University College London

Department of Physics and Astronomy

Physics MSci Project Final Report

Design of a Detector for Fast Treatment Plan Verification in Proton Radiotherapy

Author
Saad Shaikh

Supervisors
Dr. Simon Jolly
Prof. Ruben Saakyan

1st October 2018 – 22nd March 2019

UCL DEPARTMENT OF PHYSICS AND ASTRONOMY

Submission of coursework for Physics and Astronomy course PHASM201/PHAS3400

Please sign, date and return this form with your coursework by the **specified deadline** to the Departmental Office.

DECLARATION OF OWNERSHIP

I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.

I confirm that all work will also be submitted electronically and that this can be checked using the JISC detection service, Turnitin®.

I understand that the work cannot be assessed unless both hard copy and electronic versions of the work are handed in.

I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

Signed 

PrintName SAAD SHAIKH

Dated 21/03/2019

Title	Date Received	Examiner	Examiner's Signature	MARK

Abstract

Proton beam therapy (PBT) is an alternative to X-ray radiotherapy as a method of treating cancers with ionising radiation. The use of protons offers considerable advantages to X-rays – namely the potential to deliver highly-localised doses of radiation, sparing healthy tissue. Advancements in technology and larger public awareness have resulted in PBT becoming increasingly used in healthcare services, however many technical challenges prevent PBT from reaching its full potential.

Treatment plan verification, or patient-specific quality assurance (patient QA), in PBT requires detailed information about the volumetric dose deposition within an instrumented volume, to ensure the accurate delivery of dose for a given treatment plan. Current methods of patient QA are time-consuming, due to the repeated scanning of water phantoms. A collaboration between the High Energy Physics group at University College London and the Physics Department at the University of Birmingham is developing a prototype system for fast patient QA.

Single-module plastic scintillator-based calorimeters were developed at UCL for the SuperNEMO high energy physics experiment that provide fast, accurate measurements of proton energy; these are combined with 2D particle position information from the Birmingham silicon trackers developed for the PRaVDA proton CT project to reconstruct the position and energy of each proton. This provides the potential for reconstructing the 3D dose deposition for individual protons and therefore build up the complete volumetric dose distribution for a given treatment plan. This would require the treatment plan to be delivered only once, rather than repeatedly as is the case for existing methods.

Presented in this report are preliminary measurements made at the 36 MeV Birmingham University cyclotron with a 3 cm PMT-based calorimeter module coupled to a single PRaVDA tracker module. Significant improvements were made to the analysis routines of energy measurements made with the calorimeter, to reliably calculate proton energies in an external trigger setup. Single-proton measurements made by the tracker were then successfully correlated with energy measurements, and the position and energy distributions for a variety of collimator configurations were reconstructed.

Contents

1 Introduction

The goal of this project is to provide proof-of-principle for an early prototype of a detector that could allow for fast treatment plan verification in proton radiotherapy. This report will first cover an overview of cancer, illustrating the importance of the continued development of treatment methods and how radiotherapy fits in with other modalities. It will then discuss the principles of radiotherapy and the differences between X-ray and proton radiotherapy, showing the advantages of the latter. Treatment plan verification is then introduced, where details on its necessity and its current methods are given, which will naturally provide justification for this project. The principles of the prototype detector will then be discussed, where a brief summary of scintillator and semiconductor detector physics is also given for the reader. The specifications of the prototype detector are then given along with a more detailed summary of the project goals. The experimental procedure is then discussed, which will entail the desired objectives and the data analysis procedures. Final results are then presented along with concluding remarks.

2 Cancer: Biology, Diagnosis and Treatment

Cancer continues to serve as one of the most challenging human diseases to-date: according to Cancer Research UK [1], in 2016 there were 163,444 deaths from cancer in the UK and each year there are more than 360,000 new cancer cases, costing the National Health Service (NHS) £5 billion per year [2]. 1 in 2 people in the UK born after 1960 will be diagnosed with some form of cancer in their lifetime.

R. W. Ruddon defines cancer as an abnormal growth of cells caused by multiple changes in gene expression leading to an imbalance of cell proliferation and cell death [3]. Biological cells naturally undergo division via mitosis or meiosis as part of their life cycle [4]. These delicate processes can introduce mutations in cellular genetic code, typically as chromosomal and point mutations [5]. With sufficient growth of such abnormal cells, a tumour develops, which is classed as being either benign or malignant [3]. Benign tumours are considered to be less dangerous than malignant tumours – as the latter is capable of disseminating mutated cells throughout the body, through metastasis. In 2000, Weinberg and Hanahan defined the six hallmarks of cancer, which were updated in 2011 with the addition of four others [6, 7]. These are:

1. Self-sufficiency in growth signals
2. Desensitisation to anti-growth signals
3. Evasion of apoptosis (cell death)
4. Angiogenesis
5. Indefinite cell division
6. Metastatic capacity
7. Abnormal metabolism
8. Evasion of immune system
9. Inflammation
10. Genomic instability

Diagnosis of cancer is typically done after indication of the disease from one or a combination of the following: screening tests, medical imaging and the appearance of symptoms [8]. Imaging can be performed

in several ways: X-ray imaging, computerised tomography (CT) imaging, magnetic resonance imaging and nuclear medicine [9]. Treatment for cancer broadly falls into three categories: palliative, curative and preventative [10]. Palliative treatments seek to relieve the symptoms of cancer; curative treatment methods aim to eliminate the cancer completely; and preventative treatments attempt to reduce the risk of cancer recurrence. The main curative methods are:

1. Surgery – the excision of a solid tumour from the body. Whilst effective, it can only be used for tumours that are localised and accessible via surgical methods. Typically 45% of patients in the UK will have surgery to remove their tumour as part of their primary treatment [1].

2. Chemotherapy – the use of cytotoxic agents to interfere with mitosis in non-specific cells [11]. An effective form of treatment, but significant side-effects occur in the vast majority of patients due to damage to both healthy and cancerous cells. 28% of patients in the UK will have chemotherapy, often in conjunction with other, more localised treatments [1].

3. Radiotherapy – This treatment seeks to initiate apoptosis in cancerous cells by disrupting cell DNA with ionising radiation, and minimising the dose delivered to healthy cells. Whilst more localised than chemotherapy, the delivery of radiation to healthy cells can cause carcinogenesis (cancer formation) [12]. As the method is non-invasive, it is highly useful for the treatment of inoperable tumours. 27% of patients in the UK undergo radiotherapy as part of their treatment [1]. Both internal and external methods exist – the latter is more typical and broadly comes in two forms: X-ray radiotherapy and proton radiotherapy.

3 Radiotherapy

The principle of radiotherapy is to use ionising radiation to damage the DNA of cancerous cells to initiate cell death. Ionising radiation is defined as radiation that can transfer energy to the electrons of atoms or molecules, such that an electron can be liberated, leaving the atom or molecule ionised [13]. Organic molecules are primarily bonded together with covalent bonds, where an electron pair is shared between two atoms. If an electron is liberated from this pair, the bond breaks and the molecule breaks down. For DNA, which has a double-helix structure, this takes place as double-strand breaking [5]. A double-strand break will often lead to cell death due to neither DNA strand being viable as a template for repair after the break. Both X-rays (photons) and protons are classed as ionising radiation, however the mechanisms through which they interact with matter are very different.

It is important to clarify the distinction between the term “energy” and the term “dose”. The dose is defined as the quantity of energy deposited by ionising radiation in a medium, per unit mass, measured in grays (Gy) and is the term of choice when discussing radiotherapy [12]. As will become apparent later, the energy of single particles is also discussed, wherein lies the distinction between the two terms: energy is used to describe a single particle, whereas dose is used to describe the effect of a collection of particles.

3.1 X-ray Radiotherapy

X-ray radiotherapy utilises photons to irradiate tumours. It is by far the most common and best understood method of external radiotherapy, with the earliest examples dating back to the early 20th century [14]. Photons are indirectly ionising radiation as they do not possess electric charge, and so cannot ionise matter directly. Photons can however transfer energy onto atomic electrons creating free, energetic, charged particles along their path. These electrons can then go onto to cause double-strand breaks in DNA molecules. Photons transfer energy via three processes [13]:

1. The photoelectric effect: the excitation and liberation of electrons in atoms through absorption of a photon. The cross-section for this process dominates in the KeV energy range, and then rapidly decreases in the MeV range.
2. Compton scattering: the scattering of photons off (free) electrons. Like the photoelectric effect, its cross-section is largest in the KeV range, dropping in the MeV range. However the cross-section is smaller than the photoelectric effect in the KeV range, and larger in the MeV range.
3. Pair production: This process requires photon energies of at least 1.022 MeV (i.e. twice the rest-mass of the electron). In the electric field of an atomic nucleus (necessary for momentum conservation), a photon can produce an electron-positron pair. The cross-section for this process continues to rise in the MeV range.

For clinical applications, photons of energy 1-20 MeV are typically used [15], making Compton scattering and pair production the primary mechanisms of energy-loss. Photons are highly-penetrating and do not have a finite range in matter. Instead their interactions are probabilistic: a photon will either interact with matter and be lost, or not interact at all, having the same energy as when it entered the body. Therefore, photon intensity loss in a medium follows an exponential relation with the depth of the material [13]:

$$I = I_0 e^{-\mu x} \tag{1}$$

where I is the intensity at depth x, I_0 is the initial intensity and μ is the attenuation coefficient, a description of the probability of interaction per unit length of a photon traversing a medium. This relation is visualised as the purple line in Fig. 1. The initial build-up period for the photon dose is known as the "skin-sparing" effect, and occurs due to the finite time taken for the secondary flux of electrons to accumulate (recalling that it is the electrons that actually deliver the dose) [16]. This is a useful feature, and becomes larger with increasing photon energy. However, the exponential drop of dose is undesirable: only a fraction of the full dose is delivered to the tumour itself and a non-negligible amount is delivered beyond the tumour. Therefore to sufficiently irradiate deep tumours, significantly more damage will have to be done to healthy tissues. If the tumour is close to sensitive organs (as is often the case with inoperable cancers), the non-negligible tail can result in serious side-effects for the patient post-treatment [17]. In practice, this is partially off-set by treating from multiple angles, centering on the tumour, to minimise the dose delivered to healthy tissue [18].

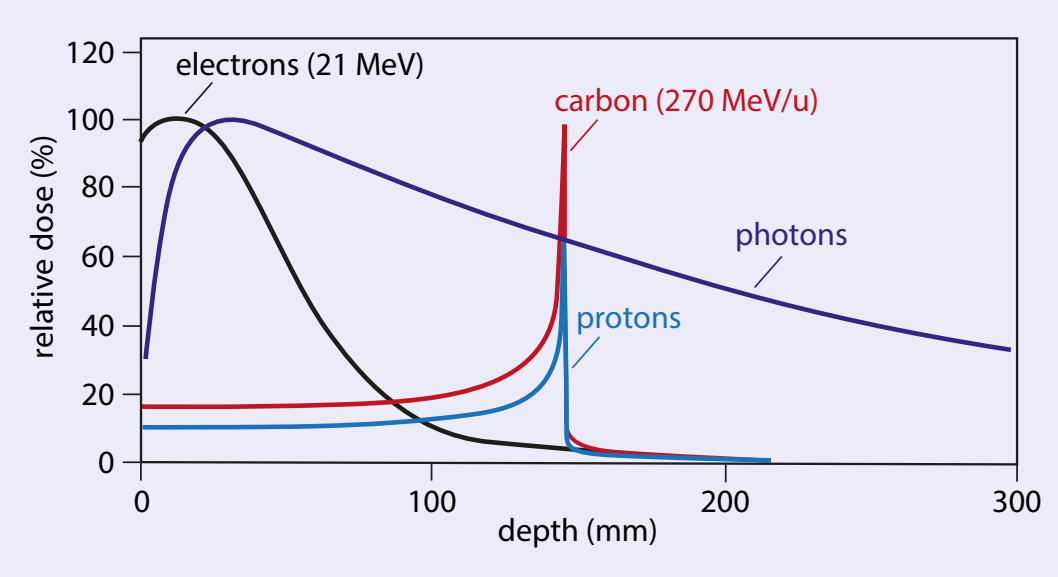


Figure 1: The variation of delivered dose with depth for several types of radiotherapy. The dose delivered by photons gradually decreases with depth, whereas the dose from protons culminates in a Bragg peak [19].

3.2 Proton Radiotherapy

Proton radiotherapy utilises protons to deliver ionising radiation to cancerous cells. Unlike photons, protons are electrically charged and are therefore directly ionising. More notably, protons have a well-defined range in matter for a given proton energy. Protons will primarily interact electrostatically, and their energy loss is described by the Bethe-Bloch equation [20]:

$$\left\langle -\frac{dE}{dx} \right\rangle = Kz^2 \frac{Z}{A} \frac{1}{\beta^2} \left[\frac{1}{2} \ln \frac{2m_e c^2 \beta^2 \gamma^2 W_{max}}{I^2} - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \tag{2}$$

where E is the energy, x is the depth, Z is the atomic number of the medium, A is the mass number of the medium, z is the charge of the incident charged particle, I is the mean ionisation potential, W_{max} is the maximum kinetic energy transfer upon collision of a particle with an electron in the medium, c is the speed of light in a vacuum, m_e is the mass of the electron, $\beta = \frac{v}{c}$, where v is the velocity of the particle, and $\gamma = \frac{1}{\sqrt{1-\beta^2}}$ is the Lorentz factor. $K = 4\pi N_A r_e^2 m_e c^2$, where N_A is Avogadro's number and r_e is the classical electron radius. $\delta(\beta\gamma)$ is a density correction term.

Protons also interact via the mechanisms of Multiple Coulomb Scattering (MCS) and inelastic scattering off atomic nuclei. The former is important when considering the angular spread of a proton beam, whereas the latter has little effect on the overall dose deposition and beam shape [21, 22].

At clinical energies (50-250 MeV [14]), the $\frac{1}{\beta^2}$ term in (2) dominates, resulting in the characteristic "Bragg peak" shown in light-blue in Fig. 1. As a proton slows down, the probability of interaction with an atomic electron increases, thus the proton loses more energy and further slows down [23]. This positive feedback loop is the main advantage of proton therapy: the majority of the dose is delivered to a highly localised

region (where the proton slows to a stop) and with very little dose delivered in the region beyond. Given that the energy of a proton defines well the range in matter, several Bragg peaks can be superimposed to deliver a consistently large dose over a tumour region [24]. This is known as a "Spread-Out Bragg Peak" (SOBP) and is shown in Fig. 2.

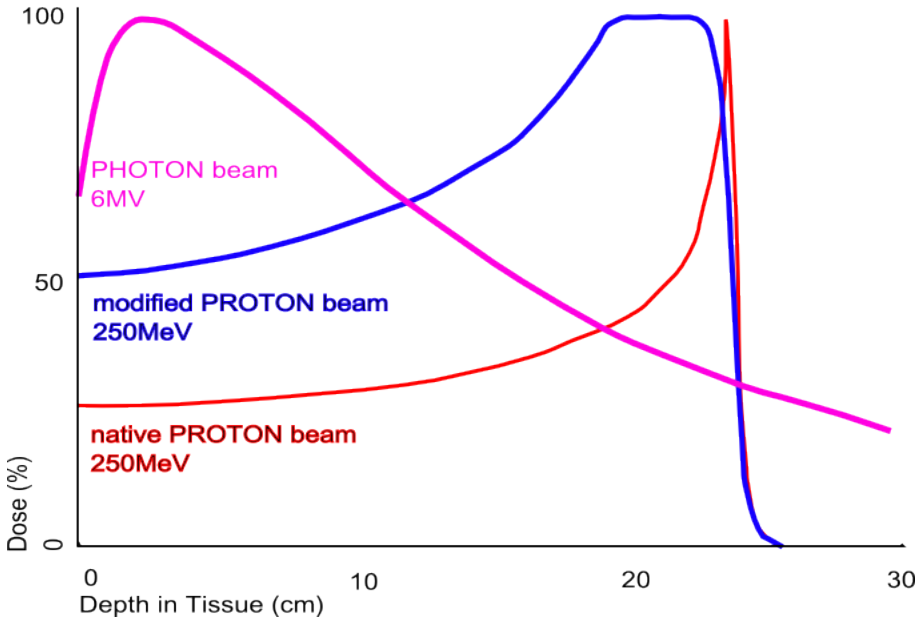


Figure 2: The delivered dose by superimposing several Bragg peaks from protons of different energies. Photon dose variation and pristine (single) Bragg peak is shown for comparison [25].

This behaviour makes proton radiotherapy an attractive alternative to photon-based radiotherapy: a large dose can be delivered to a tumour, with minimal collateral damage to healthy tissue. Proton radiotherapy typically offers 60% less total integral dose compared to photon radiotherapy, thus making it optimal for the treatment of paediatric patients, who are more susceptible to radiation side-effects [26]. However, this introduces new technical difficulties. The highly-localised nature of the dose deposition requires the beam to be delivered accurately, with little margin for error (typically to mm precision, although practices vary across institutes [27, 28]). If the beam is delivered incorrectly, cancerous tissue may receive no dose at all (under-shooting), or healthy tissue may receive the full dose instead (over-shooting). Given that radiotherapy is often used for treating tumours near vulnerable parts of the body (e.g. the spine or the brain stem), an incorrectly delivered dose could lead to devastating effects for a patient. This necessitates the need for treatment plan verification, otherwise known as patient-specific quality assurance (QA).

3.3 Treatment Plan Verification

In both X-ray and proton radiotherapy, a designed treatment plan must be verified before treatment, to ensure optimal patient safety. In the context of proton radiotherapy, a patient's treatment plan is defined as the sequence of proton fields that have been determined to be the best option for treating the patient. An example of a the difference between an X-ray treatment plan and a proton treatment plan is shown in Fig. 3.

