initially, I thought it would be best to use all multiple event training data (with as many events as practical): this would provide a smooth deterministic mapping. For reasons discussed in my thesis, exclusively using this type of data isn't optimal for position reconstruction training; in short, the network learns from the statistical fluctuations present in single event data. 20% to 60% multiple event (100 to 200 event) training data was found to be optimal – changing the percentage of multiple event training data does not drastically affect reconstruction, so long as there are at least 30000 data points. To see how ineffective purely multiple event training data is, compare Tables 2.15 and 2.19.

Do not use more than 150000 data points for training in the first 3 cm of the chamber. It will take a long time (a few days or more), and will not likely improve accuracy much: in fact, it may get worse. In the first 3 cm, when going from training sets with 30000 data points to 150000 points, often no difference in accuracy was noted. Exclusively using multiple event data often made radial error of reconstruction worse by about 1 cm, and exclusively using single event data I did not usually get better reconstruction accuracy than 2 cm (with no noise added to training points). For detailed experiments of this kind, please contact me.

When trying to reconstruct data with lower scintillation yields (or lower energies), it helps to retrain the network with examples produced at these lower yields or lower energies. This is not as obvious as it might seem; why not just scale data produced at higher scintillation yields? Scaling actually works quite well, especially if you aren't quite sure what scintillation yield you have, or what energy particle you're reconstructing. I used a `mapminmax` function in `Matlab` to map all photo-detector readings to values in [-1,1]. In tables 2.3, 2.4, 2.5, you can see the results with YZ reflectors. Scaling 511 KeV data to reconstruct scaled 300 keV particles works a lot better than just leaving everything to the network, as seen in tables 2.12, and 2.13. But if you know the scintillation yield really accurately, it's better that you don't scale data. Compare the scaled results to the YZ reflectors section of "best results". For an example of how effective it can be tore-train a network with a more appropriate scintillation yield, take a network trained using 511 keV data, with a scintillation yield of 40000 photons per MeV, and reflectors on slanted surfaces. It reconstructs 300 keV, 25000 photon per MeV particles with the following radial error statistics:
`AVE: 1.69 cm, 90% 3.035 cm, STDEV: 1.0585 cm`
Compare this to a network trained on 300 keV data with a scintillation yield of 25000 photon per MeV, and reflectors on slanted surfaces. It reconstructs the same data as the other network with these results:
`AVE: 0.48 cm, 90%:  0.941 cm, STDEV: 0.4108 cm`.

In tables 2.6 through 2.13 I've given reconstruction statistics for 450 KeV, and 300 KeV data sets, using networks trained on 511 keV data. First I show statistics for absorbing walls, and then reflecting side walls (YZ). The scintillation yield for training and testing data is 40000 photons per MeV. In the reflecting side walls data set, I also show reconstruction of a 300 keV data set using a network trained on 300 keV data – this time with a 25000 scintillation yield for both sets. The accuracy is dramatically better than reconstructing a 300 keV data set with a 511 keV network, even though the scintillation yield is 25000 instead of 40000. These tables are a small sample (0.1%) of the experimental data I have taken, and graphs I have produced. What I present in this section may clarify what I mean by "experimental data".

Finally, contrast Table 2.12 with Table 2.14, where the network is trained and tested using 300 keV data (helping reconstruction of lower energy data), and with a scintillation yield of only 25000 (which

would hurt relative to having a 40000 scintillation yield for both training and testing sets). Even at the lower scintillation yield, the network reconstructs the 300 keV data far more accurately!

In summary, for position reconstruction in the first 3 cm of the chamber,

- Use 100 to 200 events for multiple event data
- Use 30000 to 150000 data points (for full chamber, 150000 to 400000 points).
- Between 20 to 60% of these data points should be multiple event
- With poor scintillation yield data, train on scaled data for versatility
- With accurate scintillation yield data, train on non-scaled data for precision
- Use a network trained on a scintillation yield and energies very close to input

2.2.6 Adding thresholds

When noise is added to input data, there is a chance that much of this data will take predictably improbable values. For instance, we may see photo-detectors recording numbers far higher than we would expect given the position and energy of the source, and the scintillation yield of the LXe. We may also have many photo-detectors recording values lower than we expect. Or we may even see impossible values, like a negative number of photons picked up at a detector.

By setting limits on data, we can remove some of these improvable values, effectively cleaning some of the noise from our data. Clearly, a threshold value of 0 will never hurt – that is, setting any negative photo-detector readings to 0. Generally, adding this threshold improves radial accuracy of reconstruction by at least a millimetre. Increasing this threshold value to about 2 has little effect on reconstruction, but higher thresholds noticeably deteriorate accuracy. An upper limit on photo-detector values is worth experimenting with further; I do not think it will have a significant effect, since the highest values are in the range of 4000 to 6000 photons, and though APD noise is in part proportional to the number of detected photons, it will be a relatively small fraction of these large numbers. In any case, it is worth testing further – perhaps by using an upper limit of 200 photons, a lot of very noisy data would be eliminated at the expense of only some mostly clean data. (Very unlikely based on my own preliminary tests, but possible).

2.2.7 Adding reflectors

Reflectors increase light output, which can improve energy resolution – with more light data, energy changes have a more pronounced effect on what is being recorded by photo-detectors. Reflectors can also obscure position information, however. Graphs of reconstruction accuracy as a function of position were produced for absorbing walls, reflecting walls only on slanted surfaces (YZ), and reflecting walls on XY and YZ surfaces. Please contact me if you wish to see these graphs. They were excluded as they follow a similar trend to error as a function of position in the full chamber with absorbing walls (figures 2.1 and 2.2).

2.2.8 Changes to Energy Resolution Measurement

My method for testing energy resolution is explained in my thesis and second presentation; I generate a multiple event grid of points all with 511 keV and the same scintillation yield. I then generate a single event test grid of points all with the same energies (e.g. 300 keV), and the same scintillation yield as my multiple event grid. To reconstruct the energy of each test point, I take the sum of the light data at that point, divided by the sum of the light data for the corresponding multiple event grid point, and multiply by 511, and then measure the absolute difference from the actual energy (e.g. 300 keV).

Using a multiple event grid either with a higher scintillation yield (and then scaling), or an energy closer to energy we expect to reconstruct is worth testing. Though I have not experimented with very high scintillation yields, using a 300 keV multiple event grid to reconstruct 300 keV particles improves energy resolution by about 0.5% (see table 2.24), over using a 511 keV grid. This is not a very dramatic improvement, but is good to keep in mind for the final refinements.

2.2.9 Changing the photodetectors

Photo-detectors should be placed where they will most help with position and energy reconstruction. A careful re-arrangement of the 32 photo-detectors currently planned could improve reconstruction as much as (or more than) spending tens or hundreds of thousands of dollars on purchasing and adding new photo-detectors.

One way to do this would be to distribute a representative sample of points in the chamber – since most interactions would take place within the first 3 cm, the density should be greater in this region. Some care should be put into choosing a sample of interaction points we would actually expect in a typical scanning session. This could be done through simulating photon sources, and varying the gaussian width of the photon beam; care should be taken to make sure the simulated situation is close to what we actually expect.

Then in some way, disable one of the photo-detectors, and see how it affects the overall accuracy of reconstructing all of these points – paying close attention to the 90% values. Then try this with each of the photo-detectors. You can 'disable' a photo-detector by not including it in training data. I wrote an algorithm that instead adds a constant factor of your choice, e.g. 20 photons, to the photo-detector in question, reconstructs all points, determines the 90% value, and repeats this process for every photo-detector, and then ranks the photo-detectors in order of importance. You can try different offsets, e.g. -40 photons, or +100 photons, and see how the ranking list changes. My program is available for download and listed in the `Matlab` tutorial section of this document.

It's also important to determine the most effective shape and size of the photo-detectors. Assuming circular detectors, when the radius gets very large, there is almost no useful information: a similar amount of light would be detected for many different positions in the chamber, and there is no room for other photo-detectors. Conversely, if the radius is very small, almost no light will be detected, and so the photo-detector becomes basically useless. One must look for the optimal intermediate radius. I recommend first optimizing photo-detector positions, and then optimizing size, and then checking if positions can be changed for the better, once again.

Grid	Spacing	Points	Events	Energy	Yield	Walls	Region	Used for
1	0.2 cm	33825	200	511 keV	40000	Absorbing	First 3 cm	Training
2	0.23 cm	22165	1	511 keV	40000	Absorbing	First 3 cm	Training
3	0.12 cm	157375	1	511 keV	40000	Absorbing	First 3 cm	Both
4	0.41 cm	3811	100	511 keV	40000	Absorbing	First 3 cm	V, EG
5	0.41 cm	3811	1	300 keV	40000	Absorbing	First 3 cm	ET, Te
6	0.41 cm	3811	1	450 keV	40000	Absorbing	First 3 cm	ET, Te
7	0.41 cm	3811	1	511 keV	40000	Absorbing	First 3 cm	Te
8	0.41 cm	3811	1	511 keV	25000	Absorbing	First 3 cm	Te
9	0.41 cm	3811	1	480 keV	25000	Absorbing	First 3 cm	Te
10	0.41 cm	3811	1	300 keV	25000	Absorbing	First 3 cm	Te
11	0.26 cm	15846	100	511 keV	40000	XY+YZ R	First 3 cm	Training
12	0.17 cm	55332	1	511 keV	40000	XY+YZ R	First 3 cm	Training
13	0.41 cm	3811	1	511 keV	40000	XY+YZ R	First 3 cm	Te
14	0.41 cm	3811	1	450 keV	40000	XY+YZ R	First 3 cm	Te
15	0.41 cm	3811	1	300 keV	40000	XY+YZ R	First 3 cm	ET, Te
16	0.41 cm	3811	100	511 keV	40000	XY+YZ R	First 3 cm	EG, V
17	0.23 cm	22165	100	511 keV	40000	YZ Refl	First 3 cm	Training
18	0.19 cm	37525	1	511 keV	40000	YZ Refl	First 3 cm	Training
19	0.41 cm	3811	1	511 keV	40000	YZ Refl	First 3 cm	Te
20	0.41 cm	3811	1	450 keV	40000	YZ Refl	First 3 cm	Te, ET
21	0.41 cm	3811	1	300 keV	40000	YZ Refl	First 3 cm	Te, ET
22	0.41 cm	3811	100	511 keV	40000	YZ Refl	First 3 cm	EG, V
23	0.23 cm	22165	100	511 keV	25000	YZ Refl	First 3 cm	Training
24	0.41 cm	37525	1	511 keV	25000	YZ Refl	First 3 cm	Training
25	0.41 cm	3811	1	511 keV	25000	YZ Refl	First 3 cm	Te
26	0.41 cm	3811	1	450 keV	25000	YZ Refl	First 3 cm	Te, ET
27	0.41 cm	3811	1	300 keV	25000	YZ Refl	First 3 cm	Te, ET
28	0.41 cm	3811	100	511 keV	25000	YZ Refl	First 3 cm	V, EG
29	0.23 cm	22165	100	300 keV	25000	YZ Refl	First 3 cm	Training
30	0.41 cm	37525	1	300 keV	25000	YZ Refl	First 3 cm	Training
31	0.41 cm	3811	100	300 keV	25000	YZ Refl	First 3 cm	V, EG
32	0.27 cm	68145	100	511 keV	40000	Absorbing	Full chamber	Training
33	0.24 cm	96453	1	511 keV	40000	Absorbing	Full chamber	Training
34	0.61 cm	6375	100	511 keV	40000	Absorbing	Full chamber	V, EG
35	0.61 cm	6375	1	300 keV	40000	Absorbing	Full chamber	ET
36	0.41 cm	19943	1	511 keV	40000	Absorbing	Full chamber	Te
37	0.41 cm	19943	1	300 keV	40000	Absorbing	Full chamber	Te
38	0.41 cm	3811	100	300 keV	40000	YZ Refl	First 3 cm	Te

Table 2.1: **New simulated photo-detector data generated since May 2008**. In all simulations, ffcards parameters not listed in this table were specified as in the appendix. "Both" means both used for training and position testing. ***Any*** of this data could be used for testing or training. Abbreviations: Te= Position testing, V=validation, EG=energy grid, ET=energy test. "XY+YZ R" means reflectors on XY and YZ surfaces.

Spacing	Points	Events	Energy	Yield	Walls	Region	Used for
1.0 cm	1365	1000	511 keV	40000	Absorbing	Whole chamber	Training
0.5 cm	11687	500	511 keV	40000	Absorbing	Whole chamber	Training
0.2 cm	33825	200	511 keV	40000	Absorbing	First 3 cm	Training
0.41 cm	3811	100	511 keV	40000	Absorbing	First 3 cm	EG
0.41 cm	19943	1	511 keV	40000	Absorbing	Whole chamber	Te
1.7 cm	297	1	511 keV	40000	Absorbing	Whole chamber	Te
0.23 cm	22165	1	511 keV	40000	Absorbing	Whole chamber	Both
0.41 cm	3811	1	450 keV	40000	Absorbing	First 3 cm	Te, ET
0.41 cm	3811	1	300 keV	40000	Absorbing	First 3 cm	Te, ET

Table 2.2: **Simulated photo-detector data generated for thesis**. **Simulated photo-detector data generated for thesis**. In all simulations, ffcards parameters not listed in this table were specified as in the appendix. "Both" means both used for training and position testing. ***Any*** of this data could be used for testing or training. Abbreviations: Te= Position testing, V=validation, EG=energy grid, ET=energy test.

	90% (cm)	average (cm)	stdev (cm)
radial	0.998	0.5295	0.3635
x	0.694	0.3031	0.3017
y	0.423	0.2018	0.1917
z	0.643	0.278	0.2734

Table 2.3: **Reconstruction statistics for 3811 particles (511 keV) without noise, YZ reflectors, first 3 cm of chamber, Test: 511 keV (scaled), Train: 511 keV (scaled), 40000/MeV.** 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	0.849	0.4475	0.3919
x	0.608	0.2546	0.3162
y	0.359	0.1724	0.2168
z	0.536	0.2357	0.2383

Table 2.4: **Reconstruction statistics for 3811 particles (511 keV) without noise, YZ reflectors, first 3 cm of chamber, Test: 511 keV, Train: 511 keV (scaled), 40000/MeV.** 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	1.274	0.6966	0.6183
x	0.872	0.3827	0.4609
y	0.629	0.3008	0.3825
z	0.803	0.3491	0.3874

Table 2.5: **Reconstruction statistics for 3811 particles (511 keV) withnoise, YZ reflectors, first 3 cm of chamber, Test: 511 keV (scaled), Train: 511 keV (scaled), 40000/MeV. Uses multiple event training data only (grid 30).** 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	0.914	0.526	0.4743
x	0.619	0.271	0.2782
y	0.292	0.1584	0.3795
z	0.683	0.3295	0.2601

Table 2.6: **Reconstruction statistics for 3811 particles (450 keV) with noise, absorbing walls. Network trained using 511 keV data.** 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	1.599	0.8418	0.8784
x	1.14	0.4952	0.5307
y	0.593	0.3136	0.7088
z	0.937	0.422	0.4178

Table 2.7: **Reconstruction statistics for 3811 particles (450 keV) with noise, absorbing walls. Network trained using 511 keV data.** 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	1.914	1.174	0.5934
x	1.08	0.5306	0.4045
y	0.614	0.3178	0.3817
z	1.653	0.8578	0.5502

Table 2.8: **Reconstruction statistics for 3811 particles (300 keV) without noise, absorbing walls. Network trained using 511 keV data.** 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	2.953	1.8214	1.6759
x	1.876	0.8836	0.8143
y	1.306	0.7825	1.6207
z	1.924	0.9806	0.6938

Table 2.9: **Reconstruction statistics for 3811 particles (300 keV) without noise, absorbing walls. Network trained using 511 keV data.** 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	1.05	0.5907	0.35
x	0.645	0.2838	0.2869
y	0.312	0.1535	0.1563
z	0.832	0.4183	0.2925

Table 2.10: **Reconstruction statistics for 3811 particles (450 keV) without noise, YZ reflectors. Network trained using 511 keV data.** 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	1.332	0.7187	0.4778
x	0.952	0.400	0.4120
y	0.443	0.2169	0.2483
z	0.914	0.4361	0.3408

Table 2.11: **Reconstruction statistics for 3811 particles (450 keV) without noise, YZ reflectors. Network trained using 511 keV data.** 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	1.841	1.1294	0.5279
x	1.093	0.483	0.4409
y	0.912	0.4191	0.3924
z	1.408	0.7569	0.4733

Table 2.12: **Reconstruction statistics for 3811 particles (300 keV) without noise, YZ reflectors. Network trained using 511 keV data.** 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	2.297	1.3486	0.7624
x	1.396	0.6271	0.579
y	1.301	0.581	0.6634
z	1.501	0.7853	0.5265

Table 2.13: **Reconstruction statistics for 3811 particles (300 keV) without noise, YZ reflectors. Network trained using 511 keV data.** 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	0.891	0.4708	0.318
x	0.675	0.2859	0.288
y	0.298	0.1385	0.1506
z	0.576	0.2599	0.2206

Table 2.14: **Reconstruction statistics for 3811 particles (300 keV) without noise, YZ reflectors. Network trained using 300 keV data, 25000/MeV scintillation yield.** 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	1.623	0.8041	0.9042
x	1.329	0.5851	0.7441
y	0.484	0.2327	0.3250
z	0.737	0.3438	0.5386

Table 2.15: **Reconstruction statistics for 3811 particles (511 keV) without noise, YZ reflectors, first 3 cm of chamber, Test: 511 keV, Train: 511 keV (mult only), 40000/MeV. Uses multiple event training data only (grid 30).** 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	0.618	0.3309	0.4544
x	0.464	0.2016	0.2322
y	0.243	0.1271	0.3920
z	0.347	0.1565	0.1646

Table 2.16: **Reconstruction statistics for 3811 particles (511 keV) without noise,absorbing walls, first 3 cm of chamber, Test: 511 keV (G7), Train: 511 keV (G1,2,4), 40000/MeV.** 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.

2.3 Best results

2.3.1 Introduction

In the process of trying to refine reconstruction, I tested many different ideas – new error functions, new network designs, new thresholds, new training data, new scintillation yields. Many of the ideas were introduced in the previous section, and detailed test results are available upon request. In this section, I present the best results of these tests: these are the figures to reference for the most up to date performance of stationary source position and energy reconstruction. All future sections also present only the most up to date, accurate results.