A patient will first undergo a CT scan to map the part of the body undergoing treatment [14]. A treatment planning system then formulates the treatment plan by converting this CT scan into relative proton stopping powers, which is the energy lost by a charged particle per unit distance in a medium. This is typically done via analytical pencil-beam dose calculation algorithms to project the range of protons based on the water-equivalent depth in the patient [29]. The use of CT scans to formulate the treatment plan introduce an uncertainty of 1-3 mm in the location of the Bragg peak, due to the differences in the interactions of photons and protons in the medium [30]. Other uncertainties include those from CT noise, resolution and artefacts. Treatment plan verification determines whether the range and dose predictions for the proton beams are correct within an acceptable uncertainty.

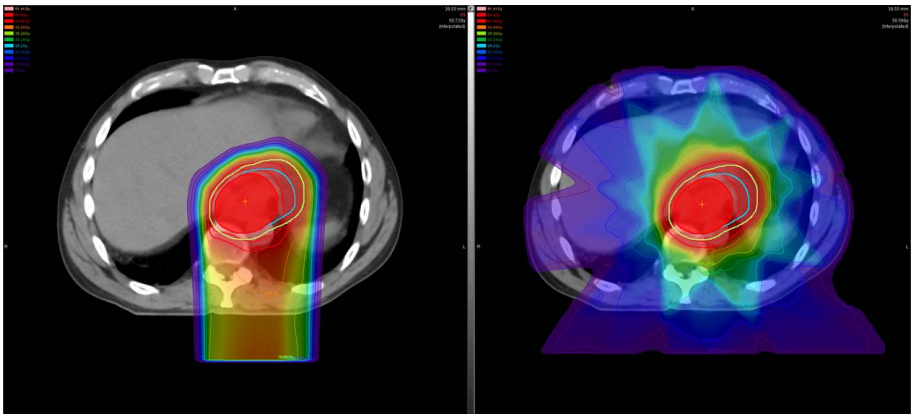


Figure 3: A treatment plan for a case of esophageal cancer, showing the difference in planned dose deposition between proton (left) and X-ray radiotherapy (right) [31].

A treatment plan is typically verified by delivering the plan to a water tank dosimeter, water phantom, or other detector formed of layers of ionisation chambers interspersed with water-equivalent material [32]. Water-equivalent material is chosen as a close approximation of human tissue [23]. However, this process is typically time-consuming: the limitations on how close dosimeters can be placed together mean that the treatment plan has to be repeatedly delivered and the entire volume of the water phantom must then be scanned piece-wise [27]. After each measurement, the dosimeter is moved to a new position within the water volume and the treatment plan is then redelivered. This takes valuable time away from actual patient treatment. The focus of this project is to develop a detector that eliminates the need for repeated treatment plan delivery in patient QA, to shorten the process from more than an hour to just a few minutes.

4 Prototype Detector for Fast Treatment Plan Verification

Treatment verification requires complete information about the volumetric dose deposition of protons within an instrumented volume: in other words, the position and energy of protons in 3D. It is proposed that a silicon-based semiconductor tracker could be used to track 2D spot (collections of protons) position, and energy measurements could be found using a scintillator-based integrating range telescope, which acts as water-equivalent material. Such a device would not require repeated piece-wise measurements of energy and

position and could rotate with a clinic gantry, eliminating the need for realignment. While the technical details of the device are beyond the scope of this project, the purpose of this experiment is to provide proof-of-concept for the simultaneous use of such detectors in a scaled-down version that operates on the same principle.

The setup used in this experiment is designed to reproduce the 3D deposition of protons at much lower particle rates than are used in clinical applications. This is referred to as single-proton counting, as individual proton position and energy are measured. At clinical rates, it is not possible to resolve individual protons, but single-proton counting is sufficient for the demonstration of the detector principle. Before the details of the simple prototype are discussed, a broad overview of the physics of scintillator and semiconductor detectors is given, to better illustrate the motivation for their use.

4.1 Scintillators: A Brief Overview

The scintillator operates on the phenomenon that in certain materials, when ionising radiation excites an electron of an atom to a higher energy state (through photon exchange), a photon in the visible range will be emitted when the electron de-excites [13]. Scintillators are one of the most commonly used technologies in high-energy physics, having applications in tracking, time of flight measurements, particle identification, as well as calorimetry (to name a few). A few definitions [33]: scintillators exhibit the property of *luminescence*, which is the emission of visible light upon de-excitation. The term *fluorescence* is used if this emission is within 10^{-5} s of the initial absorption whereas *phosphorescence* or *afterglow* is used if the emission takes longer (say, if the excited state is metastable).

There are several types of scintillator, which have different scintillation mechanisms and properties. The scintillator used in this experiment is a polystyrene-based plastic scintillator (PS), which boasts a decay time (the time taken for photon emission after electronic excitation) on the order of nanoseconds [20]. A PS is made up of organic molecules (known as its *base*), with its sensitivity to ionisation arising from the benzene ring found in the polymer [33]. The benzene ring has free valence electrons that are not associated with any particular atom in the molecule, but instead reside in π-orbitals, essentially rendering the electrons delocalised. In order to make scintillators transparent to their own radiation, that is to stop photon re-absorption, the base is typically doped with a *fluor*, which has slightly different energy levels to the base [34].

Both singlet and triplet π-electron energy levels of a base feature sub-levels that arise from vibrational fine-structure [33]. Upon excitation from the singlet ground state, the electron will be promoted to one of the vibrational sub-levels of the excited singlet energy level. This electron will then undergo *vibrational relaxation* or *internal degradation* to the normal excited singlet energy level. When the electron then finally de-excites to one of the ground state energy levels, a photon (in the visible range) will be emitted that is of a longer wavelength than the initial excitation (the so-called *Stokes' shift*) [20]. A similar argument applies to the triplet states, however the transfer between singlet and triplet states typically has a longer decay time, and can be a source of afterglow. The addition of a fluor can improve the efficiency of this process: after the initial excitation, relaxation can occur but instead to one of the excited energy levels of the fluor through

Förster energy transfer [33]. This results in a larger Stokes' shift and faster decay times, both of which are desirable. A simple schematic of this process is shown in Fig. 4.

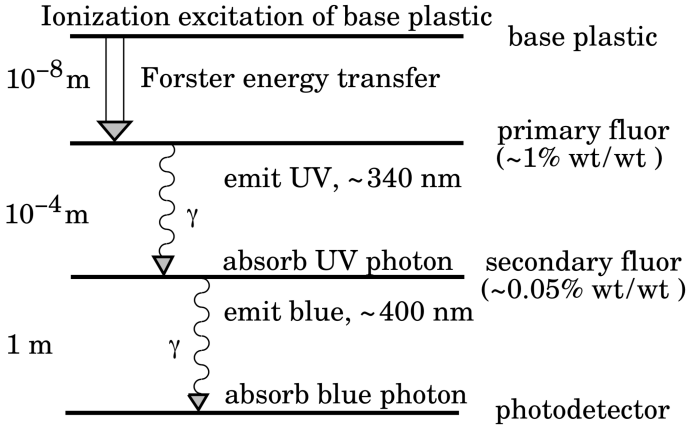


Figure 4: Schematic of the scintillation "ladder" depicting the operating mechanism of an organic scintillator. Approximate fluor concentrations and energy transfer distances for the separate sub-processes are shown [20].

Note: the fluor should not be confused with an *activator*, which is an impurity added to an inorganic crystal scintillator but broadly serves the similar purpose of improving scintillation light output by providing additional routes for electron de-excitation [20]. A *sensitiser* serves somewhat of a similar purpose as a secondary fluor; it is a small concentration of ions added to an inorganic scintillator to improve the performance of the activator (but could also quench the light output).

The qualities of a good scintillator are [33]:

1. High efficiency in the conversion of excitation energy into fluorescent radiation.
2. Transparency to its fluorescent radiation to allow for the passage of photons.
3. Emission in the spectral range consistent with the spectral response of existing photomultipliers.
4. A short decay constant.

However, scintillators are not typically used for calorimetry due to the non-linear response in energy transfer to light output for large rates of energy transfer. This is where electrons are excited but do not radiate a photon upon de-excitation, reducing the fast decay component from excited singlet states. The first successful semi-empirical model to explain the non-linearity effects was put forward by Birks [35]:

$$\frac{dL}{dx} = \frac{A\frac{dE}{dx}}{1 + kB\frac{dE}{dx}} \tag{3}$$

where: $\frac{dL}{dx}$ is the light output per unit length, $\frac{dE}{dx}$ is the energy transfer per unit length, A is the absolute scintillation efficiency and kB is a parameter that relates the density of ionisation centres to $\frac{dE}{dx}$. Non-linearity effects are most pronounced in heavier particles, which have more pronounced Bragg peaks. For

protons at energies used in this experiment however, one can reasonably assume linearity [33]. Another consideration in efficiency comes from the conversion of light to current in the PMT, where saturation effects can occur with large light output from the scintillator.

While PSs are reliable, cheap and flexible, they do have some disadvantages [20]:

1. Subject to ageing.
2. Suffer from radiation damage.
3. Degrade upon exposure to certain chemicals (e.g. oils and solvents).
4. Are not temperature resistant.
5. Liable to surface crazing, which damages light output due to the reliance on total internal reflection.

4.2 Semiconductors: A Briefer Overview

Semiconductor detectors operate on a principle similar to that of ionisation chambers and they were originally designed to measure particle energy. However, over the last few decades, semiconductors have become increasingly useful in particle tracking [34].

Semiconductors are crystalline materials whose outer electrons exhibit band structure, such that there is a small energy gap (on the order of eV) between the valence band (VB), where electrons are bound to their lattice atoms, and the conduction band (CB), where electrons are delocalised and able to carry a current [36]. Common examples are silicon (used in this experiment) and germanium. At 0 K, all the electrons in a semiconductor occupy the VB, while at room temperature, thermal excitation can promote a few electrons to the CB. When an electron is excited to the CB, it leaves a hole in the VB, which behaves like a positron [37]. That is to say, a neighbouring electron in the VB can jump to fill the hole, leaving a hole where it used to be. If the process repeats, it appears as if a positive charge carrier is moving through the material. Therefore in a semiconductor, charge carriers arise from electrons in the CB and holes in the VB.

Typically, semiconductors are doped with an impurity to enhance the number of electrons and holes. Both silicon and germanium are tetravalent atoms, hence the addition of a trivalent or pentavalent impurity can increase the number of electrons and holes respectively [36]. This introduces the distinction between n-type and p-type doped semiconductors: the former has a greater concentration of electrons whereas the latter has a greater concentration of holes. The respective donor and acceptor levels are closer to the CB and VB, allowing for easier electron/hole transfer [37].

Practical applications utilise both n- and p-type semiconductors together – a simple example is the juxtaposition of an n-type with a p-type semiconductor to create a *pn-junction* [38]. Due to the difference in the concentration of electrons and holes the n- and p-types, there is a diffusion of electrons and holes across the junction. The electrons fill up holes in the p-side and the holes are filled by electrons in the n-side. Therefore, the n-type side becomes positively charged, while the p-type side becomes negatively charged.

An electric field is then established across the junction, halting the diffusion, leaving a *depletion zone* in the junction where there are no mobile charge carriers [37]. This allows for the detection of ionising radiation: ionising radiation will liberate electron-hole pairs, which are then swept across the pn-junction, generating a current that is proportional the the energy loss, which can be readout [37]. This is shown in Fig. 5.

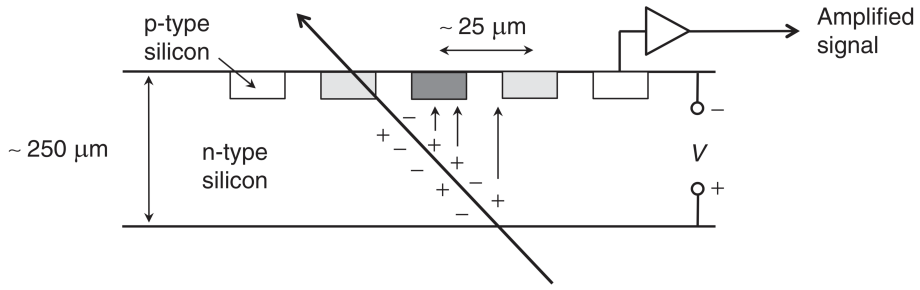


Figure 5: The production and collection of charge in a silicon tracking sensor [34].

The pn-junction can be made larger and the electric field across it stronger, to allow for better efficiency of charge collection and the ability to detect more energetic particles. This is typically done by applying a *reverse bias* across the pn-junction (i.e. a negative potential in the p-side and a positive potential in the n-side) to further separate electrons and holes from the depletion zone [38]. This can increase the width of the zone from a few microns to a few millimetres, however breakdown can occur at high potential difference. To readout from the pn-junction, heavily doped n and p material is used between the semiconductor and the metal leads. Provided the depletion zone is thick enough to stop the particles, detection is extremely efficient [37].

Broadly speaking, there are two types of silicon trackers: continuous and discrete [37]. The discussion shall be restricted to micro-strip detectors, which are a type of discrete tracker. The micro-strip detector works by using a highly resistive n-type silicon base and placing a series of p-type readout strips onto the surface [34]. A suitably heavily doped n-type electrode is placed on the other side of the base. The spatial resolution is then governed by the separation of the strips [13]. It should be noted that like the plastic scintillator, semiconductor detectors are also liable to radiation damage. High-energy particles can knock atoms out of the semiconductor lattice, to create point defects and disrupt the band structure by introducing trapping levels in the band gap [20].

4.3 The UCL Single-module Calorimeter

The University College London (UCL) Proton Beam Therapy group is currently developing a single-module calorimeter utilising scintillator technology from the SuperNEMO experiment, which is seeking evidence for neutrino-less double-β decay. Such an experiment requires highly accurate and sensitive apparatus in order to detect low-energy electrons against a strong background of γ-rays, fulfilling the requirements of fast, accurate data collection. The apparatus was also designed with cost-effectiveness and longevity in mind [39]. While these characteristics are attractive for medical applications, an element of luck was involved for the use of this scintillator for treatment plan verification measurements. The single-module calorimeter prototype consists of a 3 cm $\times$ 3 cm $\times$ 5 cm polystyrene plastic scintillator block (wrapped in Mylar foil to

increase light output) of density 1.06 g/cm^3, maximum emission wavelength of 425 nm (where the refractive index is 1.6), and a decay time of 2.5 ns. The scintillator is coupled via optical gel to a Hamamatsu R13089 photomultiplier tube (PMT), which is powered by a Caen NDT1470 high voltage power supply, and with a Teledyne LeCroy HDO6104 oscilloscope recording the output of the PMT. The scintillator and PMT are shown in Fig. 6.

Figure 6: The 3 cm × 3 cm × 5 cm polystyrene plastic scintillator block wrapped in Mylar foil and Hamamatsu R13089 PMT.

4.4 The PRaVDA Silicon-strip Tracker

This project is conducted in partnership with the Proton Radiotherapy, Verification and Dosimetry Applications (PRaVDA) consortium, who have developed solid-state silicon-based trackers capable of simultaneously detecting several protons in 2D at beam rates of 26 MHz to sub-mm precision [40]. Based on technology designed for use in the proposed High-Luminosity Large Hadron Collider, its intended use is the development of proton CT techniques to improve treatment planning in proton therapy[1]. In this experiment however, it is proposed that the tracker could be utilised in conjunction with a calorimeter for treatment plan verification measurements. Each tracker module consists of three layers of 150 μm thick *n-in-p* silicon strip detector, each with an active area of 93 × 96 mm^2 and a strip pitch of 90.8 μm. The layers are offset by 60 degrees to each other to form a *x-u-v* coordinate system, to allow unambiguous 2D measurements of position [41]. Experimental runs were conducted at the Birmingham University MC40 cyclotron, where a PRaVDA tracker module was available. The module is shown in Fig. 7.

[1]If protons were used instead of photons in mapping treatment areas for patients, many of the uncertainties associated with the differences in particle interactions could be overcome, giving more accurate treatment plans.

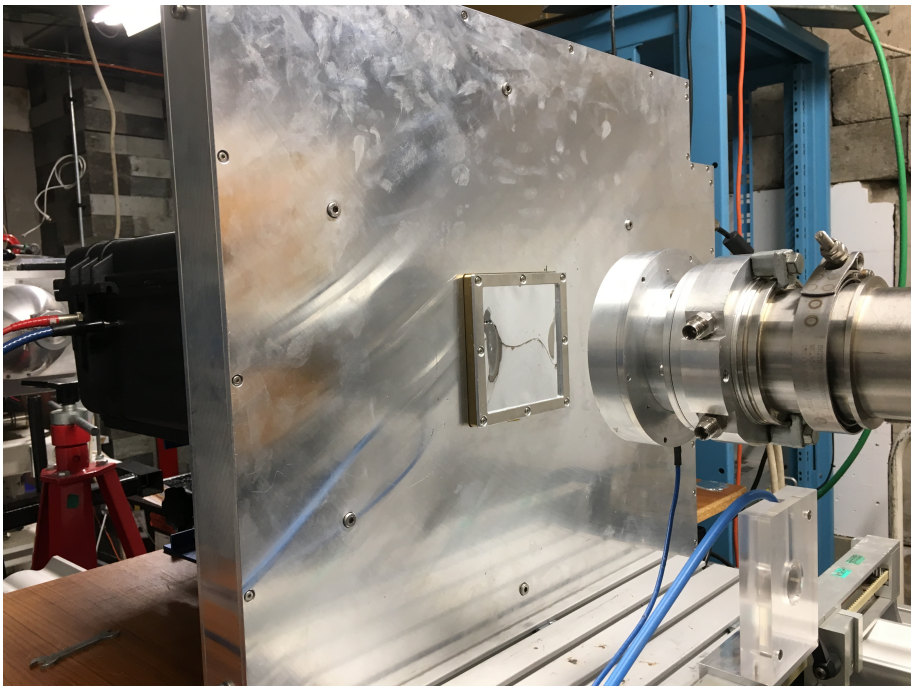


Figure 7: A single PRaVDA silicon-strip tracker module, placed in the beamline at the Birmingham University MC40 cyclotron. The single-module calorimeter housing is placed directly behind the tracker.