All figures here are for a scintillation yield of 40000 photons per MeV, except for some results shown in the YZ reflector section, where a 25000 photon per MeV scintillation yield is used. Initially tests were done with 40000 photons per MeV, but with an electric field, 25000 is more representative of what we expect. Future simulation runs should be done with a 25000/MeV yield. However, changing the scintillation yield from 40000 to 25000 does not have a dramatic effect, so long as a 40000 yield network is not reconstructing 25000 yield data. This is specifically illustrated by changing the the scintillation yield from 40000 to 25000, with YZ reflectors, in the **Changing Training Data** section, above.

All of the results shown in the following sections use a [32-24-17-12-3] network, with the Levenberg-Marquardt backpropagation algorithm, using a mean square error function. *G* is shorthand for *grid*. So *G1,2,5* would mean *grids 1,2,5 in table 2.1.*

2.3.2 Absorbing Walls

See tables 2.16 and 2.17 for the best results using absorbing walls in the first three centimetres of the chamber. The networks were trained using grids 1 and 2, and using grid 4 for validation data. When all walls are absorbing, position reconstruction is best (when there is no APD noise), and energy resolution is worst (see table 2.18).

2.3.3 YZ reflectors

With YZ reflectors, we achieve the best position reconstruction (with APD noise), and an energy resolution better than with absorbing walls, but worse than with reflecting walls on XY and YZ surfaces. I do not currently have results for reconstructing 511 keV particles with a 25000 photons/MeV scintillation yield. They would likely have 1-2 mm more radial accuracy than the results I show for reconstruction 300 keV particles. See tables 2.19 and 2.20 for position reconstruction with a 40000 photon/MeV yield. Contrast this with position reconstruction using a 25000 photon/MeV yield; the 25000/MeV yield causes results to be about 0.15 mm less accurate (radially). This isn't a big change in accuracy given that the number of photons detected is almost halved!. Energy reconstruction error goes up by about 1.5% when using a

	90% (cm)	average (cm)	stdev (cm)
radial	1.293	0.6796	0.8370
x	0.951	0.4104	0.4810
y	0.508	0.2694	0.6825
z	0.736	0.3185	0.3504

Table 2.17: **Reconstruction statistics for 3811 particles (511 keV) with noise, absorbing walls, first 3 cm of chamber, Test: 511 keV (G7 noise), Train: 511 keV (G1,2,4), 40000/MeV.** 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%	average	stdev
no noise	7.7%	3.6%	5.3%
noise	13.5%	6.5%	8.8%

Table 2.18: **Energy reconstruction of 3811, 300 keV particles (G5), using 100 event 511 keV grid (G4), absorbing walls, first 3 cm.** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

25000 photon/MeV yield. This is pretty good; contrast tables 2.23 and 2.25. But changing the energy grid doesn't seem to improve resolution much; there is a 0.5% improvement using a 300 keV grid (as opposed to a 511 keV grid) to reconstruct 300 keV data, as seen in table 2.24.

2.3.4 XY and YZ reflectors

Position reconstruction is poorest with XY and YZ reflectors. Adding reflectors has a noise effect. With YZ reflectors, this noise effect was overcome by the increased light output, when adding APD noise. With both XY and YZ reflectors, the noise effect becomes more significant. It isn't significant enough to make energy resolution worse than with YZ reflectors though. Energy resolution is substantially more dependent on light output than position reconstruction. Therefore it is no surprise than XY and YZ reflectors provide the best energy resolution. See tables 2.26 and 2.27 for position reconstruction, and 2.28 for energy reconstruction.

	90% (cm)	average (cm)	stdev (cm)
radial	0.714	0.3616	0.2974
x	0.609	0.2531	0.2647
y	0.251	0.120	0.1433
z	0.331	0.1532	0.1634

Table 2.19: **Reconstruction statistics for 3811 particles (511 keV) without noise, absorbing walls, first 3 cm of chamber, Test: 511 keV (G25), Train: 511 keV (G22,23,24), 40000/MeV.** 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	1.042	0.5357	0.4066
x	0.874	0.3706	0.3747
y	0.738	0.1791	0.2056
z	0.496	0.2295	0.2177

Table 2.20: **Reconstruction statistics for 3811 particles (511 keV) with noise, YZ reflectors, first 3 cm of chamber, Test: 511 keV (G25 noise), Train: 511 keV (G22, 23, 24), 40000/MeV.** 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	0.891	0.4708	0.318
x	0.675	0.2859	0.288
y	0.298	0.1385	0.1506
z	0.576	0.2599	0.2206

Table 2.21: **Reconstruction statistics for 3811 particles (300 keV) without noise, YZ reflectors, first 3 cm of chamber, Test: 300 keV (G27), Train: 300 keV (G29, 30, 31), 25000/MeV.** 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	1.281	0.6725	0.5136
x	0.978	0.4260	0.4358
y	0.499	0.2395	0.3143
z	0.712	0.319	0.2945

Table 2.22: **Reconstruction statistics for 3811 particles (300 keV) with noise, YZ reflectors, first 3 cm of chamber, Test: 300 keV (G27 noise), Train: 300 keV (G29, 30, 31), 25000/MeV.** 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%	average	stdev
no noise	5.7%	2.9%	4.65%
noise	9.6%	4.5%	6.21%

Table 2.23: **Energy reconstruction of 3811, 300 keV particles (G21), using 100 event 511 keV grid (G28), YZ reflectors, first 3 cm, 40000/MeV scintillation yield** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

	90%	average	stdev
no noise	5.3%	2.62%	4.09%

Table 2.24: **Energy reconstruction of 3811, 300 keV particles (G21), using 100 event 300 keV grid (G38), YZ reflectors, first 3 cm.** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

	90%	average	stdev
no noise	6.9%	3.29%	4.65%
noise	13.1%	6.06%	8.17%

Table 2.25: **Energy reconstruction of 3811, 300 keV particles (G27), using 100 event 300 keV grid (G31), YZ reflectors, first 3 cm.** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

	90% (cm)	average (cm)	stdev (cm)
radial	0.914	0.4459	0.3626
x	0.67	0.2744	0.3129
y	0.255	0.1207	0.1181
z	0.591	0.2477	0.2595

Table 2.26: **Reconstruction statistics for 3811 particles (511 keV) without noise, XY and YZ reflectors, first 3 cm of chamber, Test: 511 keV (G13), Train: 511 keV (G11, 12, 16), 40000/MeV.** 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	1.221	0.6012	0.5089
x	0.898	0.3753	0.4303
y	0.359	0.1723	0.2159
z	0.756	0.3239	0.3365

Table 2.27: **Reconstruction statistics for 3811 particles (511 keV) with noise, XY and YZ reflectors, first 3 cm of chamber, Test: 511 keV (G13 noise), Train: 511 keV (G11, 12, 16), 40000/MeV.** 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%	average	stdev
no noise	4.2%	2.2%	4.4%
noise	6.3%	3.2%	4.4%

Table 2.28: **Energy reconstruction of 3811, 300 keV particles (G15), using 100 event 511 keV grid (G16), XY and YZ reflectors, first 3 cm.** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

2.4 Full Chamber, Absorbing Walls

2.4.1 Introduction

Up until now, I have only presented results for reconstruction within the first three centimeres of the chamber. This is where the majority of interactions will take place. Effort should go into making sure that reconstruction works especially well within this region; however, in the end, we should be able to reconstruct interactions at any point in the chamber.

In this section, I present reconstruction throughout the entire chamber, with absorbing walls. I do this to show more clearly how position and energy reconstruction accuracy vary as a function of position. In this case a scintillation yield of 40000 photons per MeV is used. The results of using 25000 photons per MeV would not differ by more than 1 to 2 mm radially, and the trends would be exactly the same.

2.4.2 Position Reconstruction

I used grids 32, 33, 34, 35, 36 in table 1.1 for training and testing the network. In tables 2.29 and 2.30 are statistics for reconstructing grid 35 with and without noise, respectively. Figure 2.1 shows the average reconstruction error as a function of the three spatial positions. For example, for the upper left figure, each black dot represents an x value where reconstruction was tested. Each of these x values was reconstructed about 30 times, each time with different y or z values. For example (-5, -7.5, -5), (-5, -7.1, -5), (-5, -7.1, -4.4), etc. The errors of reconstructing each of these points with a fixed x value are then averaged to determine the height of the black dot; in other words, for each black dot, we reconstruct points along a plane with constant x value, and varying y and z values, and then average the errors of these reconstructions to determine the height of the black dot. The most obvious problem with this technique is that the coordinates are closely interconnected as far error output goes. It *does* matter what the y and z positions are when trying to estimate the error of reconstructing a point; (-5, 3, 4) is much different than (-5, -2, 4). It would be better to draw level curves in three dimensions, varying the z coordinate – make a new graph for every new level curve $z = constant$. Or a select few points could be chosen, and plotted in 3D. The elliptical error surface (e.g. 90%) associated with reconstructing that point could then be drawn around it. This way, no accuracy would be lost to averaging non-similar data: we would get a very good feel for how error varies as a function of position.

Nonetheless, there is a clear trend in these figures: x and z errors decrease towards the centre of the chamber, and y errors increase. Also, the x and z errors are somewhat constant within 4 centimetres of the centres. This is intuitive. The total solid angle the photodetectors take up will be smallest when the y coordinate is centred. Conversely, the total solid angle of the photodetectors is largest when the x and z coordinates is centred. Therefore we expect to see the most light information when the x and z coordinates are centred, but when the y coordinate is close to the detectors. That's just what we see in the graphs. This is reassuring.

The tabular values for position (and energy in the next section) are not to be taken too seriously. They give a rough estimate (upper limit) of how well we can expect reconstruction to work throughout the full chamber, but with more simulation time, position reconstruction would be 1-4 mm better radially, and energy reconstruction would be 3% better.

2.4.3 Energy Reconstruction

Energy reconstruction error is shown in table 2.31, and energy reconstruction as a function of position is illustrated in figure 2.2. This error is calculated in the same way as was done in figure 2.1. And this figure follows similar trends: error spikes at the edges, a constant minimum from -4 cm to 4 cm in the x and z coordinates, and an increase towards the centre in the y coordinate. This likely happens for the same reason it does with position reconstruction: light information is greatest when the y coordinate is close to the detectors, and the x and z coordinates are centred. This trend is expected.

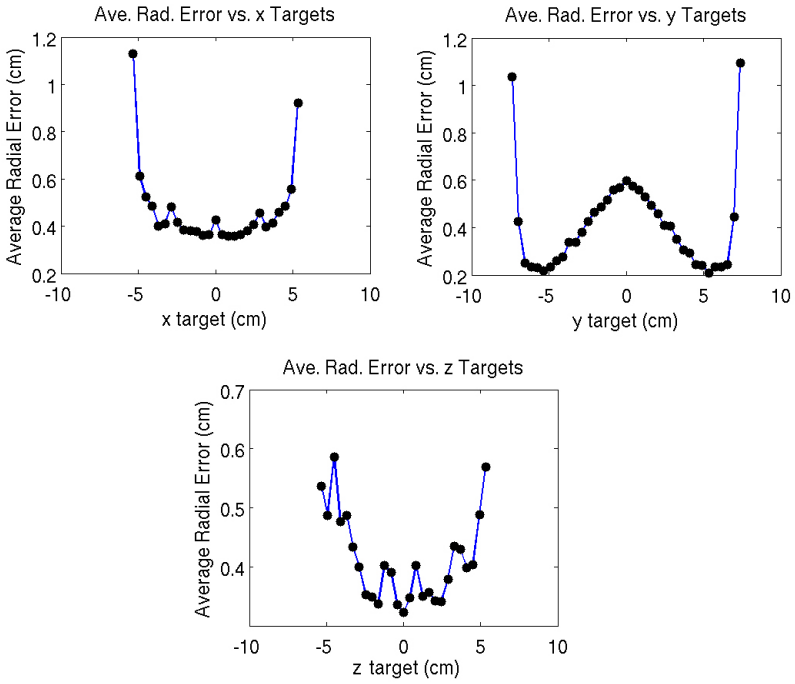

Figure 2.1: **Average position reconstruction error as a function of position.** Error is highest at the very far edges of the chamber. Error increases as the y coordinate approaches the centre, and decreases as x and z coordinates approach the centre. When the y coordinate is close (but not too close) to the sides, the solid angle of the detectors is maximized (holding x and z constant). Conversely, when y is constant, solid angle of the detectors is maximized when x and z are centred.

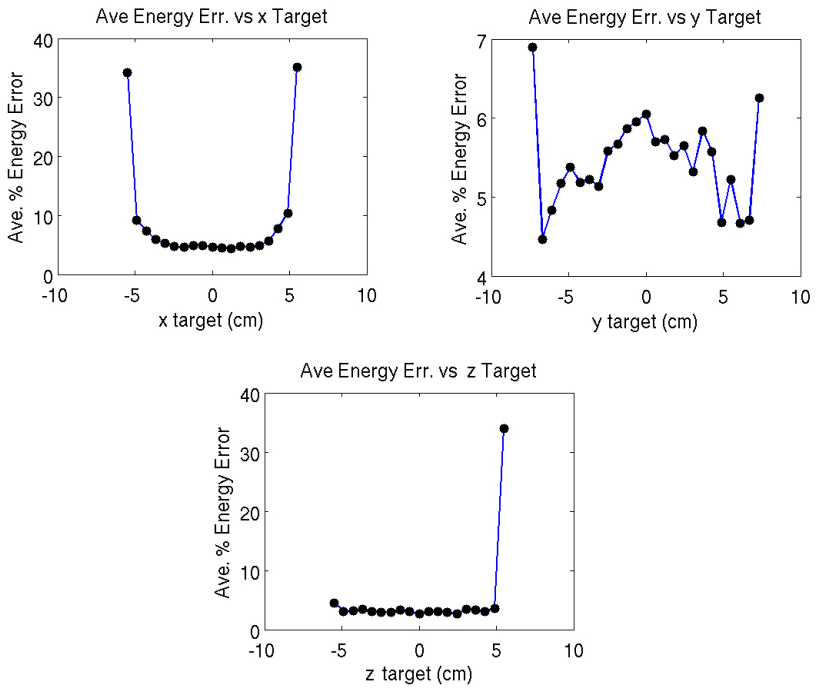

Figure 2.2: **Average energy reconstruction error as a function of position.**. Error is highest at the very far edges of the chamber. Error increases as the y coordinate approaches the centre, and decreases as x and z coordinates approach the centre. When the y coordinate is close (but not too close) to the sides, the solid angle of the detectors is maximized (holding x and z constant). Conversely, when y is constant, solid angle of the detectors is maximized when x and z are centred.

	90% (cm)	average (cm)	stdev (cm)
radial	0.738	0.4101	0.4217
x	0.514	0.2306	0.2462
y	0.324	0.170	0.3288
z	0.462	0.2089	0.2271

Table 2.29: **Reconstruction statistics for 19943 particles (511 keV) without noise, absorbing walls, full chamber, Test: 511 keV (G36), Train: 511 keV (G32,33,34), 40000/MeV.** 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	1.257	0.6463	0.5885
x	0.853	0.3690	0.3879
y	0.517	0.2555	0.4113
z	0.762	0.3334	0.3631

Table 2.30: **Reconstruction statistics for 19943 particles (511 keV) without noise, absorbing walls, full chamber, Test: 511 keV (G36 noise), Train: 511 keV (G32,33,34), 40000/MeV.** 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%	average	stdev
no noise	9.9%	5.45%	10.71%
noise	16.7%	7.94%	12.8%

Table 2.31: **Energy reconstruction of 6375, 300 keV particles (G35), using 100 event 511 keV grid (G34), absorbing walls, full chamber.** 90% of reconstruction under error in first column. Average reconstruction error in second column. Standard deviation of reconstruction error from average error.

Chapter 3

Photon Sources

Up until now, I've only considered 'stationary' sources. In practice, photons will enter the chamber and interact with electrons. I have simulated this process, producing a sample of 47788 representative events, each with 1 to 5 interactions. In the tutorial section, I've explained how to repeat my simulations.

I allowed interactions to occur at any place within the chamber: I wanted this to be similar to what we expect in practice. However, I did use all absorbing walls, and a scintillation yield of 40000/MeV – since the network I use to handle full chamber data was trained under these conditions: it's the same network I presented in the previous section, trained on grids 32 through 36. This won't be much different than if the the scintillation yield were 25000 photons per MeV, as evidenced in the previous sections. And the absorbing walls often cause error to be higher than YZ reflecting walls. In the end, the figures are comparable to using a 25000 yield, to within a couple millimetres.

But how should multiple interactions be reconstructed, and what's the error? They happen so quickly that one set of photo-detector data can correspond to several interactions: interaction 2 adds to the light from interaction 1, interaction 3 adds to the combined light from interactions 1 and 2, and so on. Putting all of this light data into my network only gives back one point, when there were in fact many points we wanted to reconstruct. Getting around this isn't easy: the error measure we should use isn't obvious. We could use the distance of the one reconstructed point from the average locations of the interaction points. Or we could take the so-called "centre of mass" (read: centre of energy deposition) of the interaction points, by doing a weighted average of the locations with how much energy was deposited at each interaction, and measure the distance from the reconstructed point. Or we could measure the distance from the reconstructed point to the furthest interaction point. I took all of these measures – and wrote software that will allow others to continue to use these measures – in determining error from reconstruction with photon sources. Tables 3.1 through to 3.5 show the results.

Using the max distance measure, the average error measures are reasonable: without noise, the radial error is 1.805 cm, and with noise the error is 2.016 cm radially. The standard deviations on each figure are 1.37515 cm and 1.36903 cm, respectively. (It is surprising that the standard deviation is slightly lower with noise: this suggests noise has only a small effect on this type of reconstruction). On the other hand, the error increases rapidly as a function of the confidence level; for example, 80% of radial errors are under 3 cm (with or without noise), but 90% of radial errors are under no less than 4.5 cm (with or without noise). In this case, it is likely best to go with 80% values, and form an error sphere (or ellipse) out of these values.

	90% (cm)	average (cm)	stdev (cm)
radial	2.67	1.0814	1.2541
x	1.32	0.5189	0.6958
y	1.531	0.5389	0.8616
z	1.51	0.5619	0.8002

Table 3.1: **Average position photon reconstruction error. 47788 events without noise, absorbing walls, full chamber, Train: 511 keV (G32,33,34), 40000/MeV.** 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. The error is the distance of reconstructed point from the average positions of interactions.

	90% (cm)	average (cm)	stdev (cm)
radial	2.963	1.316	1.3087
x	1.567	0.6702	0.7774
y	1.535	0.604	0.8935
z	1.687	0.6966	0.8617

Table 3.2: **Average position photon reconstruction error. 47788 events with noise, absorbing walls, full chamber, Train: 511 keV (G32,33,34), 40000/MeV.** 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. The error is the distance of reconstructed point from the average positions of interactions.