4.5 Project Goals

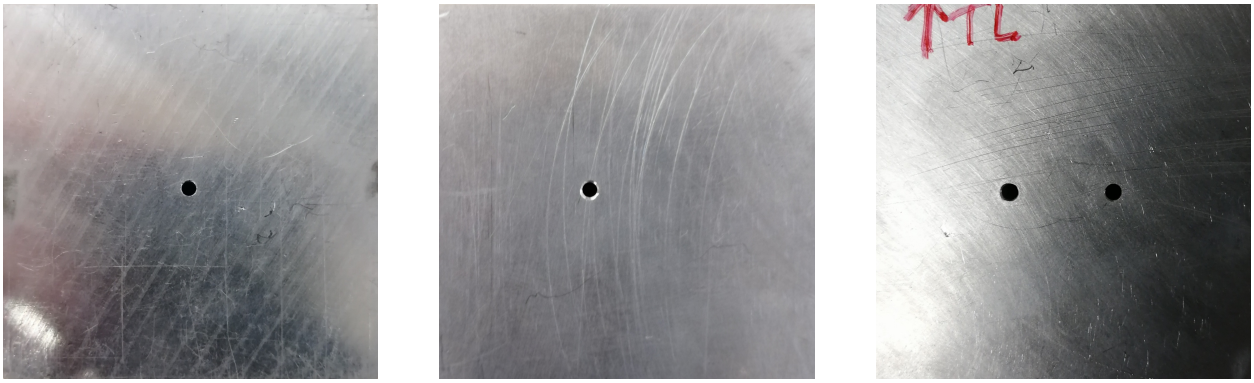
The goals of the project were:

1. To develop analysis tools to accurately analyse data from the calorimeter to calculate proton energies. C++ code utilising CERN's ROOT data analysis framework written previously for the group could successfully calculate proton energy, however required significant improvements in pile-up filtering and pulse location. Extensions to the code were required to allow analysis of tracker data from PRaVDA and match tracker events to calorimeter events. The planned work sought to repackage and improve this code to make it more efficient, more powerful, and easier to use.

2. To provide proof-of-principle of the use of a calorimeter and tracker simultaneously to reproduce dose deposition distributions by successfully matching calorimeter and tracker events. Experiments were held at Birmingham University using a test proton beam to fire protons at the calorimeter and tracker, placed in line of each other and the beam. Collected data was analysed in an effort to extract the energy spectrum of recorded protons, and to pair tracker and calorimeter events to reproduce dose deposition maps.

5 Birmingham Beam Test (25th-26th October 2018)

The aim of the experiment was to test the use of the single-module calorimeter alongside the PRaVDA silicon strip tracker with a 36 MeV proton beam. The intention was to perform measurements of proton energy and position through increasing thickness of absorber (PMMA), and with different types of collimators. Proof-of-principle for the simultaneous use of the two detectors would come from reproducing the 3D dose-deposition distributions of the different collimators, which are shown in Fig. 8, and by demonstrating the shift in the peak of energy spectra of protons through increasing thickness of PMMA absorber. Similar experiments had been conducted previously to provide proof-of-principle, however they ultimately failed due to the inability to sync the internal clocks of the tracker and LeCroy oscilloscope, where the latter was self-triggered at the time. This meant that position measurements could not be correlated with energy measurements. The experiment discussed here improved upon the previous attempt by having the tracker trigger the calorimeter and by performing data collection in short bursts.



(a) 2 mm centred collimator.

(b) 2 mm off-centre collimator, offset by 6 mm in the x-direction.

(c) 2 mm double collimator, with centres 12 mm apart in the x-direction.

Figure 8: The different collimator configurations used in the experiment. Each tested with 0-6 mm thick PMMA. The collimator shown in Fig. 8c was tested with 3.95 mm of PMMA in front of one hole.

The calorimeter was placed on a level stand downstream of the proton beam to the tracker (as shown in Fig. 9), with the windows of both aligned by careful measurement and markings placed on the calorimeter case. Experimental runs were conducted across two days with: a Caen high voltage supply providing -900V to the PMT, the LeCroy oscilloscope operating on a positive edge trigger set at a threshold of 70 mV, and a beam current on the order of 10 kHz ($\sim$160 pA). The oscilloscope had a time-base of 100 ns/div, a trigger delay of 80 ns and a vertical scale of 50 mV/div. These were set by monitoring the input for some test protons, such that the pulse was centred with sufficient tail visible. The tracker trigger was sent into channel 2 of the oscilloscope; a square wave of amplitude $\sim$400 mV and period $\sim$15 ns, which triggered the input of channel 1 (the output of the calorimeter) to be recorded for the length of the window (100 ns/div $\times$ 10 div = 1000 ns). This is referred to as an acquisition. The output of the PMT was split between the oscilloscope and a Caen DT5751 digitiser, the latter was used to take independent measurements of proton energy spectra for verification with the oscilloscope. The experimental set-up is summarised in Fig. 10.

Figure 9: The housing of the single-module calorimeter placed on a level stand downstream of the tracker, such that the windows of the tracker and calorimeter were aligned.

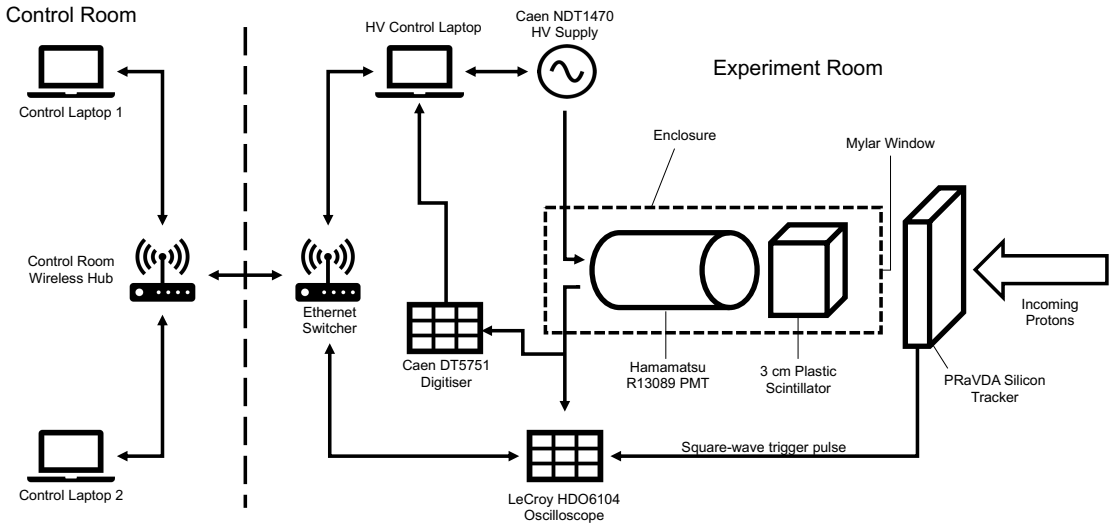


Figure 10: Experimental schematic: a proton first travels through the chosen collimator/absorber configuration and then enters the tracker, where a position measurement is made. When the tracker registers a hit, a trigger is sent to the oscilloscope to record the output of the PMT for a predetermined length of time, during which the proton should stop in the scintillator block, creating a flash of light that generates a current in the PMT.

Due to technical difficulties with the proton beam, limiting the time available for data acquisition, and recording of the incorrect channel on the oscilloscope, the run on the 25th failed to provide useable data. However, the remote desktop configuration (used to operate the oscilloscope and digitiser remotely in the control room), calorimeter, digitiser, oscilloscope and tracker were all tested to be functional. The oscilloscope was found to successfully receive a trigger from the tracker; measurements were made of the waveform trace of the trigger both directly from the tracker, and when split between the oscilloscope and the digitiser (where it was also attempted to trigger from the tracker). The traces revealed an expected $\sim$50% drop in amplitude when split, however it was found that the amplitude was too small to trigger the digitiser, which only accepts triggers on the order of $\sim$V. From the reconstructed hit distribution in 2D, the tracker was also found to be slightly off-centre. This was corrected before data acquisition the next day.

The run on the 26th proved more fruitful, where a 36 MeV beam was delivered for a longer period of time. Data collection was performed in 5 second bursts of protons, where the tracker fed triggers to the oscilloscope and the digitiser was set to self-trigger for its measurements of energy spectra. The oscilloscope recorded the output of the PMT with a maximum of 50,000 acquisitions per file, which was much greater than the number of triggers delivered in 5 seconds. After each run, the trace of the oscilloscope (the series of all the acquisitions recorded) was saved to disk. Each test configuration was repeated 3 times where the digitiser recorded one spectrum continuously for each 3 repeats of data collection with the oscilloscope. The test configurations were:

- A centred 2 mm collimator with no PMMA, used to confirm that the oscilloscope was being triggered appropriately by the tracker. The number of pulses was analysed after this run, and it was found that both the tracker and calorimeter recorded approximately the same numbers of events.
- A centred 2 mm collimator with increasing thickness of PMMA: 0.95 mm, 2.01 mm, 2.96 mm, 3.95 mm, 4.90 mm, 5.70 mm. Each thickness was repeated 3 times.
- A 2 mm collimator offset by 6 mm in the positive x-direction, with no PMMA and with 5.70 mm of PMMA.
- A 2 mm collimator pair, with centres apart by 12 mm, with no PMMA and with 5.70 mm of PMMA. An extra set without the PMMA was performed at a lower current rate.
- A 2 mm collimator pair, with centres apart by 12 mm in the x-direction, with one hole covered by 3.95 mm of PMMA. An extra set was performed at a lower current rate.

5.1 Development of Analysis Tools

5.1.1 Energy Spectra Reconstruction

The first stage of analysis was to reconstruct the proton energy spectra (histograms of calculated proton energies) from the acquisitions recorded by the LeCroy oscilloscope for all test configurations, and compare against the equivalent spectra recorded by the digitiser. The digitiser is held as a "gold-standard" for

spectrum reconstruction, however it lacks the ability to accurately resolve waveforms of individual protons, which is necessary in order to correlate individual proton measurements across detectors. While the digitiser automatically calculates energy spectra, a method must be devised to integrate over acquisitions recorded by the oscilloscope to recover proton energies. Comparison of energy spectra then allows for verification of the chosen method of integration. An example of an acquisition is shown in Fig. 11. A data set consists of thousands of such acquisitions, most are considered to be "good", where only a single proton is recorded, however some are considered to be "bad" (typically 10%), where there is more than one proton (pile-up), or no proton at all (false-trigger).

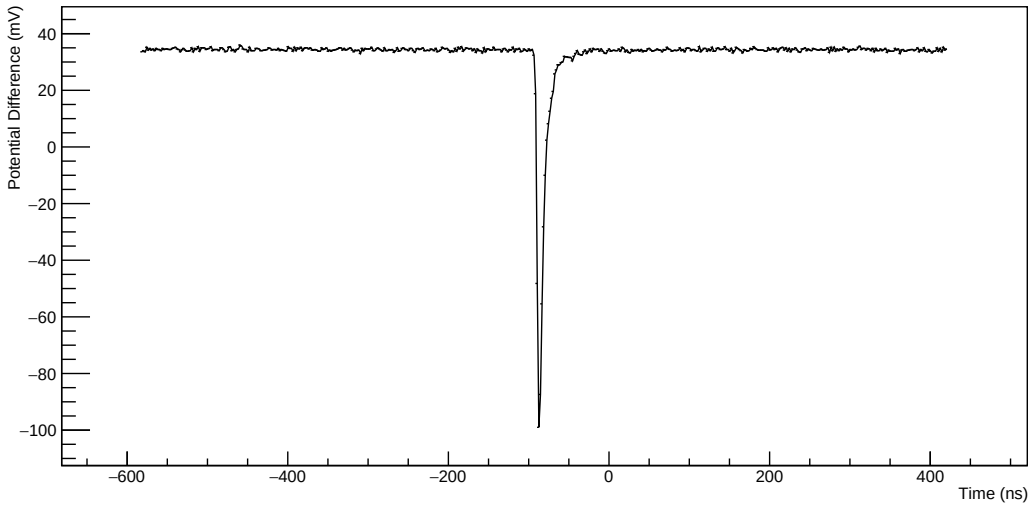


Figure 11: An example of a good acquisition recorded by the LeCroy oscilloscope, i.e. one with only a single proton. 0 ns on the time axis represents when the trigger arrived from the tracker. The pulse takes negative values due to the negative potential powering the PMT. The acquisition must be baseline subtracted (to remove dark current noise), flipped in the y-axis to give positive values and then integrated over to find the energy.

Code was written to first import and then plot digitiser data in histograms, for comparison with LeCroy spectra. Development of the analysis tools used the centred collimator with 2.01 mm PMMA configuration as a test data sample, as it appeared to be the cleanest data set from viewing the digitiser spectra. The spectrum produced by the digitiser for this configuration is shown in Fig. 12.

Acquisitions recorded by the oscilloscope require three stages of processing:

1. Baseline sigma testing: This is where the signal variation of a distinct region of the acquisition is tested to determine whether it lies within a pre-defined standard deviation (called the maximum baseline sigma). This is employed as a rudimentary way to filter acquisitions with pile-up, and is explained further below.

2. Baseline subtraction: This is where the dark noise current is subtracted (shown as the variation around 35 mV in Fig. 11) and then the signal is flipped to give positive potential difference values.

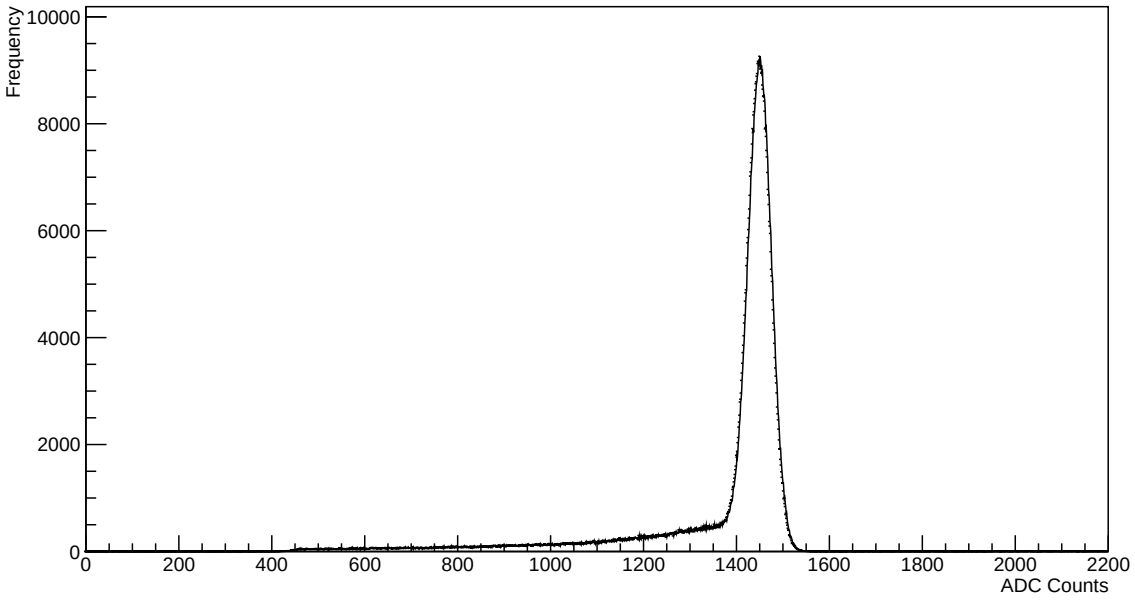


Figure 12: The reconstructed spectrum from the Caen digitiser for the centred collimator with 2.01 mm PMMA configuration. The digitiser records data into bins of size 1 mVns. Analysis of the LeCroy data seeks to reproduce such spectra.

3. Integration: Where a region of the acquisition is chosen and the signal integrated over, to give a value representative of the proton energy.

Previously written code for the analysis of LeCroy data successfully calculates proton energy by integrating using a composite rule Newton-Cotes formula [42, 43] over a pre-determined window within an acquisition (i.e. where the pulse should be). However, this operated on the assumption of a fixed pulse position within an acquisition, due to the oscilloscope self-triggering at the time, such that the pulse would typically start just before 0 ns. This was no longer the case. The tracker triggered the oscilloscope to record an acquisition, so the pulse could be anywhere within an acquisition (due to timing jitter between the arrival of a proton and the output of the trigger pulse from the tracker) or not at all (if a false trigger is sent from the tracker). Initially, the user-definable fixed-position parameters were adjusted to give the best spectrum possible, which were found to be: an integration length of 150 ns, a horizontal offset of -120 ns (i.e. where integration starts relative to 0 ns), and a maximum baseline sigma of 20. An example of the resulting spectrum, for one repeat of the centred collimator with 2.01 mm PMMA configuration is shown in Fig. 13.

Comparison of Fig. 12 and Fig. 13 showed that the peak was recovered with approximately the correct height. It was not essential for the peak to have the same absolute energy across the digitiser and the oscilloscope; instead it was more important to recover the finer features of the energy spectrum, such as the correct peak shape. The energy scale of the LeCroy could be calibrated to match the Caen, if needed. The most apparent problem in Fig. 13 was the non-negligible number of events that appear in a Gaussian-like shape centred around 0 mVns, indicative of acquisitions with no pulses. To rectify this, and to improve pile-up filtering, the baseline testing method was overhauled.