	90% (cm)	average (cm)	stdev (cm)
radial	2.46	0.9673	1.2473
x	1.165	0.486	0.6945
y	1.391	0.4945	0.8604
z	1.116	0.4701	0.7531

Table 3.3: **Centre of mass photon reconstruction error. 47788 events without noise, absorbing walls, full chamber, Train: 511 keV (G32,33,34), 40000/MeV.** 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. The error is the distance of reconstructed point from the centre of energy deposition of the interactions.

	90% (cm)	average (cm)	stdev (cm)
radial	2.795	1.2174	1.3088
x	1.503	0.6439	0.7774
y	1.468	0.5651	0.8918
z	1.444	0.6145	0.7531

Table 3.4: **Centre of mass photon reconstruction error. 47788 events without noise, absorbing walls, full chamber, Train: 511 keV (G32,33,34), 40000/MeV.** 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. The error is the distance of reconstructed point from the centre of energy deposition of the interactions.

	ave (cm)	80% (cm)	90%	stdev (cm)
no noise	1.805	3	4.5	1.376
noise	2.016	3	4.5	1.369

Table 3.5: **Largest Radius Photon Reconstruction Error. 47788 events without noise, absorbing walls, full chamber, Train: 511 keV (G32,33,34), 40000/MeV.** The first column is the average error in reconstruction. 80% of outputs are under the specified error in the second column. 90% of outputs are under the specified error in the second column. The fourth column is the standard deviation in reconstruction error. The error is the distance of reconstructed point to the farthest interaction.

Chapter 4

Reliability of Results

As much as I have made an effort to provide good results, I have also focused on being consistent in my experiments: in order to see how certain changes affect accuracy, I keep other variables as constant as is practical; a change from absorbing walls to reflecting walls isn't significant if the locations of the testing points are also different, or the composition of the training data is drastically changed, or if the scintillation yields aren't the same, etc. I've used the same 3811 testing point coordinates in my comparisons, similar training data, the same scintillation yields, and the same energies of the particles. This comes at a cost; it takes significant time to simulate testing and training grids, and many grids are required for the many experiments I have conducted. It's often tempeting to sacrifice consistency to get some really good results for a particular experiment, or some really quick results. In this situation – where I wish to preserve consistency – economical decisions are a necessary expense to accuracy.

For example, using a [32-26-17-23-16-3] structure, as opposed to the [32-24-17-12-3] structure I used in all of my tests, radial accuracy would improve by 1 to 2 mm. It's also possible to get another millimetre of radial accuracy out of tampering with training data a little more. For example, I could use high scintillation yields and then scale to the desired scintillation yield, to simulate high event data. Or I could use more than 100 to 200 events for multiple event data.

However, testing data seems representative. For example, when I reconstructed 157375 noiseless points (as opposed to 3811) in the first three centimetres, using absorbing walls, the 90% radial error was 0.599 cm, as opposed to 0.618 cm. This difference is negligable. And the differences in training data between experiments intended to test factors other than training data – reflectors, for example – are also negligable. I've tested this notion extensively, using many different training grids to check for differences in accuracy.

On the reliability of the tests I've presented in this document, I have two important conclusions:

- **More than a 2 mm radial difference between two different tests is significant**
- **All of the results presented could be improved radially by up to 3 mm**

Chapter 5

Additional work

5.1 Solid Angle Mapping for Energy Reconstruction

Astrid is measuring energy resolution from light in a way that requires a solid angle correction. My multiple event grids are an approximate solid angle mapping, when scaled by the number of photons generated. The more events, and the finer the grid, the more accurate the mapping is.

In this situation, I want to eliminate the effect of solid angle on photo-detector data. For any given position, I can provide a scaling factor that does this. The multiple event grid is a matrix, with 32 rows, and as many columns as there are points in the grid. The entry at the *ith* row and *jth* column is the reading at photo-detector i for point j. I also have a matrix of coordinates (3 rows for 3 spatial dimensions), where each column corresponds to a column in the multiple event photo-detector matrix. Point j, in this case, would have coordinates described the *jth* column of the coordinate matrix. I transform the photo-detector matrix into a vector by summing each row. Scaling this new vector by the largest entry, I have my solid angle scaling factor, in (0,1]. The correction factor for point j in the coordinate vector is column j in my new vector – divide the photo-detector readings by this number.

So we want to be able to enter any coordinate in the chamber, and then be given a correction factor in (0,1]. And we need this to happen quickly: we need to do this mapping millions of times, and have it only take minutes. My first approach was to measure the euclidean distance between the input coordinates, and every entry in my coordinate vector, find the nearest coordinate, and then output the corresponding correction factor. This greatly improved energy resolution: it went from 21% to 13%. But it took too long – 3 hours to reconstruct 150,000 points.

I trained a C++ `Root` neural network, to generate a simple function that takes coordinates to a scaling factor. In addition to the undoubtedly large speed improvements, perhaps the interpolation would also improve accuracy. This wasn't the case. Energy resolution went from 21% to 19%. But reconstruction of 150,000 points took seconds.

So I needed at least the accuracy of the first approach, and the speed of the second approach. I devised a way to map coordinates to an index in a scaling vector, so that input to output would take $\sim$O(1) operations! The x, y, and z values in the grid were all bounded – the LXe chamber has finite dimensions – and the grid spacing was 0.27 cm. Multiplying the grid values each by 100, I had $-540 \leq x \leq 540$, $-729 \leq y \leq 729$, $-540 \leq z \leq 540$. To avoid dealing with negative integers later, I add 540, 729, and 540 to the bounds. So $0 \leq x \leq 1080$, $0 \leq y \leq 1458$, $0 \leq z \leq 1080$. There are 1080/27+1=41 possible x values, 1458/27+1=54+1 possible y values, and 1080/27+1=41 possible z values. I can uniquely map the entire coordinate grid onto scaling values, without wasting space, with the conversion index=x+41*y+41*55*z.

Now, I need to find a way to convert input into the nearest grid value. With that nearest grid value, I can access the scaling factor with my mapping. This conversion is simple modular arithmetic. I can find the nearest grid point for each coordinate by examining the remainder when I divide by 27. For instance,

this is how I would adjust the input y coordinate:

```
adj = y % 27;
  if (adj<14)
    y=y-adj;

  else
    y=y+(27-adj);
```

Now I have an $\sim$O(1) mapping that is just as accurate as my first attempt! But honestly, it's a little more complicated than I've let on; there are fewer than 41 possible x values, since this grid is within the confines of a trapezoidal 3D chamber (a trapezoid pushed through another dimension). So there will be a little bit of wasted space in this mapping, and I have to be careful when adjusting the input x coordinate value with modular arithmetic: I can't push it outside of the chamber.

Right now I'm using grid 32. 100 events, absorbing walls, etc. I can improve the accuracy of this mapping by increasing the number of events. I found the accuracy most suffered from low event data when $-5 < y < 5$. In this region, I am presently generating 10000 event data. The program `ksearch.cpp` is available for download in the tutorials section.

5.2 Lab work

For two weeks I became familiar with the experiments taking place in the cryogenics lab, so that I could continue them if needed. I took measurements used for calibrating light detectors; by moving a radioactive source, I determined the sensitivity of each detector.

Chapter 6

Suggested Tasks

I've accumulated some ideas for future work. Here is a point form list:

- (Look through all tutorials, complete tasks there)
- Modify ffcards to make multiple position simulations involve only one file
- Experiment with high scintillation yields to substitute for high event data
- Graph level surfaces of reconstruction error vs. position with different z=constant values
- Graph reconstruction points in 3D with error ellipses surrounding them
- Estimate efficacy of reconstruction focusing on y coordinate, as opposed to radial error
- Make really high event (4000+), closely spaced ($< 1mm$) grid throughout full chamber, with absorbing, reflecting XY+YZ, and reflecting YZ
- Determine optimal positions of the 32 photo-detectors
- Then determine optimal sizes of 32 photo-detectors
- Then re-determine optimal positions
- Important: experiment with 0 width gaussian photon beams (see tutorial)

Associating photo-absorption and Compton scattering events with the correct light data can be helped by grouping possibilities based on energy measurements. For example, suppose two photons enter the chamber and both interact twice. We can determine the locations of these points from the timing of the light information and the charge information. To place the interactions into the correct groups, we match up energies. The energies of each group should add to 511 keV. This method can extend to a larger number of interactions, and it is worth incorporating into my photon-source tests. Fabrice Retiere can provide details.

Chapter 7

Conclusions

- Going from a 40000/MeV to a 25000/MeV scintillation yield affects radial accuracy by 1 to 2 mm
- With a 25000/MeV scintillation yield, 90% of radial error should be less than 5 mm without noise, and less than 9 mm with noise, for stationary sources.
- We can expect 90% of energy reconstruction from light to have an error less than 4% with noise, and less than 6% without noise.
- With noise, YZ reflectors minimize position reconstruction error, and XY+YZ reflectors minimize energy reconstruction error. Some intermediate compromise should be chosen, considering the relative importance of position reconstruction versus energy reconstruction.
- Position and energy reconstruction is most accurate when $y = \pm 6$ cm, and x and z are centred.
- A network structure, algorithm, and composition of training data have been optimized for this problem. (LM, 22%-60% 100 event data, rest single event).
- With photon sources, an upper limit on error increases rapidly with confidence level. For example, an 80% confidence level is more appropriate than a 90% one in this case: a large majority of points reconstruct accurately, but some points with very high errors unjustly skew the overall impression of accuracy when using high confidence levels.
- With photon sources, with noise, we can expect to reconstruct interaction at least to within a sphere of 3 cm, 80% of the time.
- Reconstruction will be more accurate than suggested by radial figures. The y coordinate reconstructs with exceptional accuracy – to within 3.5 mm, 90% of the time, with or without noise.