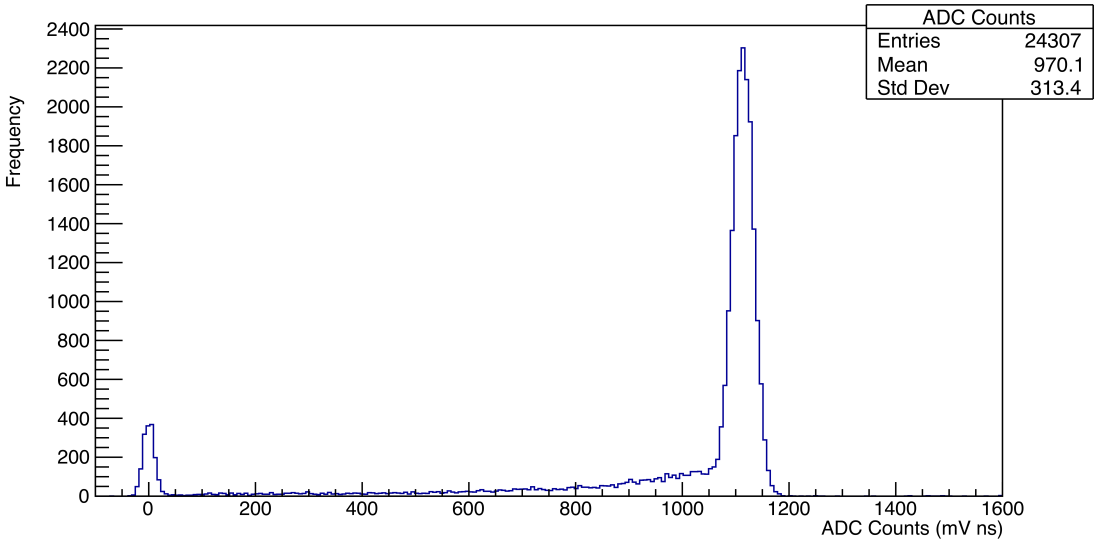


Figure 13: The reconstructed proton energy spectrum from the LeCroy oscilloscope for one repeat of the centre collimator pair with 2 mm PMMA configuration using a fixed integration window. Data placed into 100 bins.

First, a few words to explain baseline testing. This process was inherited from the previous code and its purpose was to determine an average background noise level, which could then be subtracted in the baseline subtraction process, and to test (semi-rigorously) whether an acquisition had any pile-up. The original method tested 9/10 of the region before 0 ns (i.e. -600 ns to -60 ns in Fig. 11, referred to as the pre-trigger region), calculating an average potential difference and testing the potential difference variation in this region to determine if a pulse was present. If the standard deviation of the signal was too high in the pre-trigger region, an acquisition would be labelled as "bad" and would be filtered from the energy spectrum. However, this method only works if one can assume that the pulse will always be close to 0 ns (hence the reason 9/10 of the region is used, to avoid the pulse skewing the test), and it does nothing to tackle empty acquisitions or filter pile-up in the positive time region.

Initially, it was chosen to continue to test the pre-trigger region, assuming that a pulse would still be somewhat near 0 ns and that most acquisitions would only contain one pulse. Routines were added such that if the baseline test was passed then another test would take place that searched for a negative potential difference value in the acquisition, indicative of a pulse present in the acquisition. If both tests were passed, then an acquisition was labelled as "good". Additionally, if the initial test was failed, then 9/10 of the points *after* 0 ns (i.e. 40 ns to 400 ns in Fig. 11, referred to as the post-trigger region) was tested, in case the pulse happened to be in the pre-trigger region. If this second test was passed, and a negative potential difference was found, then an acquisition could also be labelled as "good". Acquisitions that failed both baseline tests or failed to provide a negative potential difference value were labelled as "bad" and filtered from the energy spectrum. This baseline testing method is more robust than the previous version as a single pulse can be in any part of an acquisition and still pass the baseline test. However, the issue of pulses close together still presides and if the first baseline test is passed, then any extra pile-up towards the end of the acquisition will

not be identified. In practice however, these two issues did not cause any major problems, as it turned out that the majority of acquisitions only had one pulse.

By locating the largest negative value recorded in the acquisition, i.e. the peak of the pulse, the location of the 150 ns integration window was dynamically allocated instead of relying on a fixed position. This removed the need for the horizontal offset parameter (meaning less guesswork for the user) and aided pile-up filtering by focusing integration on the most prominent pulse in an acquisition. However, it was found that constraining the maximum baseline sigma any more than the (generous) value of 20 sigma would ruin the spectrum in Fig. 13 entirely, such that no peaks were observed. Investigation of some raw waveforms led to the realisation that most pulses arrive at roughly -120 ns, which meant that choosing a pre-trigger region of 9/10 of the points before 0 ns in the baseline test was including some of the actual pulse. Shortening the testing region to 8/10 improved the spectra considerably: the maximum baseline sigma could be constrained to 5 sigma (meaning stricter pile-up filtering) and the location of the peaks shifted closer to those in the digitiser data without any scaling. The improved version of Fig. 13, combining the 3 repeats of the configuration and with all the above changes is shown in Fig. 14, demonstrating a much closer resemblance to Fig. 12.

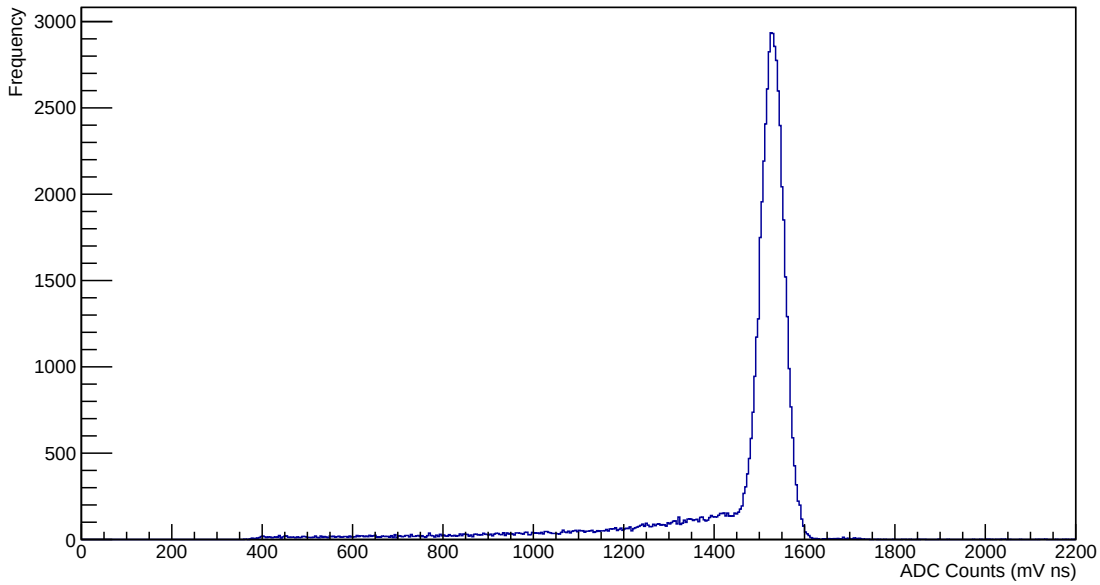


Figure 14: The reconstructed spectrum from the LeCroy oscilloscope for all 3 repeats of the centred collimator with 2.01 mm PMMA configuration, now using a dynamic integration window (150 ns in length), a shorter baseline test region, and checking for empty acquisitions. Data placed into 550 bins.

For a complete comparison of energy spectra, it was necessary to plot digitiser and oscilloscope spectra together on a single plot, scaling as appropriate. It was elected to scale the higher-resolution Caen digitiser spectra to match that of the LeCroy oscilloscope. The energy scale of the oscilloscope required some small calibration to match that of the digitiser and since the digitiser was left to self-trigger and ran continuously for each test configuration, it recorded many more events than the oscilloscope, so the frequency scale also had to be normalised. By comparing the positions of the top of the peak across the digitiser and final LeCroy spectra, approximate x- (energy) and y- (frequency) scale factors were first found, which were found to give

reasonable matches on their own. To find a more accurate y-scale factor however, the number of events within ± 160 mVns of the peak in both the oscilloscope and digitiser spectra were counted. Dividing the number of events in the Caen peak by the number of LeCroy events (multiplied by a factor of 4) gave an accurate y-scale factor. As the results will show later however, this method was not reliable for data sets with poor peaks. The factor of 4 is also discussed in the results section.

To find a more accurate x-scale factor, it was elected to minimise a χ^2 between the digitiser and oscilloscope histograms [44], using the approximate x-scale as a prior. The technical difficulty for this arose from the fact that the digitiser data is much more precise than the oscilloscope, having 4 times as many bins for an axis range of 0-2200 mVns and that in ROOT, a χ^2 can only be tested between identically binned histograms [45]. This is the reason that χ^2 minimisation could not be used for the y-scale, as one would have to re-bin the digitiser data to match that of the LeCroy, rendering any y-scale found unusable for the full-resolution spectrum. Given that only a small change was needed to the prior x-scale estimate, 30 candidate x-scale factors were constructed $\pm 1.5\%$ of the prior (i.e. in steps of 0.05%). For each candidate, a separate histogram was constructed of the digitiser data, but in the same number of bins as the oscilloscope (550) and scaled in the x-axis with the candidate scale factor. A χ^2 test was then performed between each candidate histogram and the original LeCroy histogram: the scale factor that returned the smallest χ^2 would then be chosen as the x-scale factor to apply on the full-resolution digitiser data in the scaling process. The results of such a process is shown in Fig. 15.

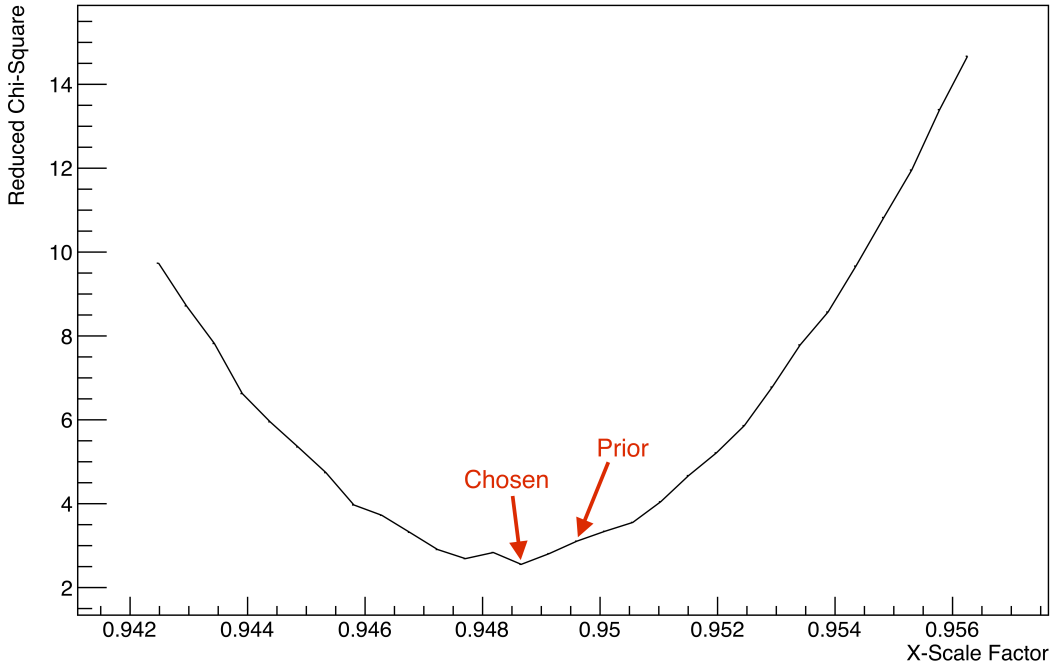


Figure 15: Results of the χ^2 minimisation to find an x-scale factor for the centred collimator with 2 mm PMMA data set. In this case, the prior is close to the final chosen scale factor.

This final match for the centred collimator with 2 mm configuration is shown in Fig. 16, where an excellent match is observed.

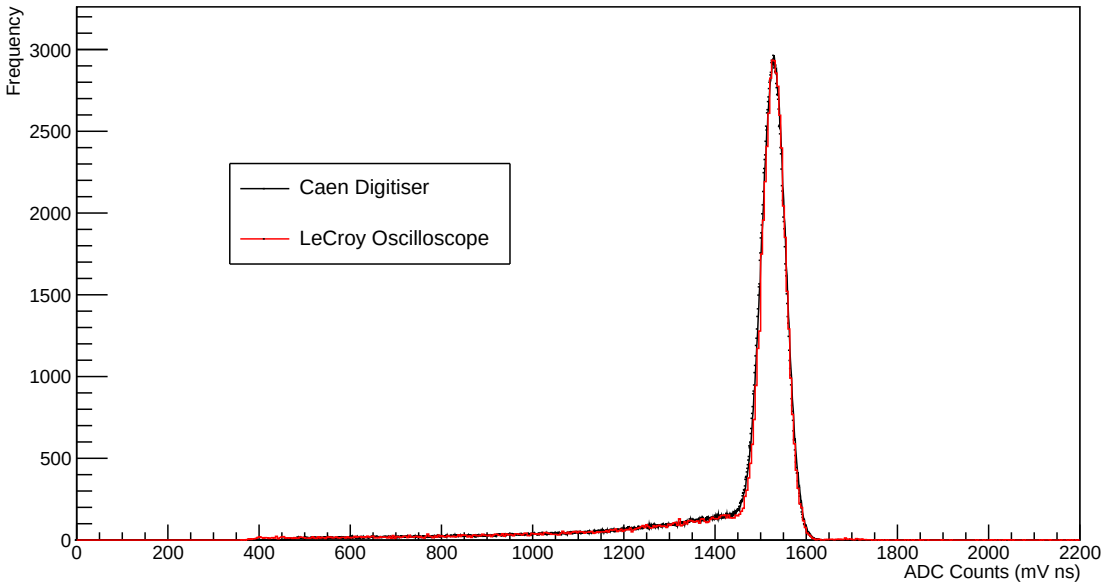


Figure 16: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 2 mm PMMA configuration, for all 3 repeats recorded by the LeCroy. Caen digitiser data scaled to match the LeCroy. x-scale factor found by χ^2 minimisation and y-scale factor found by comparing the number of events found in the peak. The reduced χ^2 was 2.547. The LeCroy recorded over 60,000 events whereas the Caen recorded over 740,000.

5.1.2 Tracker-Calorimeter Event Matching

With confidence in the proton energy calculation process, investigations were made into correlating tracker measurements to calorimeter measurements. Since the energy of a proton defines well its range in matter, the energy scale of the LeCroy was calibrated to a scale of proton range in water to provide the third dimension to 2D tracker measurements. Simulations in GEANT4 were conducted for the range in water of low-energy protons, between 4-40 MeV. The results of the simulation are shown in Fig. 17 and as expected, the range in water follows a power law with proton energy [23]. By setting the peak energy of a configuration with no PMMA absorber to correspond to a beam energy of 36 MeV, energies calculated from acquisitions were converted to a range in water.

Routines were written to plot tracker data in 2D histograms. The tracker plot of one of the repeats of the centred collimator with 2.01 mm PMMA configuration is shown in Fig. 18. The unit consists of 3 trackers, which can record up to 5 protons each, each with 3 position coordinates. As this experiment relies on single proton counting, a tracker event is only considered to be "good" if only one proton is recorded.

Each 2D position measurement made in the tracker had a timestamp, as did each acquisition recorded in the oscilloscope, where good acquisitions now had an associated energy value. It was noted that the tracker is capable of much faster data collection than the LeCroy scope, which is limited primarily by the length of an acquisition (i.e. it cannot record data faster than the length of an acquisition, which was 1000 ns, necessary to record a pulse in its entirety). This resulted in the tracker recording protons at finer intervals than the

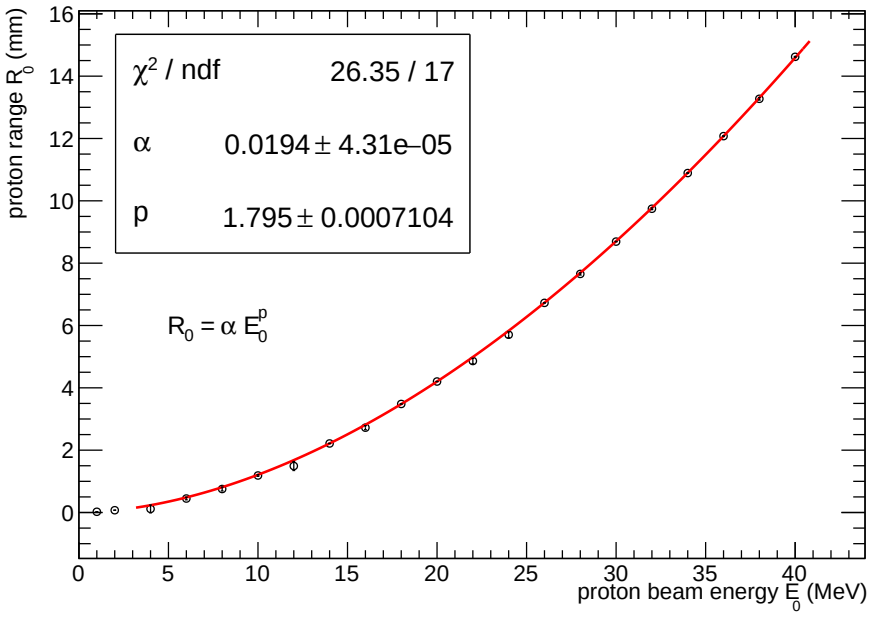


Figure 17: Results of a GEANT4 simulation of the range of protons for beam energies between 4-40 MeV. The values 1 MeV and 2 MeV were excluded due to the lack of error bars. The scale between MeV to mVns was found by setting the peak energy of the spectrum of a configuration with no PMMA absorber to 36 MeV. Plot provided by Laurent Kelleter, UCL.