Chapter 8

Tutorial

8.1 Introduction

Most of my work can be divided into two obvious categories – simulations and regressions. Energy reconstruction is completely simulation dependent, the way I have approached it. And in order to reconstruct a particle from the scintillation light it emits, I need to simulate photo-detector data from stationary sources (or produce such data experimentally, which is very difficult). These simulations act as a mapping from points in the chamber, which are specified alongside the chamber properties, to photo-detector data. These simulations are also examples of the reverse mapping. In order to devise a scheme for position reconstruction, a regression of some sort should be run over this example data.

This regression should be tested by simulating photo-detector data from new stationary sources, and measuring how well reconstruction is performed. This regression should also be tested by simulating what we expect in practice: we must simulate an event where a photon source interacts at *several* locations, and each of these interactions contribute to the photo-detector data generated.

At this point, it is worth re-iterating something I wrote previously: "To avoid confusion, a **stationary particle or source** means *there is only **one** interaction point to reconstruct.* A **photon source** means *one **or more** interactions contribute to the light data used to reconstruct a position.* So, often with photon sources, photo-detector data is generated from multiple sources, but as always, the reconstruction algorithm will only return a single point – how this point relates to the interaction points is considered in this document."

In this chapter you should learn how to simulate data and run regressions in the way I have. You should be comfortable with `C++`. If you are not, find an introductory book, and spend a couple days teaching yourself the language by coding example programs. You should also have a very basic familiarity with `root`. Look through online `root` tutorials. Don't spend more than a day doing this though. For the `Matlab` simulation section, you should become moderately comfortable with `Matlab`: you should know how to make 2 dimensional plots, how define vectors and matrices, and how to manipulate data within vectors and matrices. You should also know how to write a `.m` script and run it, and how to write a Matlab function. And you should know how to use `for` loops and `if` statements. There are many `Matlab` tutorials that would quickly teach you how to use `Matlab` in these simple ways. You can also figure some of these things out by looking at the `.m` files I include in the simulation package I describe in the simulations section. Take a look at this reference [6].

8.2 Simulations Tutorial

8.2.1 Stationary Sources

Read chapter 2 (starting on p.18) of my thesis [1]. Download `simufiles.tar` at
http : *//laplace.physics.ubc.ca/People/agwilson/simfiles.tar*

Here is a description of each file:
`pointgen.cpp`: user supplies a grid spacing, program outputs coordinates in user specified file
`xepet.ffcards`: simulation file
`xepet.com`: runs simulation file (*modified from xepet.com others at TRIUMF may know of*)
`xepet_batch`: needed to by `xepet.com` to run simulation
`xepemaker.pl`: generates xepet.ffcards files for each coordinate generated by pointgen.cpp
`xecomaker.pl`: generates xepet.com files for each xepet.ffcards file
`xecorun.pl`: runs each of the xepet.com files
`toroot.pl`: converts output `hbook` files into `root` files
`neural.C`: a C++ macro that does root histogram projections, and formats photo-detector data for `Matlab`
`photo.C`: like `neural.C`, except outputs photo-detector data in a text file
`nneural.C`: like `neural.C`, but adds typical APD noise to photo-detector data
`rng.h`: used by `nneural.C` for generating random variables
`rng.C`: used by `nneural.C` for generating random variables
`targets.pl`: creates a `Matlab` vector of target coordinates from points generated by `pointgen.cpp`
`pointgen.cpp`: supply a grid spacing, and output text file to generate coordinates for grid.
`photextr.C`: (generalized file) used for extracting data from a root file simulated with a photon source
`photonave.C`: measures average location of interactions, outputs as targets
`photonce.C`: measures centre of energy location of interactions, outputs as targets
`photondata.C`: takes photo-detector data from simulations using a photon source (with optional noise)
`photondist.C`: measures distances of reconstructed points to further interactions

To avoid issues, set all simulation files to be executable (e.g. place all files in one directory, and then type `chmod u+rwx *`).

Task 1: familiarize yourself with ffcards

Put `xepet.ffcards` and `xepet.com` in the same directory (e.g. create a `simulations` directory in your home directory, and put these files there). Open `xepet.com`, and find the line

```
setenv XEPETGEANT /enteryourdirectoryhere
```

Replace `/enteryourdirectoryhere` with the full path of the directory you would like simulation output to appear in. Then edit the line

```
../bin/xepet_batch19 >& $XEPETGEANT/run$runn.log
```

so that `../bin/xepet_batch19` is replaced with the location of `xepet_batch` on your system. In `xepet.ffcards`, set the number of events to `25`, the run number to `14`. Find the `SORC` line. Make sure the first argument is `-3`. This means we are using electron sources. Place the electron at (-1, 2, 3). Give it 400 KeV of energy. Set the scintillation yield to 25000 photons per MeV. Make sure all walls are absorbing by checking the `DMAT` parameter. Once you have finished editing `xepet.ffcards`, save your changes, and run `xepet.com`: type `./xepet.com`. If there is an error, contact Peter Gumplinger after having tried your

best to resolve the issue yourself. Be clear about what you are trying to do, and what the error is.

As soon as you start running `xepet.com`, you will see `run14.hbook` appear in the directory you specified. Do not access this file until `xepet.com` finishes running: for this 25 event simulation, this should take approximately 30 seconds. Once `xepet.com` has finished, convert `run14.hbook` to `run14.root` by typing:

```
h2root run14.hbook
```

Load this file into `root` with `root run14.root`. Run `h999.Print()`, to see all available information. Try drawing various histograms. Follow the example in the simulations section of my thesis (chapter 2, starting p. 18). Repeat this procedure with a different number of events, or a different random number seed. Try adding `YZ` reflectors. What effect does this have on the photo-detector data? (Look at pmt_energy[i], where i is the (i+1)st photodetector).

Try extracting the photo-detector data in a format of your choice, by modifying `photo.C`. Add typical APD noise to this data, by looking at `nneural.C`. APD noise is added as follows. If PDi is the number recorded at photo-detector i, then, this is converted to PDi", the number recorded at photo-detector i with APD noise, in this way:

Step 1. Let $PDi' = Gaussian(\mu = PDi, \sigma = \sqrt{PDi})$ (multiplication noise)
Step 2. $PDi'' = PDi' + Gaussian(\mu = 0, \sigma = 6)$. (additional noise)

Task 2: simulating grids of points

Suppose we want to simulate data for an entire grid of points – this is done for training or testing a network with stationary sources. First we need to generate the grid. We can do this with `pointspec.cpp`. We specify a spacing, and it outputs the coordinates in a file of our choice. View the source code of `pointspec.cpp`. Try to follow how it generates points so that they are within the chamber illustrated in Figure 1.1. Notice the line `(z>-5.5 && z<-2.5)`. This excludes all points not within the first three centimetres of the chamber. For this test, let's generate points throughout the full chamber, not just the first three centimetres. To do this, in all lines replace `(z>-5.5 && z<-2.5)` with `(z>-5.5 && z<5.5)`. Now compile `pointspec.cpp` with `g++ pointspec.cpp -o ps`. Then enter the command

```
./ps2 1.8 grid.txt
```

This should create a grid with 279 points (in the `stdout`, 'm' represents the total number of points generated, and 'w' represents the number of points generated after exclusion). When generating large grids, this output to the screen can be time consuming. In this case, redirect such output in order to save time, as follows: `./ps2 0.2 grid.txt > screen.out`. Open `grid.txt` to see the coordinates.

Open `xepet.ffcards` again. Set the `XY` and `YZ` walls to be reflecting (the others absorbing). Set the number of events to `1`. Set the scintillation yield to `25000` per MeV. In this case, we will be simulating photo-detector data at each of these 279 points, when this a scintillation yield of 25000 photons per MeV, and XY and YZ reflectors. For each of these points we are only simulating 1 event. Suppose we want each of these points to have 511 KeV of energy. Edit `xepemaker.pl` so that the `SORC` line in each generated `xepet.ffcards` file corresponding to each coordinate has the correct energy. Then open `xepet.com` to specify where the 297 simulation output files should be sent. Then enter the following commands:

```
./xepemaker.pl grid.txt xepet.ffcards xepet.ffcards
./xecomaker.pl 0 279
nohup ./xecorun.pl 0 279 >& outputs.out &
```

The first two commands should each take no more than 2 minutes to execute, and the last step should take no more than 10 minutes. The format of the last command is geared for long simulation runs, where you wish to log out and not have your simulations interrupted. Once running is complete, enter your directory with simulation output. Convert the `hbook` files into `root` files by using `toroot.pl`. Then extract the data with `neural.C` or `photo.C`, or in whatever way you wish. If you use `neural.C`, it will create a vector **p**, that contains all photo-detector data. Open `Matlab`, and type in the name the file that `neural.C` generated, (without the .m extension). Then type `p=transpose(p)`. If you then type `p(:,i)`, where i is replaced by an event number, then `Matlab` will display the photo-detector data for that event; in `Matlab`, a(:,i), means display all rows of column i. Likewise, a(i,:), would be to display all of row i. In the case of very large simulation runs, it is sometimes worth splitting up the work amongst multiple processors. To do this, have an `xepet_batch` file for each processor. For example, if there are four processors, have `xepet_batch1`, `xepet_batch2`, `xepet_batch3`, and `xepet_batch4`. Use `xecomaker.pl` to generate `xepet.com` files for the first processor; e.g. `./xecomaker.pl 0 100`. Then edit `xepet.com` so that it uses `xepet_batch2` and, for instance, run `./xecomaker.pl 100 200`, etc. Then have a different `xecorun.pl` for each processor. For example, you may want to do `./xecorun1 0 100`, then `./xecorun2 100 200`, etc.

Now we need a target vector corresponding to the photo-detector vector. Modify `pointspec.cpp` so that the line `out << tpts[i].x << " " << tpts[i].y << " " << tpts[i].z << endl;` becomes

```
out << tpts[i].x << endl << tpts[i].y << endl << tpts[i].z << endl;
```

This just causes each spatial coordinate to be on a newline in the text output file. Recompile, and run

```
./ps pointspec.cpp gridtarg.txt
```

Edit `open(IN, "energyfull");` so that it reads `open(IN, "gridtarg.txt");`, and make the output file `open(OUT, ">gridtargs.m");`. This is just a way of storing the coordinates of the grid points in a vector in `Matlab`. If you open `Matlab`, and type `gridtargs`, and `t=transpose(t)`, you can see the *ith* grid coordinates with `t(:,i)`.

At this point we have vectors **p** and **t** in Matlab, and we can use them to test a neural network I have trained in Matlab. To learn how to do this now, skip to the *Matlab* section of the *Modelling and Regression Tutorial*. I recommend you do this before reading any other part of this document.

8.2.2 Photon Sources

Unlike with stationary sources, we do not need to create very many simulation files – usually only one. At least initially, we will only want to shoot a photon in from a couple different positions. For each position, we may want to repeat the process at least 1000 times. In this case we simply need to change the number of events to 1000 or more: it's unnecessary and a lot more difficult to create 1000 simulation files. This time we want to strip the information out of each event in one simulation file, rather than take a new event from a new simulation file, as was sometimes done in stationary source simulations.

Without loss of generality, let's assume we want one 100 event simulation file with a photon source: that is, we want to simulate shooting a photon into the chamber from a fixed position, and we want to repeat this simulation 100 times. Each time the photon enters the chamber is considered an event. For each event there is a chance the photon will interact with one or more electrons. At any given event, light seen by the photo-detectors will be the sum of the light produced at each interaction – the photo-detectors are not fast enough to isolate the light corresponding to each Compton interaction. We want to extract those events where there are one or more interactions, the photo-detector data for each of these events, the x, y, z coordinates of each interaction, and the energy the photon deposits at each interaction. This is done with `photon.C`.

Task: Simulate with a photon source

Open `xepet.ffcards`, and set the number of events to 100. Make the walls absorbing (DMAT 12 19 19 19). Scroll to the SORC line:

```
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  4.00 3.00 0.00 0. 0.  0. 0.   0.  0.  0.511 1.0  0. 0.
```

The particle type `-3` is an electron. Look up "Geant Particle Type" on google for a table. We need to change this to `-1` for a photon source. The particle gun is, by default, pointing in the upwards z direction. To rotate to point in the y direction, change the fourth argument from "0.00" to "90." (a 90 degree rotation about the x axis). For these purposes, though, you will not need to rotate the particle gun. The gun is initially pointing in the positive z direction, in the centre of the chamber (at 0,0,0). Change the particle gun origin to (0,0,-12). This way the gun is pointing straight towards the lower surface of the chamber. The chamber is 11.05 cm long in the z direction, so the gun is (-12+11.05/2=-6.475), 6.475 centimetres below the lower surface. A photon from this position would shoot straight through the centre. Set the sigma x and y positions to be 5 cm. This effectively gives the beam some width, and will distribute some of the initial interaction points throughout the chamber. To reach other areas of the chamber, (For further information talk to Fabrice Retiere or Leonid Kurchaninov, or me). Now, your `SORC` line will look like this (probably!):

```
SORC -1 0.  0.  -12.  0.  0.  0.  5.  5.  0.  0.  0.511 1.0 0.  0.
```

Run the simulation, and then convert it to a `root` file. Type `h999.Print()`. The number of "entries" you see are the number of events where one or more interactions took place (out of the total number of events you set in `xepet.ffcards`). To see the number of interactions at the zeroth such event, type:

```
h999.Draw("nconv", "", "", 1, 0)
```

The fourth argument of the `Draw` command is the number of events you want to display.

```
h999.Draw("nconv", "", "", 1, 14)
```

Would show the number of interactions at the fourteenth event where there was an interaction. (fifteenth if you start counting at 1). What does this command do?

```
h999.Draw("nconv", "", "", 3, 0)
```

Try to figure it out. Hint: try looking at `h999.Draw("nconv", "", "", 1, 0)`, `h999.Draw("nconv", "", "", 1, 1)`, `h999.Draw("nconv", "", "", 1, 2)`, `h999.Draw("nconv", "", "", 1, 3)`.

To see the x,y,z coordinates of the first interaction at the zeroth event, type

```
h999.Draw("x_conv[0]", "", "", 1, 0)
h999.Draw("y_conv[0]", "", "", 1, 0)
h999.Draw("z_conv[0]", "", "", 1, 0)
```

To see the energy deposited at the first interaction of the zeroth event, type

```
h999.Draw("e_conv[0]", "", "", 1, 0)
```

Take a look at `photon.C`, and the other extraction files. Try to understand the code.

8.3 Modelling and Regression Tutorial

8.3.1 Introduction

In this section you learn how to model the stationary source data you generated in the previous section. We have examples of photo-detector data corresponding to points in the chamber, and we want to run a regression. This regression can used to approximately reconstruct data generated from non-stationary sources; however, we do not wish to run a regression on non-stationary source data: this would be difficult, and there is no advantage to doing so. Please give this some consideration.

I have used a neural network to create a non-linear function that maps photo-detector input to points in the chamber. The most effective network I have is written in Matlab. It uses *tansig* transfer functions, and the *Levenberg-Marquardt backpropagation algorithm*. It uses a *mean square error* function **without** regularization. The network has the structure [32-24-17-12-3].

The trained network is a non-linear function. Going from input to output would be about 1000 mathematical operations. Ultimately, we would like to pre-program this function into an FPGA device, which can process each operation in nanoseconds. This is simple enough: we can easily extract the weights and biases from the neural networks I have trained in Matlab. This should be done when we are closer to programming an FPGA device.

One may wish to program the network in C++, as this could be convenient during preliminary stages of testing – before anything is coded into an FPGA device. There are some neural network libraries in `root`, see [4, 5].

To follow the upcoming tutorials, please download `xemodels.tar` at
http : *//laplace.physics.ubc.ca/People/agwilson/models.tar*

All `.mat` files are trained using a [32-24-17-12-3] network, with LM backpropagation, and a mean square error function with no regularization. (The networks trained with the most success). Here is a description of each file (also see table 2.1):

`rootNN.cpp`: C++ neural network
`makefile`: makefile to compile C++ neural network

`rtrain2.dat`: Sample training data for C++ neural network
`rtest.txt`: Validation data for C++ neural network
`absorbfull.mat`: Trained with grids 32, 33. Validation: 33. Test: 35, 36, 37
`absorb.mat`: Trained: 1, 2. Validation: 4. Test: 5, 6, 7, 8, 9, 10
`reflect.mat`: Trained: 11, 12. Validation: 16. Test: 13, 14, 15
`sidereflect.mat`: Trained: 17, 18. Validation: 22. Test: 19, 20, 21, 25, 26, 27
`sideyield.mat`: Trained: 23, 24. Validation: 28. Test: 25, 26, 27
`ensideyield.mat`: Trained: 29, 30. Validation: 31. Test: 25, 26, 27
`mydivide.m`: Data division function used for training
`azposerr.m`: For plotting pos. reconstruction err. as a function of z pos
`testacc.m`: For testing the accuracy of position reconstruction
`regression.m`: For running a regression on test data reconstruction in each coordinate
`rerrors.m`: (Used after running testacc.m). For making plot of % under error vs. error
`mainenergy.m`: Use for plotting energy reconstruction err. as a function of pos
`echeck.m`: Used for measuring error in energy reconstruction
`sumv.m`: Used by many of the listed `.m` scripts
`eres.m`: Another program for checking energy resolution
`estats.m`: (Must run echeck.m or eres.m first). Makes scatter plot of energy recon. err. vs. position
`convert.m`: Converts grid vectors from format used by Matlab neural network to format for rootNN
`ksearch.cpp`: Used for solid angle correction factor mapping
`wkeygrid.txt`: `ksearch.cpp` accesses this grid in mapping
`solidangle.m`: Generates `wkeygrid.txt`
`grids.tar`: CONTAINS:

`grid1.m, grid2.m, ..., grid37.m`

:
`targ33825.m, targ22165.m, targ3811.m, etc.`

8.3.2 C++ and Root

Luca Doria (a member of Doug's group) worked with me to write a C++ neural network that uses root libraries, in July 2008. However, the Levenberg-Marquardt backpropagation algorithm is not available through root. This algorithm must likely be coded into a C++ network for it to perform as well as the Matlab neural network I am using. If you have trouble during this tutorial, Luca can help you. But please try hard to figure things out yourself first.