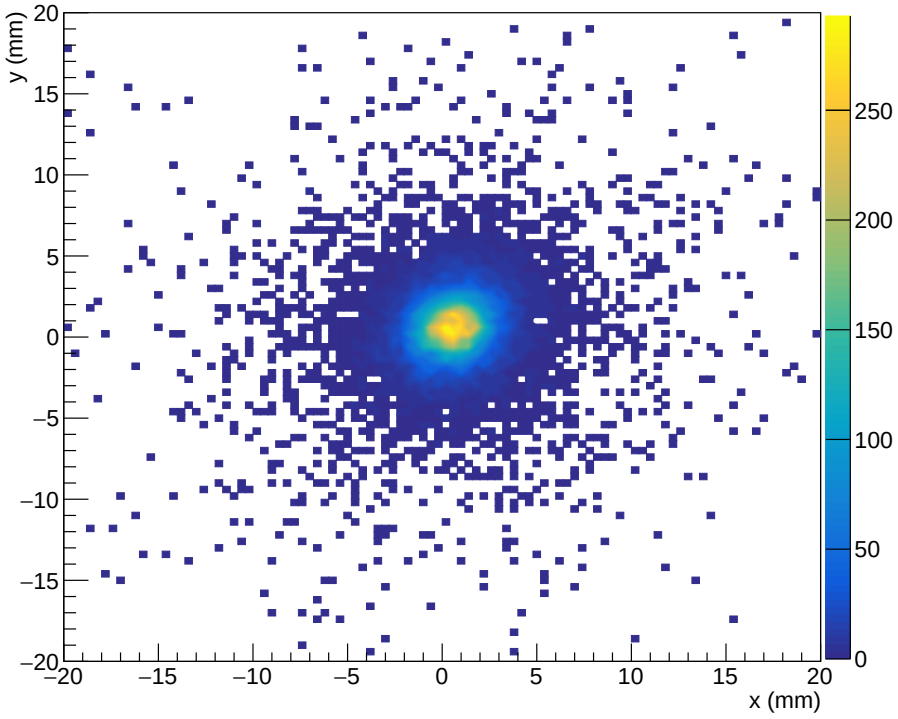


Figure 18: A 2D histogram of the x-y distribution of hits recorded in the tracker for one repeat of the centred collimator with 2.01 mm PMMA configuration. Data placed into 100×100 bins.

LeCroy, thus recording more events overall. This issue was dealt with by attempting to map each LeCroy event onto the first tracker event that satisfied the matching criterion. The initial matching criterion was to correlate events with timestamps within 500 ns of each other (smaller than the resolution of the LeCroy to avoid cross-matching), but this resulted in very few matchings. It was then realised that the clocks of the tracker and LeCroy drift further apart as time progresses, likely due to an inaccuracy in the clock of the tracker (which is believed to operate at exactly 26 MHz). By plotting the time difference between the closest tracker and LeCroy events, a straight-line equation was found to correctly increase the matching tolerance as time progresses. This is shown in Fig. 19.

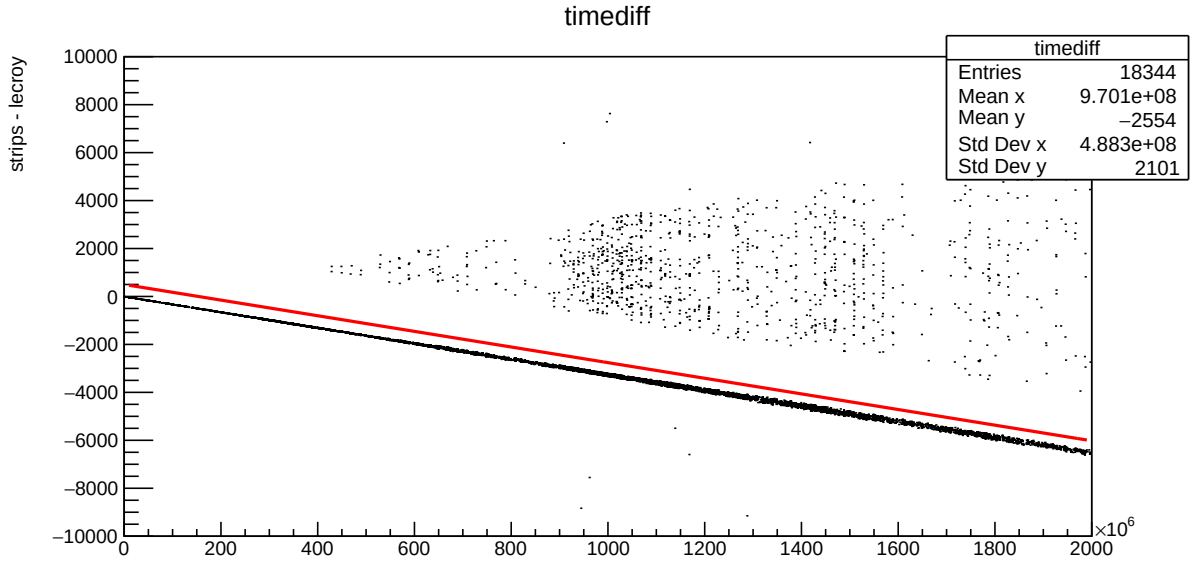


Figure 19: A plot of the difference between consecutive timestamps recorded in the tracker and LeCroy oscilloscope for one repeat of the half-covered collimator pair configuration. Points away from the line correspond to false triggers from the tracker. The red line corresponds to the equation $y = 500-(3.42\times10^{-6})x$ and increasing the tolerance window according to this gives acceptable matching of events. Plot provided by Tony Price, Birmingham University.

Re-matching events with an increasing tolerance window produced an acceptable number of matchings. Roughly 20% of the events are discarded when both LeCroy and tracker data sets are filtered to contain only "good" events, with the remaining events then correlated. To best illustrate the 4D information of x, y, range and frequency, the matched events were put into a 3D histogram of x and y (i.e. a 3D version of Fig. 18), but with the heights of the bars modified to be the average range of protons found in each $x - y$ bin, instead of the number of events. Then, the 2D number density plot (i.e. Fig. 18) was superimposed, to add the 4th dimension. The result of such a plot for the centred collimator with 2.01 mm PMMA configuration is shown in Fig. 20, which shows the expected deposition.

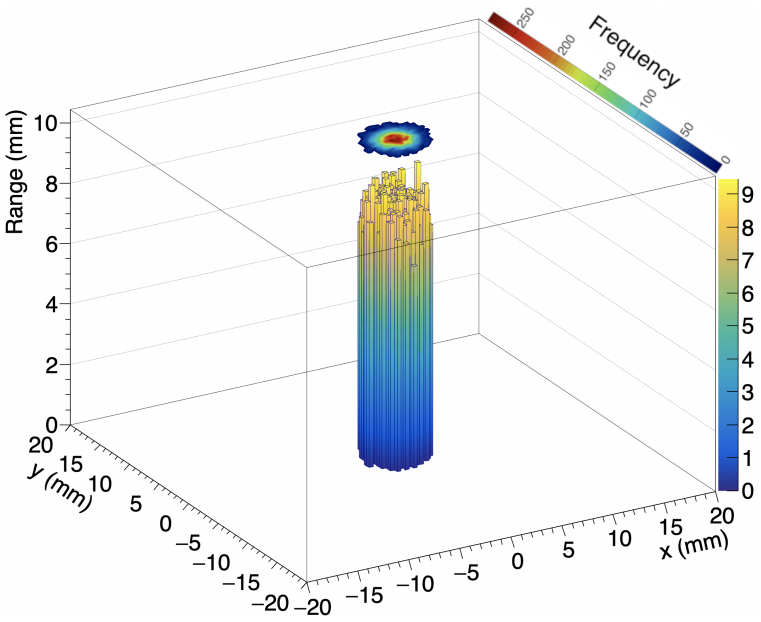


Figure 20: Reconstructed deposition in 3D for one repeat of the centred collimator with 2.01 mm PMMA configuration. The range is calculated from energy measurements by applying a power law found from a GEANT4 simulation of proton range in water for energies between 4-40 MeV. The superimposed 2D plot illustrates the XY intensity profile for the spot. Data placed into 100 $\times$ 100 bins and only bins with more than 20 protons are shown. Over 21,000 proton events were matched.

5.2 Results

With the analysis tools fully developed, the energy spectra and 3D reconstructions for all configurations tested is now presented. Recall that is expected that the range of protons will decrease with increasing thickness of PMMA absorber and that physical features of a given configuration are reproduced, such as a shift in proton position when the off-set collimator was used. The performance of the LeCroy against the Caen in reconstruction of energy spectra is evaluated by the quality of spectrum matches. All spectra produced in the same way as Fig. 16 and all 3D plots produced in the same way as Fig. 20.

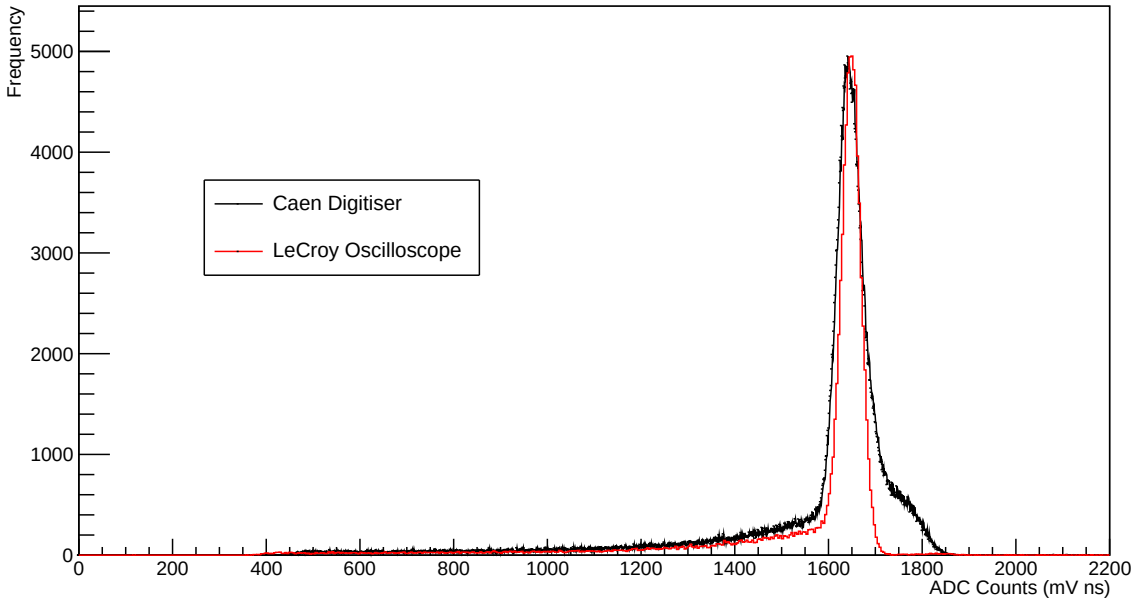


Figure 21: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with no PMMA configuration. The approximate y-scale factor was used due to the noisy peak caused by current fluctuations in proton beam. The reduced χ^2 for the x-scale factor was 37.012. The LeCroy recorded over 86,000 events whereas the Caen recorded over 254,000.

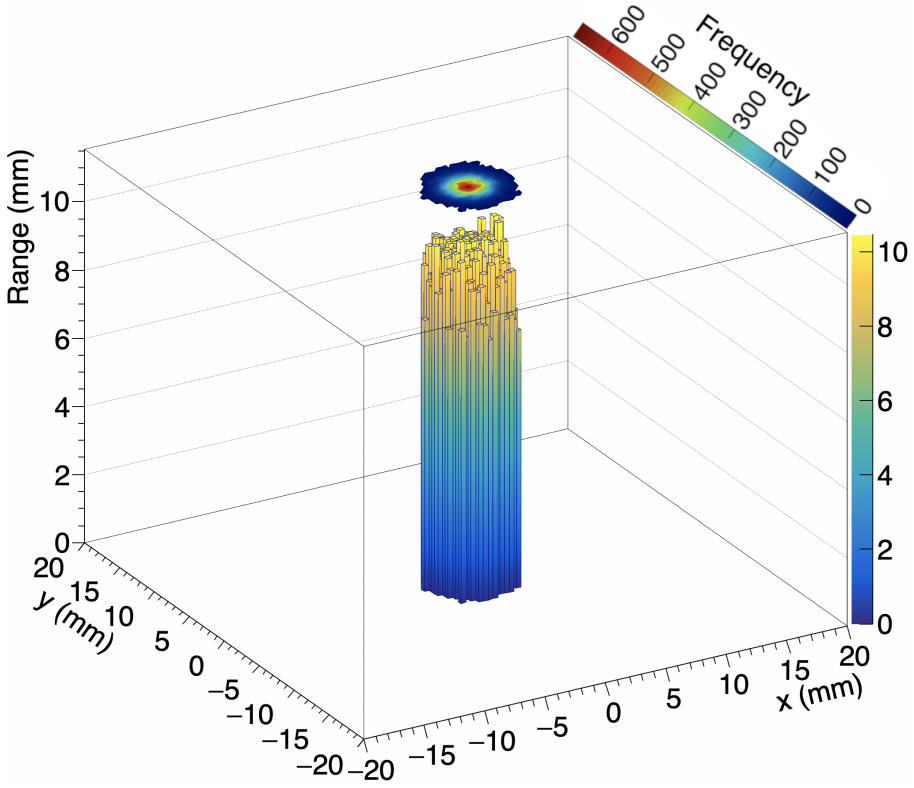


Figure 22: Reconstructed deposition in 3D for the centred collimator with no PMMA configuration. Over 39,000 proton events were matched.

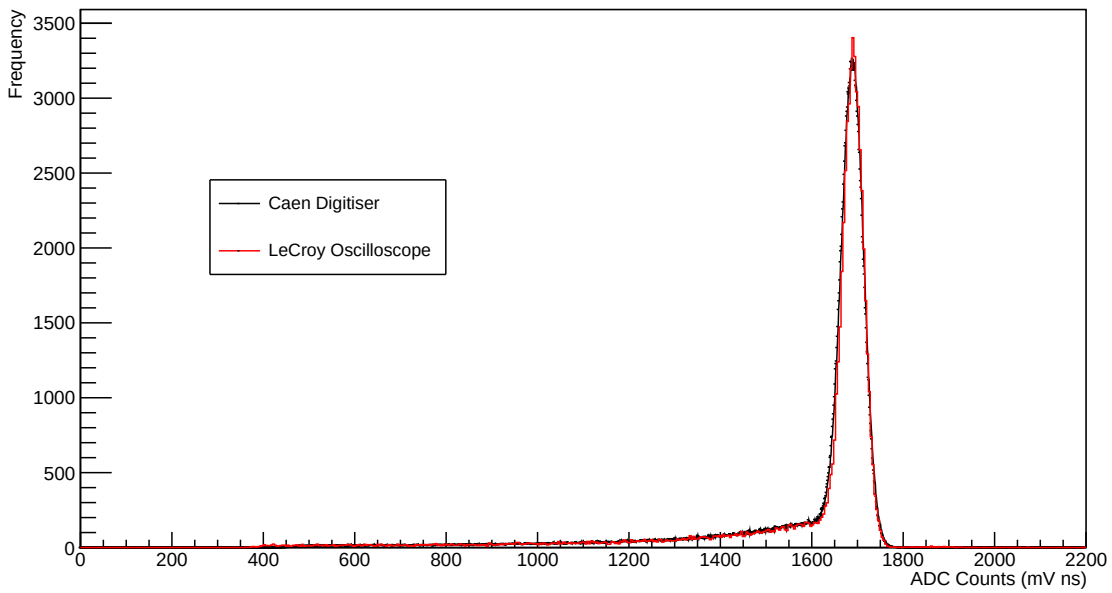


Figure 23: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 0.95 mm PMMA configuration. The reduced χ^2 for the x-scale factor was 2.7507. The LeCroy recorded over 62,000 events whereas the Caen recorded over 838,000.

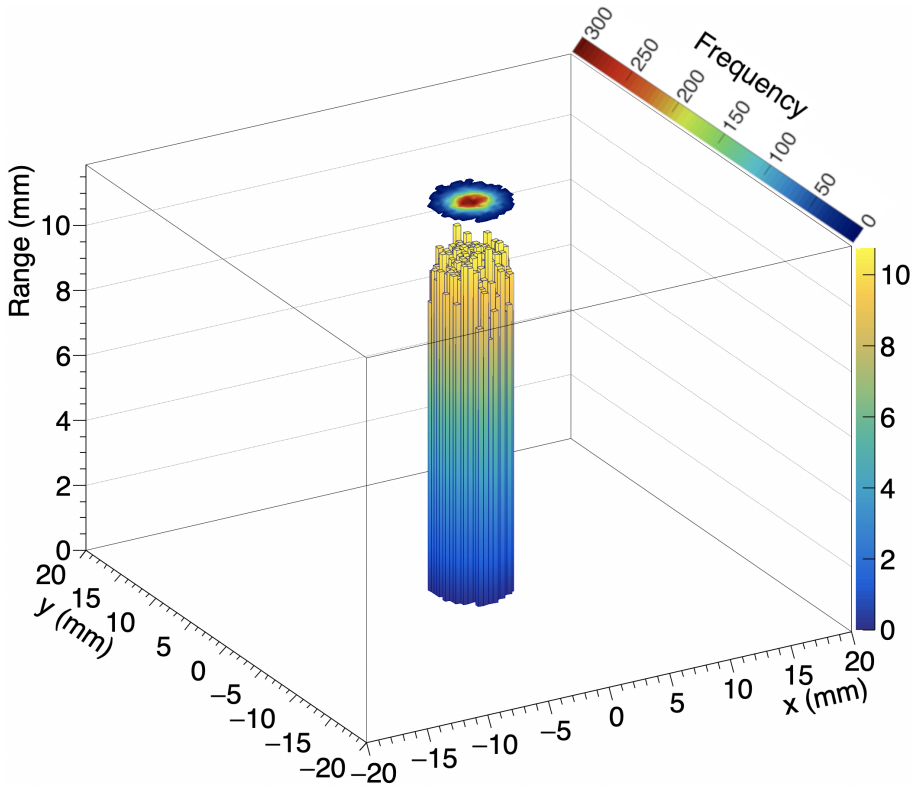


Figure 24: Reconstructed deposition in 3D for the centred collimator with 0.95 mm PMMA configuration. Over 21,000 proton events were matched.

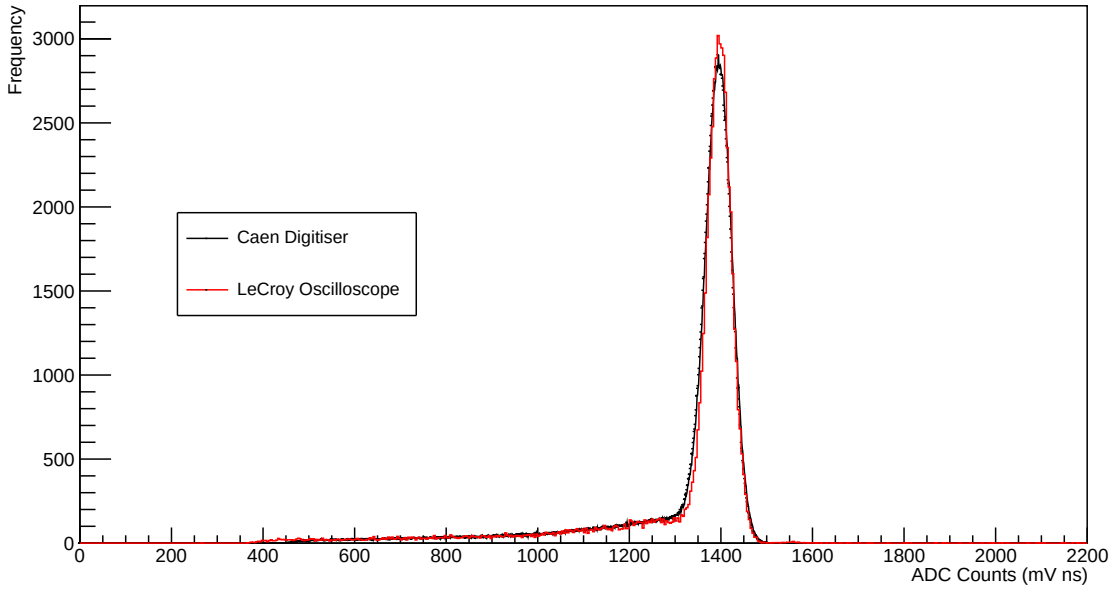


Figure 25: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 2.96 mm PMMA configuration. The reduced χ^2 for the x-scale factor was 4.0214. The LeCroy recorded over 62,000 events whereas the Caen recorded over 1,035,000.

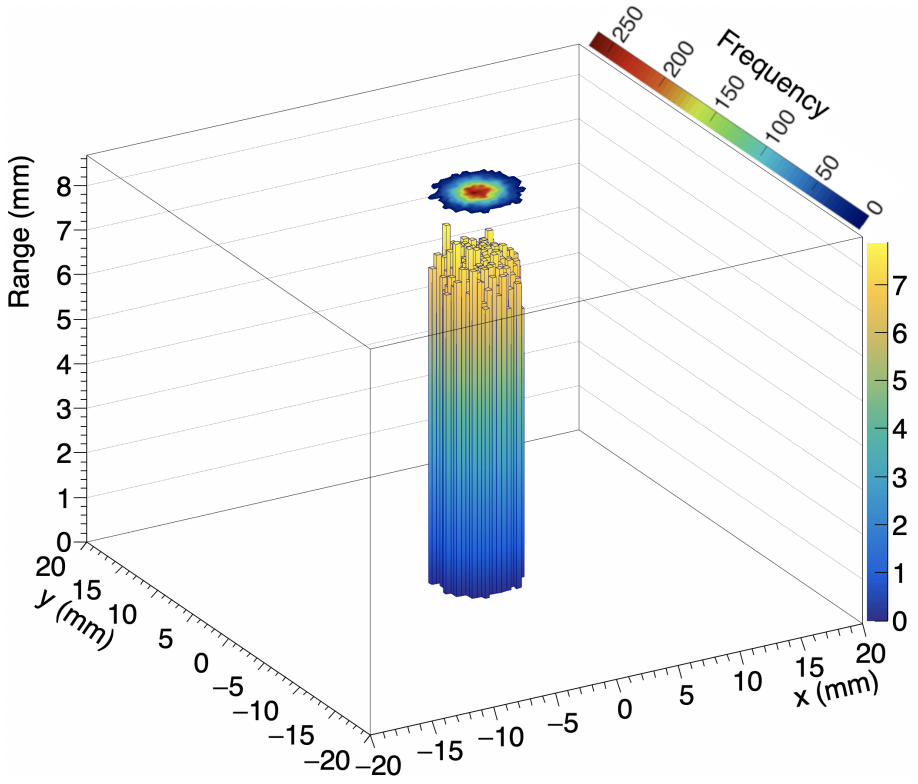


Figure 26: Reconstructed deposition in 3D for the centred collimator with 2.96 mm PMMA configuration. Over 20,000 proton events were matched.

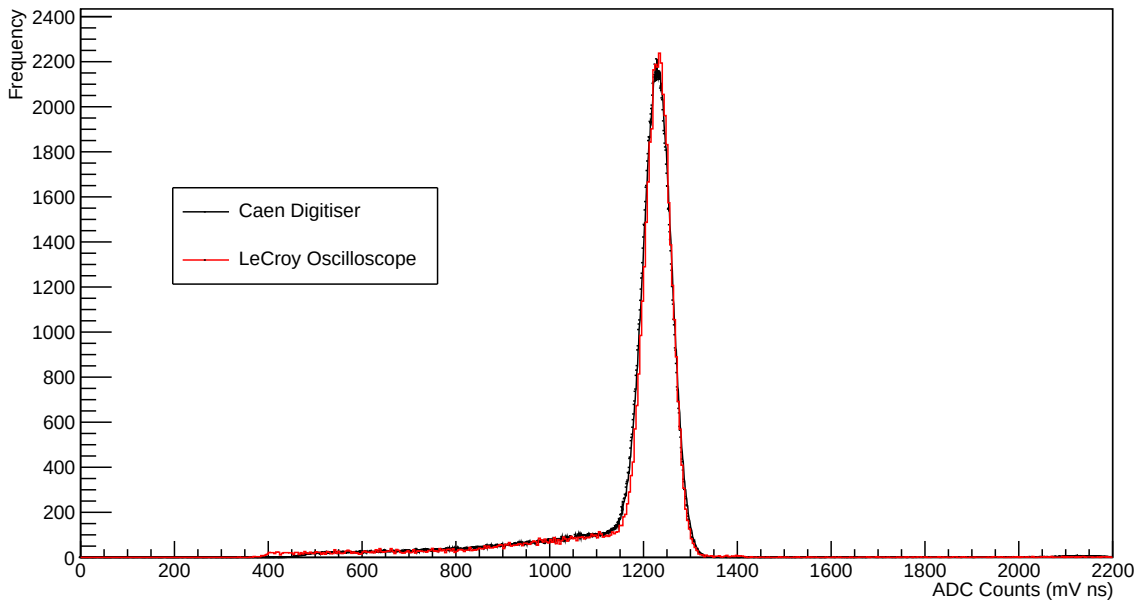


Figure 27: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 3.95 mm PMMA configuration. The reduced χ^2 for the x-scale factor was 2.8666. The LeCroy recorded over 48,000 events whereas the Caen recorded over 992,000.

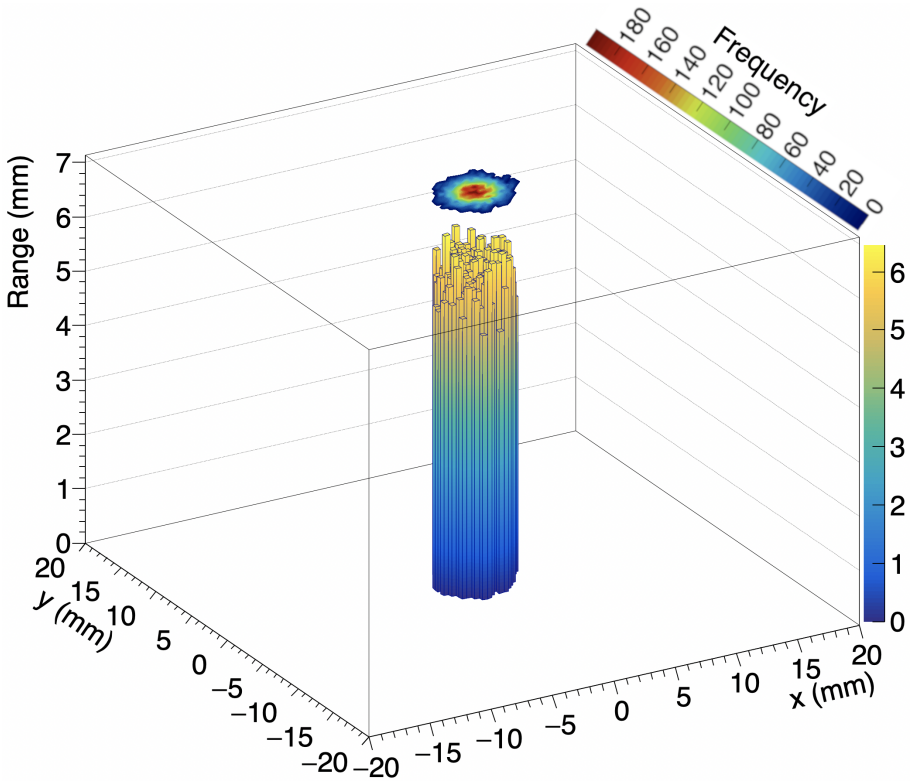


Figure 28: Reconstructed deposition in 3D for the centred collimator with 3.95 mm PMMA configuration. Over 16,000 proton events were matched.

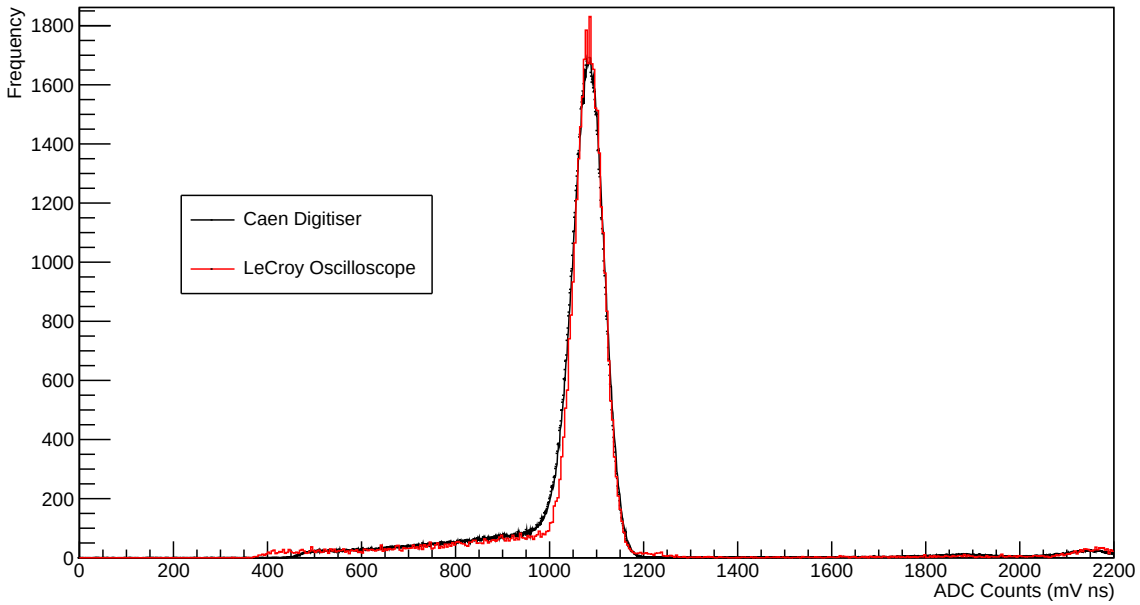


Figure 29: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 4.90 mm PMMA configuration. The reduced χ^2 for the x-scale factor was 3.2383. The LeCroy recorded over 41,000 events whereas the Caen recorded over 909,000.

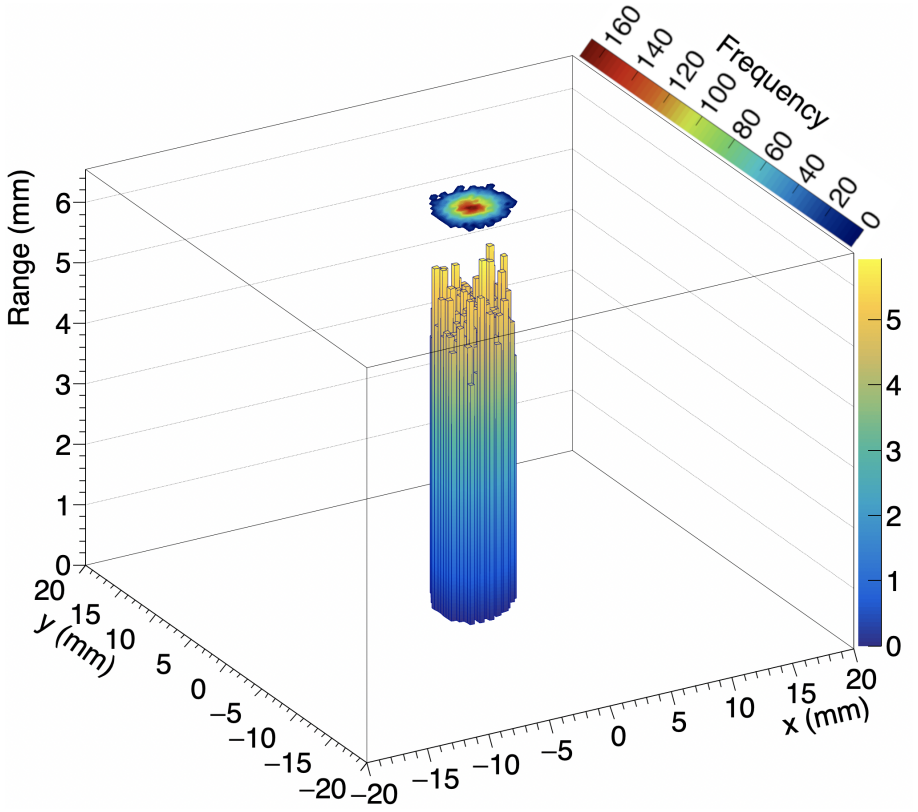


Figure 30: Reconstructed deposition in 3D for the centred collimator with 4.90 mm PMMA configuration. Over 14,000 proton events were matched.

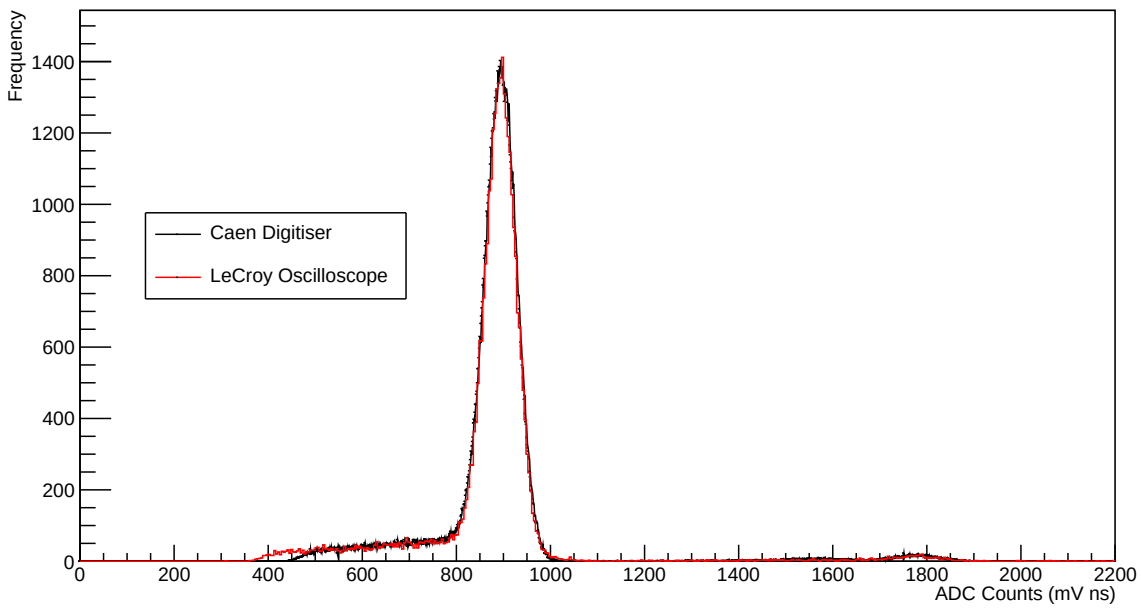


Figure 31: Scaled plot of the reconstructed Caen and LeCroy spectrum for the centred collimator with 5.70 mm PMMA configuration. The reduced χ^2 for the x-scale factor was 2.3773. The LeCroy recorded over 33,000 events whereas the Caen recorded over 391,000.

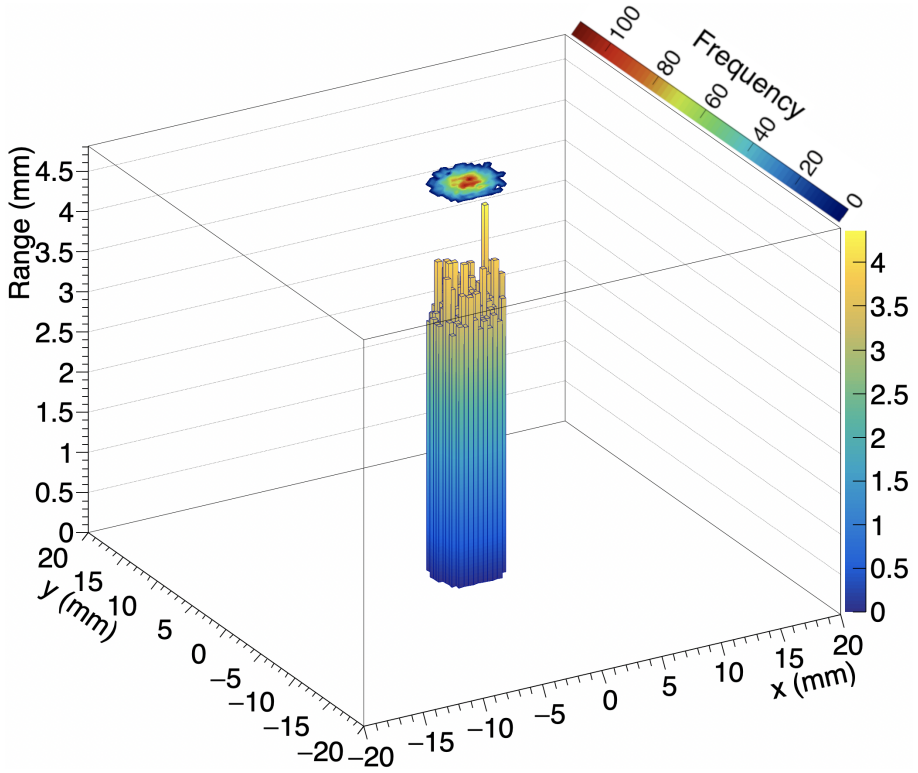


Figure 32: Reconstructed deposition in 3D for the centred collimator with 5.70 mm PMMA configuration. Over 9,000 proton events were matched.