Sample training and testing section

Put `rootNN.cpp` and the `makefile` into one directory. Compile `rootNN.cpp` by typing `make` at the terminal. Read the code in `rootNN.cpp` and try to understand it. Reference root's neural network function documentation [5]. Here are a few key lines in the code:

Define the network (32 input, 3 output)

```
TMultiLayerPerceptron *mlp = new TMultiLayerPerceptron("x1,x2,x3,x4,x5,
x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,
x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32:128:x,y,z",ntuple);
```

Set the learning method

```
mlp->SetLearningMethod(TMultiLayerPerceptron::kBFGS);:
```

Set epochs, train, set updates for user
```
mlp->Train(400,"text,graph,update=10");
```

Neural network testing file. 3811 points
```
in2.open("rtest.txt");
```

Read Training Data
```
in.open("rtrain2.txt");
```

Store Weights
```
mlp->DumpWeights("NNWeights-128neurons.dat");
```

Exports network as C++ function
```
mlp->Export("RootNet","C++");
```

Task 1: Compiling and running network

Make sure `rtrain2.dat` and `rtest.txt` are in the same directory as `RootNN.c` and `Makefile`. Type `make RootNN`. Then run the executable that appears. Notice how the network trains. Get root to generate a C++ function representing this network, and try running the function (just modify a line in RootNN.c).

Task 2: Reproducing results, modifying data

In the previous task, the network used grids 11 and 12 for training, and grid 13 for testing. Put `grid11.m`, `grid12.m`, `grid13.m`, `target3811.m`, `target15846.m`, `target55332.m`, and `convert.m` into the same directory. You can load a grid simply by typing its name. For example, to load grid11.m, type `grid11` into the `Matlab` command prompt. Try to recreate `rtrain2.txt`, and `rtest.txt`, and retrain the network. Reproduce the same results as you had in the previous task. Try training on some different grids.

Task 3: Gnuplot, graphing error and targets

Using the output of the neural network, you should be able to graph both the targets, and the error, and the error as a function of the targets. At the gnuplot command prompt, you can type `plot ''datafile.txt'' using 1:2:3`, and it will plot the first, second, and third columns of `datafile.txt`. If you run into problems, you can try talking to Luca Doria (*luca@triumf.ca*).

8.3.3 Matlab

In order to follow this tutorial, it's important that you use `Matlab` version 7.5.0338 (R2007b) or later. Most `Matlab` installations on the TRIUMF computers are earlier versions. The *hyper* computer network in Hennings (the main Physics building at UBC) has this version of `Matlab`. If you are a UBC physics student, you should be able to request an account on this network (you likely already have one), and ssh to it from a computer at TRIUMF.

If you've finished **Task 2** of the **Stationary Sources** tutorial, then you have an input vector `p`, and a target vector `t`. Make sure my `models.tar` is decompressed in the `Matlab` directory you're using. You can check this by typing `dir` or `ls` into the matlab command prompt. Many `MSDOS` and `*nix` commands work in the `Matlab` command prompt. For example, you can type `pwd` to see which directory you're in, or `mkdir` to make a directory, `which (command)` to see where the binary file for a command is. Some other useful commands are `w`, `who`, `delete (filename)help (command)`, and `clear (variable)`. Try typing `help clear`. Use `help` to figure out the other commands. Let's make a simple network to model the input and target vectors you've made. Type:

```
net = newff(p,t,[20 12], 'tansig', 'tansig', 'purelin', 'trainlm').
```

This makes a [32-20-12-3] network, with tansig transfer functions in the two hidden layers, and a linear transfer function in the last layer. It uses the levenberg marquardt algorithm. Type `p(:,1)`, and `t(:,1)`. Type `sim(net,p(:,1)`. What do you get? Now create a new vector, w=p(:,1). Change the entry at the 21st photo-detector by a tiny bit (a couple photons). For example, type `w(21)`. If the number returned is 19, then type `w(21)=17`. Now try `sim(net,w)`. I bet this give something quite a bit different than `t(:,1)`: the network isn't reconstructing very well at the moment, it needs to be trained. Before training, let's check a few things. Type:

```
net.performFcn
```

This command will probably return the text `'mse'`. That means the network is using the mean square error function. You can change it to use a regularized mean square error function by typing: `net.performFcn='msereg'`. Then you may want to adjust the *performance ratio*. You can access it with `net.performParam`. Low memory can be a big problem when training with levenberg marquardt. Often training will start, and then it will stop due to a memory error. This can be extremely frustrating if you have been waiting days for training to complete, and then you turn on the computer monitor to see that memory ran out and your time was *completely* wasted!!! (I wouldn't know). To adjust memory consumption, type `net.trainParam.mem_reduc`. By default the factor is 1. The higher you make it, the less memory is consumed. A factor of 140 or greater is what I often use. To be on the safe side, you may want to go for 300 or more. Type `net.trainParam` to see all the training parameters you can adjust. Let's do a sample training session. This is just a practice, so let's only do 10 epochs. Typically, I do 400 epochs. Type:

```
net.trainParam.epochs=10
```

And let's get updates on the network training every 2 epochs:

```
net.trainParam.show=2
```

Finally, let's train!

```
[net tr] = train(net,p,t)
```

Every 2 epochs we are given the gradient value, and the mean square error. You should see both getting smaller as training progresses. Now let's type

```
sim(net,w)
```

This uses the vector **w** as input into the network we trained. It should be closer to t(:,1) than it was before. It's still probably not *that* close, since we didn't train for very long, and the structure of this

network is fairly primitive. Try generating some more data, and training it longer, and testing it. *Try writing a matlab script that tests the average reconstruction accuracy of the network, on a fresh set of testing data you've generated and imported into matlab.* Decide on how you're going to measure accuracy. Once you're finished playing with the network, you can save it by typing

```
save yourfilehere.mat
```

I have some examples of more complicated networks that I've built and trained in `models.tar`. Let's load `sidereflect.mat`. Type:

```
load sidereflect.mat
```

You will see a bunch of variables load, including a network. This is the network I trained for reconstructing interactions when there are reflectors on the slanted sides (YZ) of an LXe chamber. Try generating a 100 testing points with YZ reflectors (look at `DMAT` in `xepet.ffcards` to set the YZ sides to be reflecting (12 19 14 19). Try reconstructing those hundred points with this network, and use your program to determine the error with which my network reconstructs.those points.

I've written some tasks that will further familiarize you with my work. I do not provide walk-throughs: you'll learn more doing these independently. But, if you would like some clarification or help, please feel free to e-mail me (*aglwilson@gmail.com*).

Task 1: Graphing results

Use `rerrors.m` and `testacc.m` to reproduce the graphs of "% of points under reconstruction error vs. reconstruction error" in my thesis. Also recreate the regression plots and the plots of "% energy recon. under error vs. error". Recreate the "error vs. position plots" (hint: use `scatter`). You may want to go through some simple online `Matlab` tutorials, such as this one [6].

Task 2: Photon Sources

Look at `photonave.C`, `photonce.C`, and `photondist.C`. Get the targets and inputs into `Matlab` using `photonave.C` and `photonextr.C`. `photonave.C` will give you the target data, and `photonextr.C` will give yout he photo-detector data. Load the network `absorbfull.mat`. Run the input data through the network, and compare the output to the target data (using `testacc.m`). Repeat this process using `photonce.C`. Using `photondist.C` requires a little more work. Extract the photo-detector data, reconstruct it in `Matlab`, and then bring the reconstructed points back to your directory in an appropriate format for `photondist.C`. `photondist.C` will read these points in, and perform accuracy tests. If you need help, e-mail me at *aglwilson@gmail.com*.

Task 3: Reproduce my results

Try reproducing some of my "best results" in this document, *exactly*. This should not take long (e.g. don't take days figuring this out), given that I have made both the networks I trained, and testing data I used, immediately available to you. If you need help, e-mail me.

Task 4: Extracting the network

Once the network is trained, it's just a non-linear function of 32 variables, with weights and biases. Figure out how to extract these weights and biases. Get them into a `C++` routine. Eventually we'll want to put these networks into an `FPGA` device. See how fast the extracted `C++` network is, and how much memory it consumes.

Task 5: Modify xepet.ffcards

Run or modify xepet.ffcards (either is possible) in a way that does not require multiple files to store simulation data for multiple stationary sources at locations you choose. You may need to look in the ffcards source for this – contact me.

Bibliography

[1] A.G. Wilson. *Position and Energy Reconstruction from Scintillation Light in a Liquid Xenon Gamma Ray Detector designed for PET.* http://laplace.physics.ubc.ca/People/agwilson/WilsonThesis.pdf UBC Honours Physics Thesis, April 2008.

[2] A.G. Wilson. *Position Reconstruction from Light in a new LXe PET.* http://laplace.physics.ubc.ca/People/agwilson/lxelightpres1.pdf Open office presentation. April 2008.

[3] A.G. Wilson. *Position and Energy Reconstruction from Light in a new LXE PET (Continued Progress).* http://laplace.physics.ubc.ca/People/agwilson/lxelightpres2.pdf Open office presentation. July 2008.

[4] CERN. Neural Network Root Example. http://root.cern.ch/root/html/examples/mlpHiggs.C.html

[5] CERN. Neural Network functions in Root. http://root.cern.ch/root/html/TMultiLayerPerceptron.html

[6] Matlab Tutorial. http://www.cyclismo.org/tutorial/matlab/

[7] Simulation Sources. Last updated Sep 8, 2008. http://laplace.physics.ubc.ca/People/agwilson/simufiles.tar

[8] Modelling Sources. Last updated Sep 8, 2008. http://laplace.physics.ubc.ca/People/agwilson/xemodels.tar