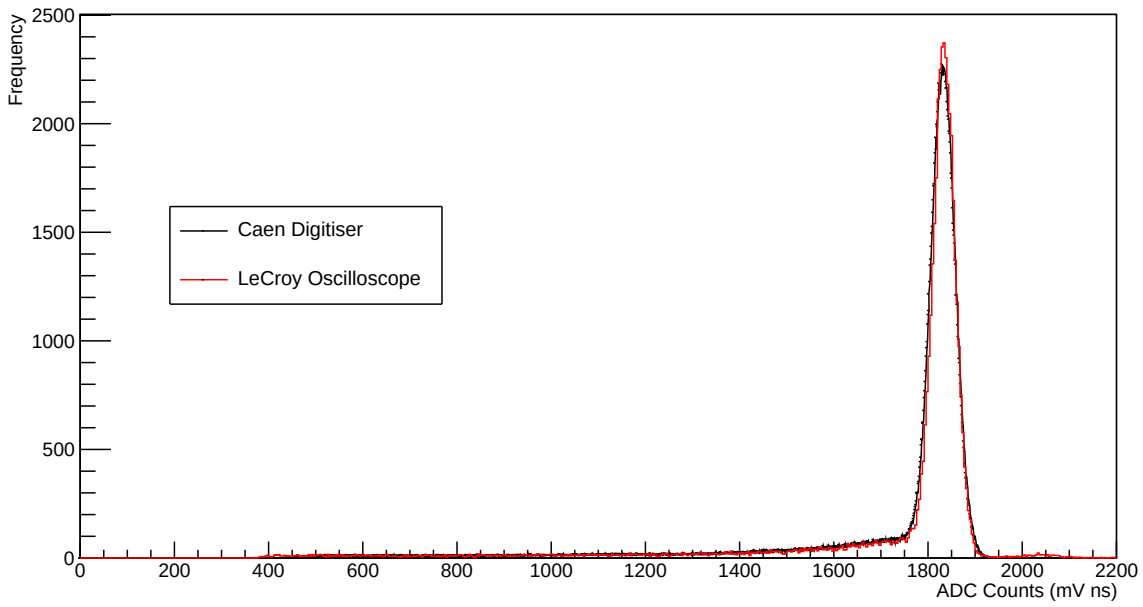


Figure 33: Scaled plot of the reconstructed Caen and LeCroy spectrum for the off-centre collimator with no PMMA configuration. The reduced χ^2 for the x-scale factor was 2.7235. The LeCroy recorded over 44,000 events whereas the Caen recorded over 577,000.

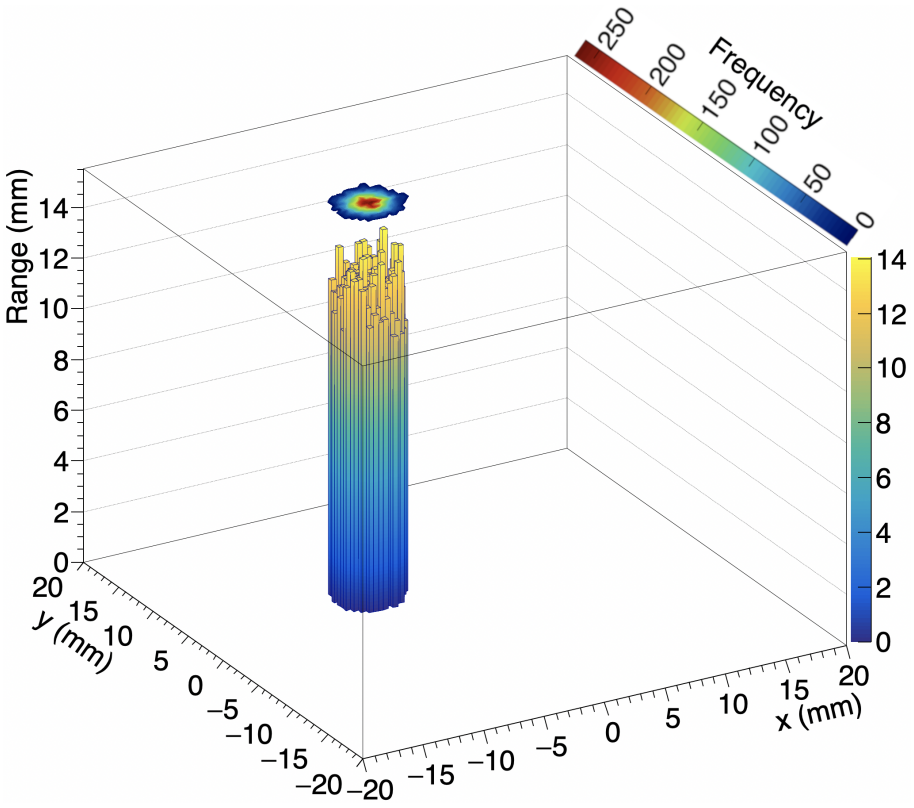


Figure 34: Reconstructed deposition in 3D for the off-centre collimator with no PMMA configuration. Over 15,000 proton events were matched.

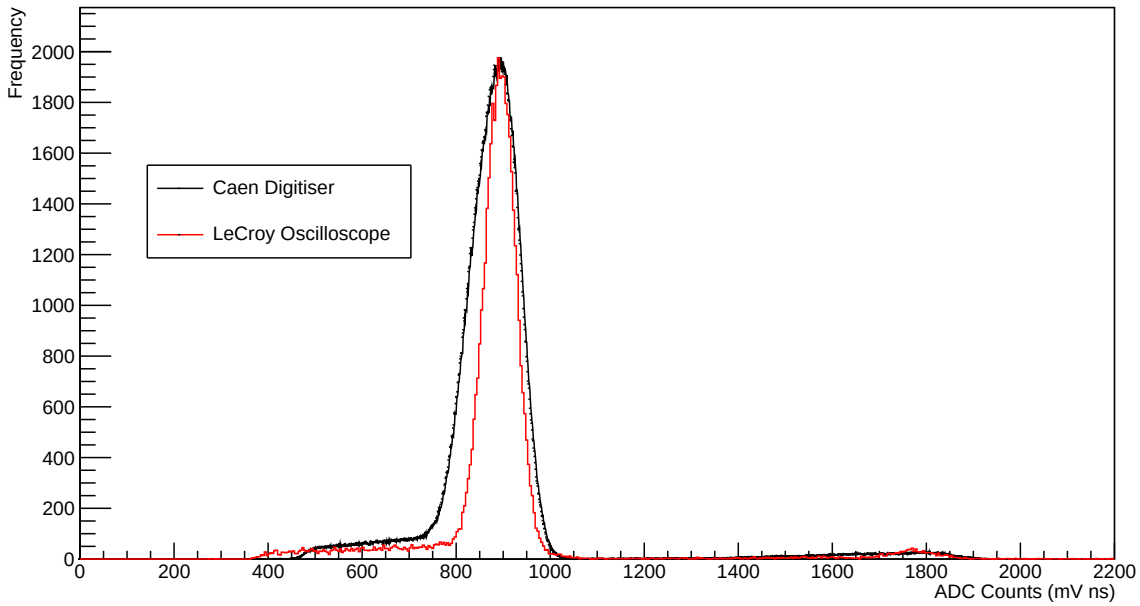


Figure 35: Scaled plot of the reconstructed Caen and LeCroy spectrum for the off-centre collimator with 5.70 mm PMMA configuration. The approximate y-scale factor was used due to the noisy peak caused by current fluctuations in proton beam. The reduced χ^2 for the x-scale factor was 14.800. The LeCroy recorded over 47,000 events whereas the Caen recorded over 1,350,000.

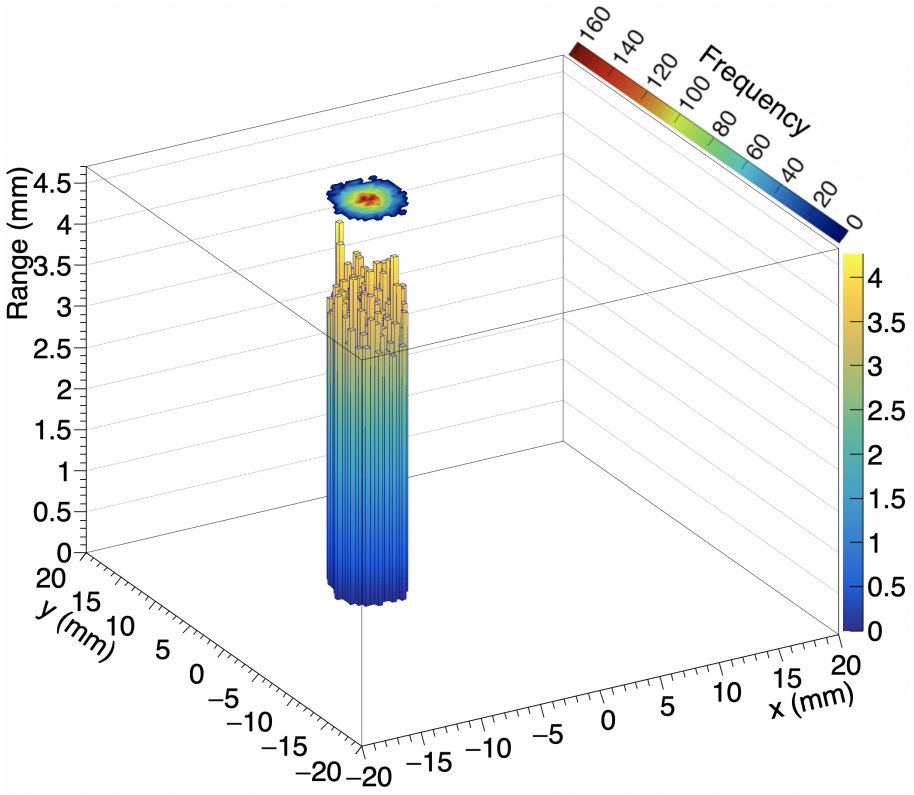


Figure 36: Reconstructed deposition in 3D for the off-centre collimator with 5.70 mm PMMA configuration. Over 13,000 proton events were matched.

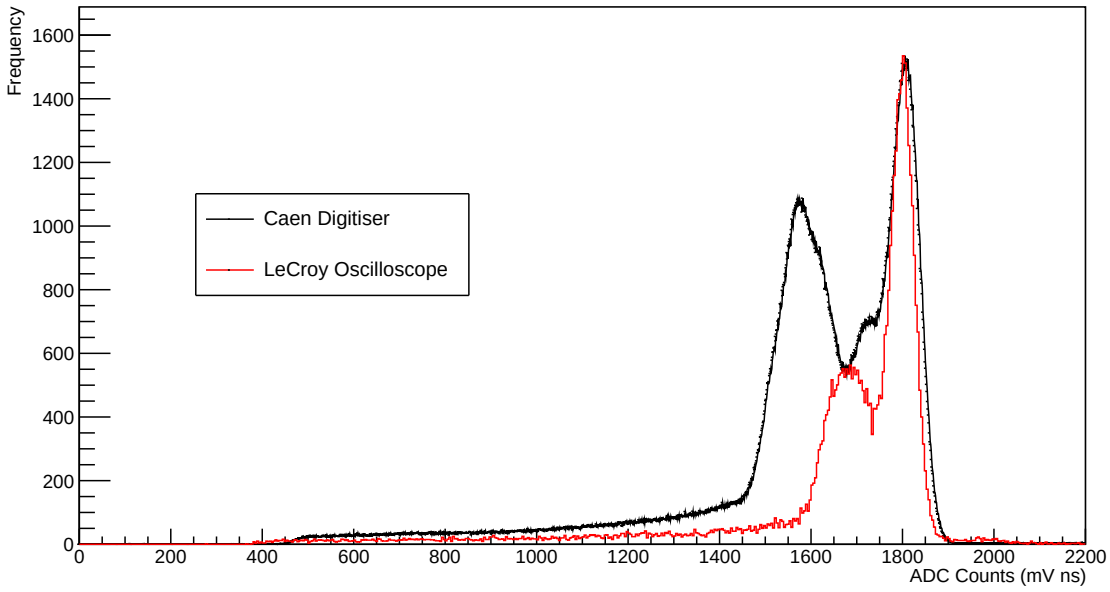


Figure 37: Scaled plot of the reconstructed Caen and LeCroy spectrum for the double collimator with no PMMA configuration. The approximate y-scale factor was used due to the noisy peak caused by current fluctuations in proton beam. The reduced χ^2 for the x-scale factor was 31.341. The LeCroy recorded over 54,000 events whereas the Caen recorded over 2,249,000.

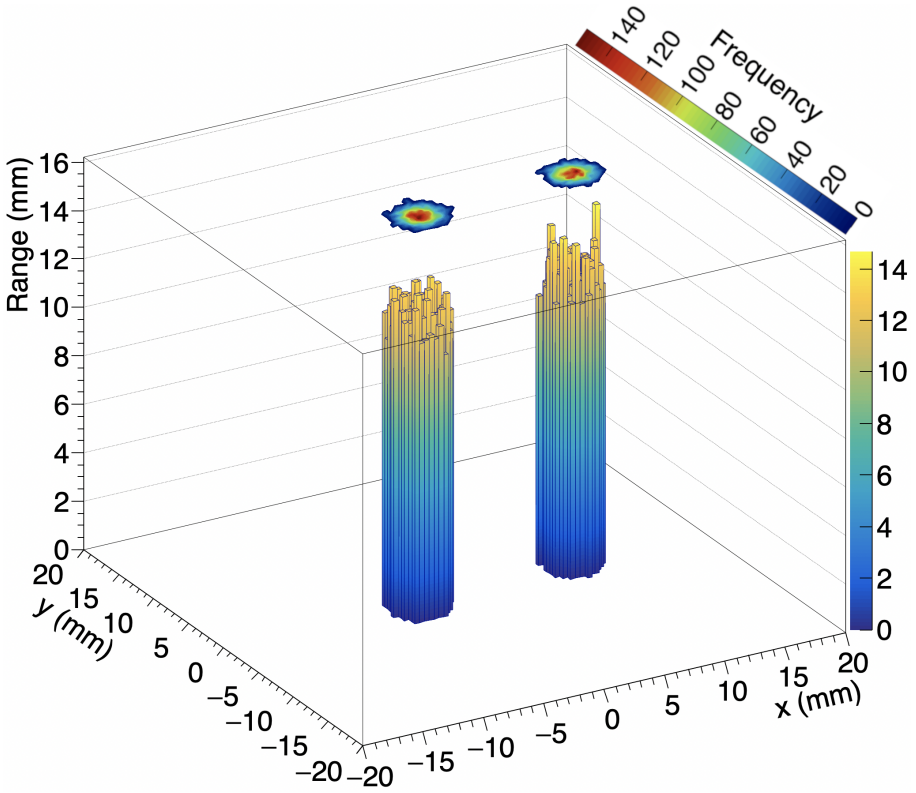


Figure 38: Reconstructed deposition in 3D for the double collimator with no PMMA configuration. Over 17,000 proton events were matched.

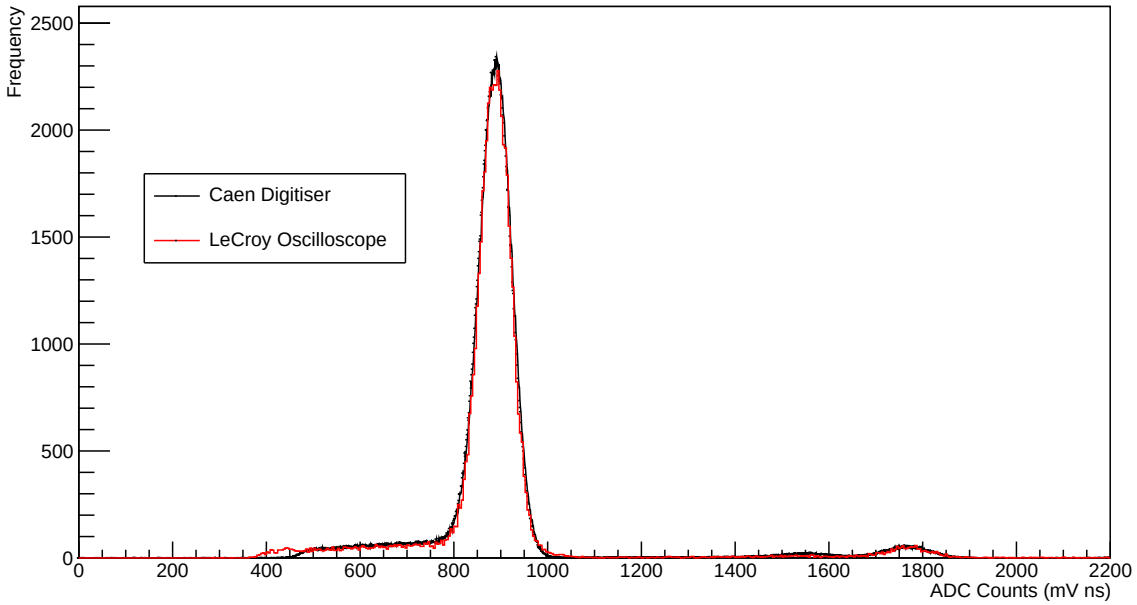


Figure 39: Scaled plot of the reconstructed Caen and LeCroy spectrum for the double collimator with 5.70 mm PMMA cover on both collimators configuration. The reduced χ^2 for the x-scale factor was 3.2867. The LeCroy recorded over 55,000 events whereas the Caen recorded over 904,000.

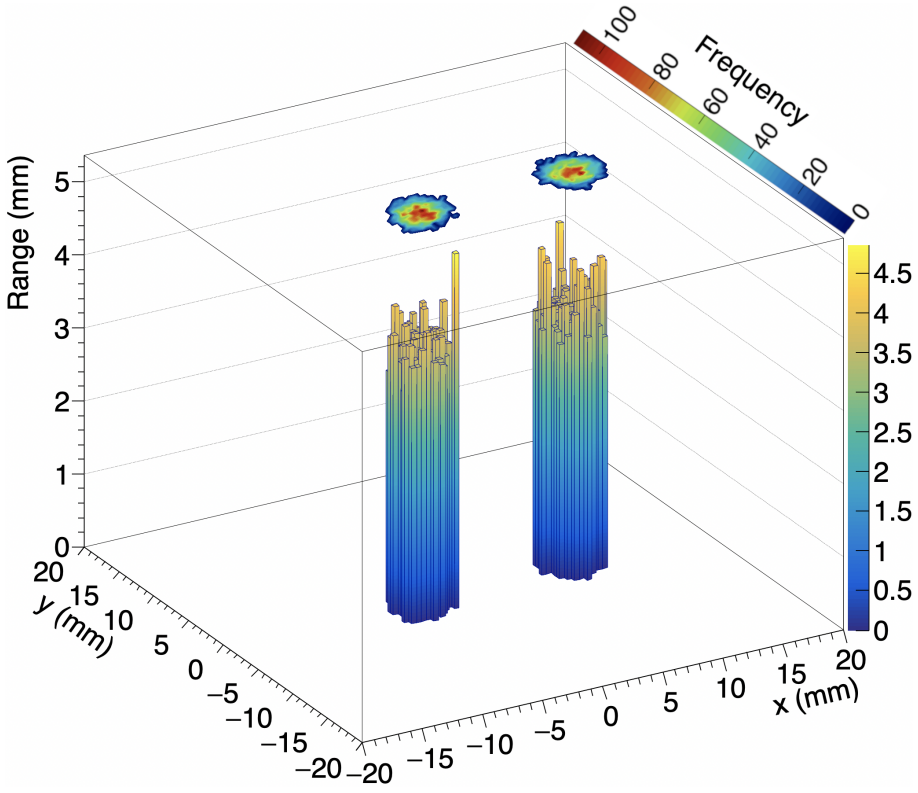


Figure 40: Reconstructed deposition in 3D for the double collimator with 5.70 mm PMMA cover on both collimators configuration. Over 18,000 proton events were matched.

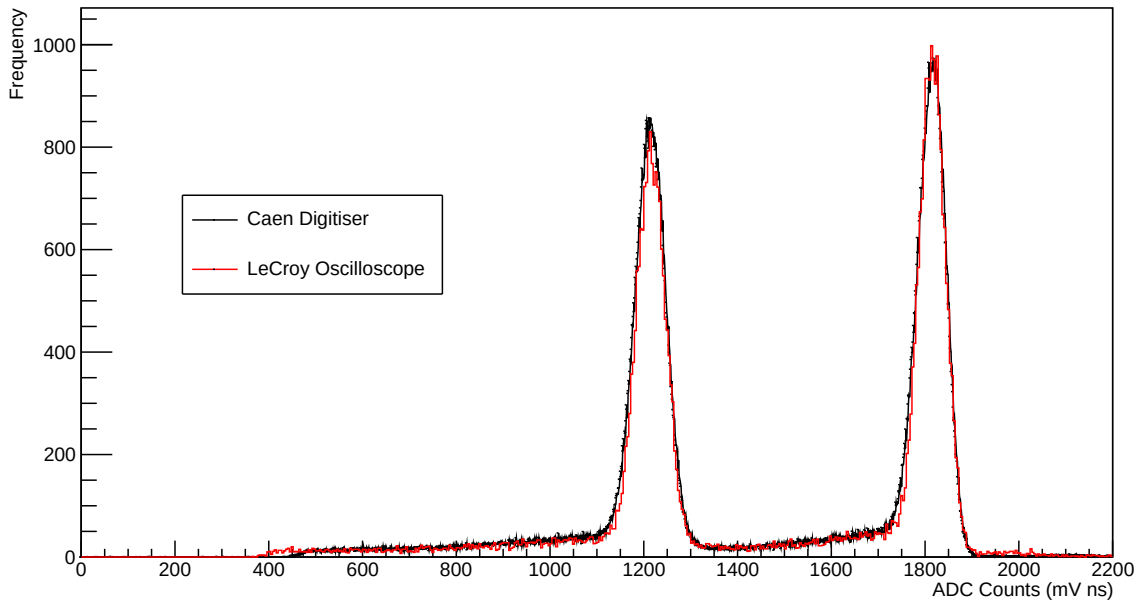


Figure 41: Scaled plot of the reconstructed Caen and LeCroy spectrum for the double collimator with 3.95 mm PMMA cover on one collimator configuration. The reduced χ^2 for the x-scale factor was 1.9955. The LeCroy recorded over 42,000 events whereas the Caen recorded over 508,000.

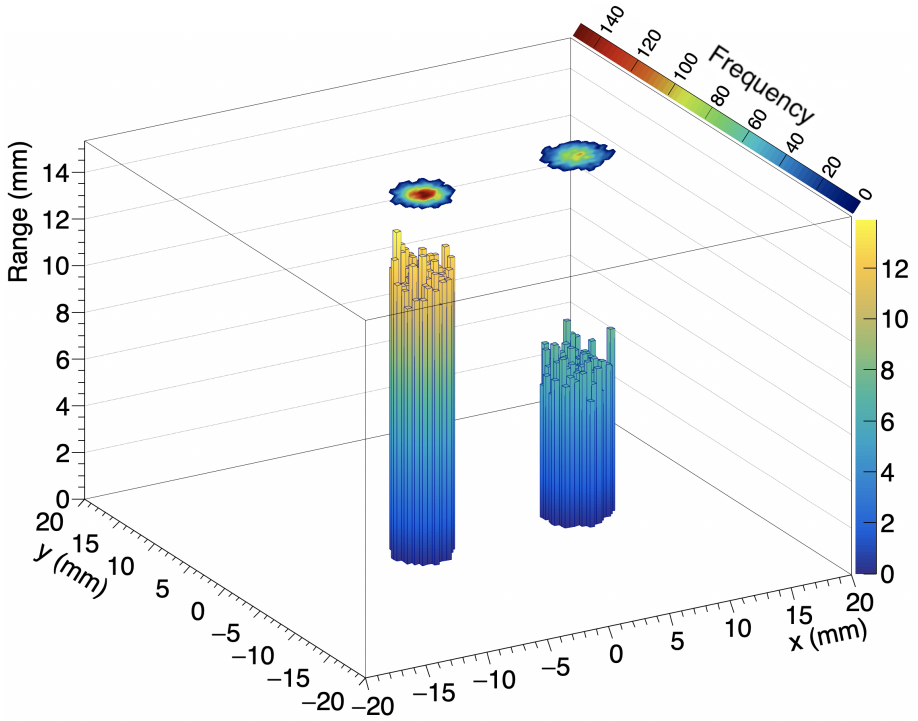


Figure 42: Reconstructed deposition in 3D for the double collimator with 3.95 mm PMMA cover on one collimator configuration. The collimator in the positive x-direction is covered. Protons travelling through the covered collimator have, on average, shorter range and are less numerous. Over 17,000 proton events were matched.

The reconstructed 3D depositions demonstrate the expected results: depositions with the centred collimator (Fig. 22, 24, 20, 26, 28, 30 and 32) show the range of protons decreasing with increasing PMMA thickness. Fig. 33 and 36 show a distinct shift of protons in the x-direction when the off-centred collimator is used, as well as showing reduction in range when PMMA is added. When the double collimator is used (Fig. 38, 40 and 42), two distinct columns of protons are observed. Most notably in Fig. 42, protons travelling through the covered collimator have on average shorter range and are less numerous.

The variation of the peak proton energy with thickness of PMMA is better illustrated using Fig. 43, which shows the expected decrease of peak energy with increasing thickness of PMMA. While there is insufficient data to draw any major conclusions, one can just begin to see a non-linear drop-off with thicker PMMA absorber, which is to be expected given the shape of the Bragg peak. For thin absorbers, the relation is approximately linear as one is still measuring energy from the tail of the Bragg peak. As the thickness increases, the measurement of energy takes place further along the Bragg peak, which would suggest a non-linear relationship beyond the tail of the peak. Further measurements made with thicker absorber sheets would be needed to verify this.

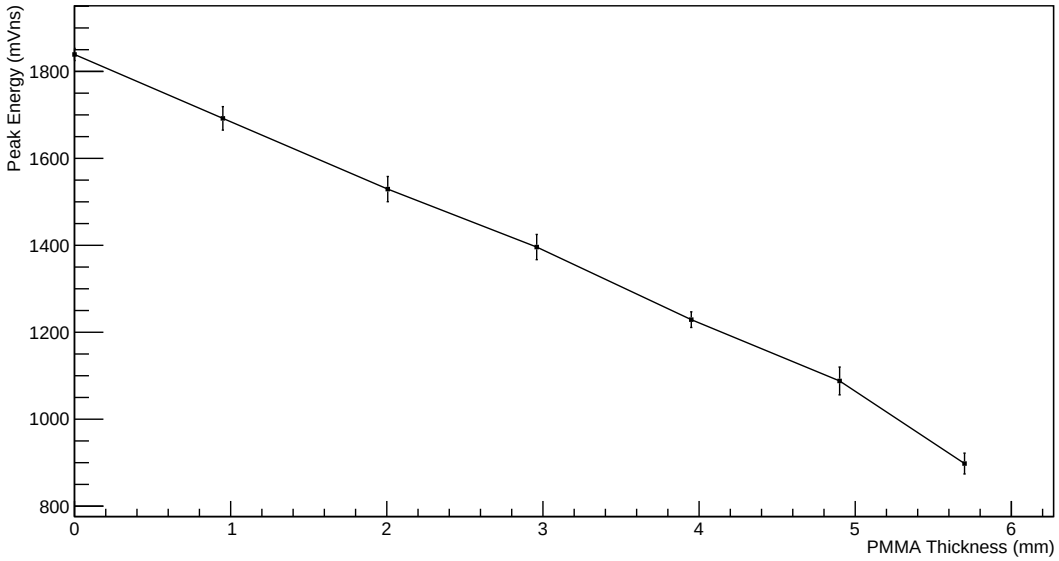


Figure 43: The variation in peak proton energy with thickness of PMMA absorber. Data from energy spectra shown in Fig. 21, 35 and 37 are excluded. Error bars represent the standard deviations of the peak in the energy spectra. For PMMA thicknesses that had multiple data sets, averages were calculated using standard error propagation results [46].

Using the standard deviation of each energy peak, the energy resolution of the calorimeter was calculated for each energy spectrum using [47]:

$$R = \frac{2.355\sigma}{E} \tag{4}$$

where R is the fractional resolution of the detector, 2.355σ is approximately the full-width at half-maximum of the (Gaussian-like) peak, σ is the standard deviation and E is the peak energy. The energy

resolution of the calorimeter was found to range between 3.5% and 9%, with energy resolution worsening at lower energies, as expected.

Largely due to current fluctuations in the cyclotron, the spectra shown in Fig. 21, 35 and 37 do not exhibit clean peaks in energy. For each of these spectra, the LeCroy appears to perform better than the Caen, as shown by the cleaner, narrower peaks. While this may be due to the LeCroy recording fewer events, it could also be the effects of the pile-up filtering measures put in place with the LeCroy. It should also be noted that for such data sets, the approximate y-scale factor was used in the scaling process, due to the number of events in the peak no longer being a reliable measure across the LeCroy and the Caen.

Finally, from all the runs that demonstrated clean energy spectra, it was expected that the x- and (normalised) y- scale factors would converge on some value. That is to say, some conversion factor to convert energies measured in the Caen to the mVns unit used in the LeCroy, and some frequency conversion factor to characterise the difference in performance in gathering protons. When calculating the y-scale factors by dividing the number of events in the peak of the energy spectrum of the Caen divided by the number of events in the peak of the LeCroy, it was found that the y-scale factor had to be multiplied by a factor of $\frac{1}{4}$ in order to give a reasonable result. Given that the Caen digitiser has 4 times as many bins as the LeCroy, it was thought that this was the source. Fig. 44 shows this factor being recovered in a scatter plot of the x- and normalised inverse y-scale factors. The average x-scale factor was found to be 0.94656 and the average normalised inverse y-scale factor was found to be 3.9998.

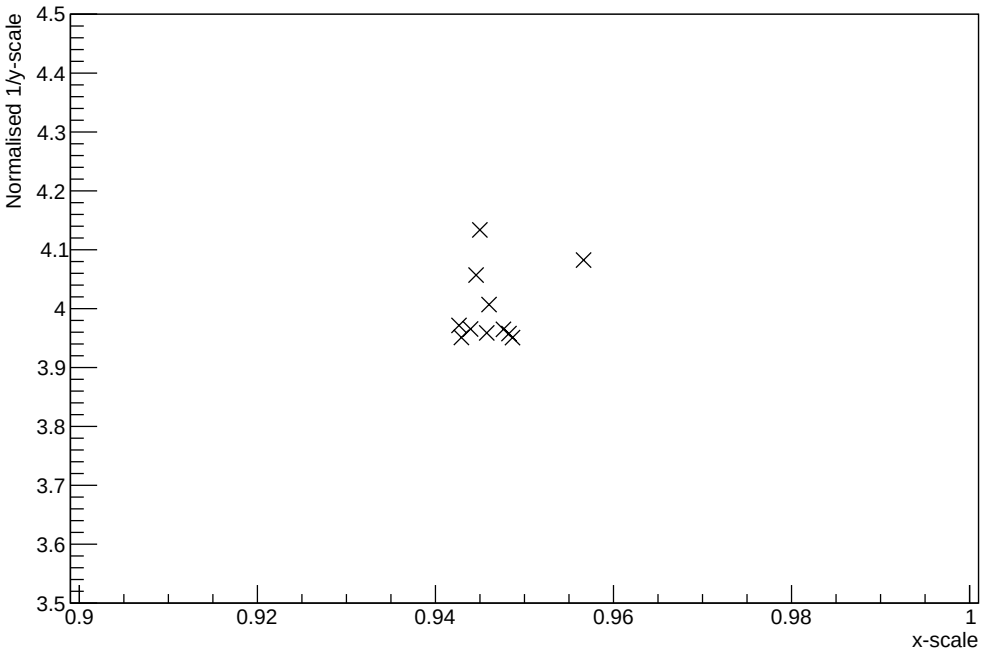


Figure 44: Scatter plot of the x-scale factors and inverse normalised y-scale factors for all energy spectra except those shown in Fig. 21, 35 and 37. y-scale factors were normalised by multiplying by the total number of LeCroy events divided by the total number of Caen events.

6 Conclusions

Both of the goals set out for this experiment have been achieved. Energy spectra for a variety of configurations were successfully reconstructed by the LeCroy oscilloscope, where it consistently performed as well as the Caen digitiser. Proof-of-principle has been provided for the use of the calorimeter and tracker simultaneously for proton position and range measurements: the 3D plots demonstrate successful reconstruction of the dose deposition, recovering physical features of configurations to sub-mm precision.

Recommended future work consists of performing similar beam tests described here but at higher particle rates. As mentioned previously, the detector used in this experiment cannot be used for clinical applications due to the particle rates being too high to resolve individual protons. For such uses, one would require a longer, segmented scintillator detector where the total light output of each slice is measured. However, one can still test how far the set up used in this experiment can go before saturation of the scintillator and/or PMT. If operating with high particle rates, one could reduce the light output of the scintillator by covering in black card or by spray-painting black instead of covering with foil, thus minimising the chance of saturation. One could also test a similar scintillator block as the one used in this experiment, to check the reproducibility of the results presented here. Finally, any future tests should endeavour to rectify the issue of the expanding calorimeter-tracker event matching tolerance window by supplying a reference sinusoidal signal from the LeCroy oscilloscope to the tracker.

With the future tests in mind, it is also suggested that the pile-up filtering routines for the LeCroy be improved further as pile-up will become a greater issue at higher beam currents. The work presented in this report largely focused on correctly implementing integration methods for the external trigger setup and correlating measurements across detectors rather than pile-up filtration. Since the work was successful in the first two aims, it is likely that the latter will make up a large part of future analysis. It is suggested that the current method of evaluating a baseline sigma in various regions in an acquisition be abandoned in favour of a routine that can find the number of pulses in an acquisition. While this would be challenging to implement, it would make the filtering more efficient and would tackle some of the already prevalent issues, such as filtering pile-up where there are overlapping pulses.

7 Acknowledgements

I would first like to thank my supervisors, Simon and Ruben, in particular Simon for his support and guidance, and for introducing me to this project – which I have found to be thoroughly engaging and rewarding. I would like to thank members of the UCL PBT group, Raffaella and Laurent, who were a fountain of knowledge and assistance. I would like to thank Anastasia from UCL SuperNEMO for her training in how not to break a scintillator. I would like to thank Tony at Birmingham University who took care of the tracker side of the project, who was essential for the project's success. I would like to thank Dan for his invaluable assistance in learning the code he had written. I would like to thank my dear friend Alex for the endless debates on whose MSci project was more important. I would also like to thank Siân, Dhruva, Kirsten and Lily for their moral support. Finally, I would like to give special thanks to my family, for their undying love and patience.

References

[1] Cancer Research UK. *Cancer Statistics for the UK*. URL: https://www.cancerresearchuk.org/health-professional/cancer-statistics-for-the-uk. [Accessed 8th October 2018].

[2] Department of Health and Social Care. *2010 to 2015 Government Policy: Cancer Research and Treatment*. URL: https://www.gov.uk/government/publications/2010-to-2015-government-policy-cancer-research-and-treatment/2010-to-2015-government-policy-cancer-research-and-treatment. [Accessed 3rd November 2018].

[3] R. W. Ruddon. "Characteristics of Human Cancer". In: *Cancer Biology*. 4th ed. Oxford University Press, 2007. Chap. 1, p. 4. ISBN: 9780195175448.

[4] A. J. F. Griffiths et al. "Genetics and the Organism". In: *Introduction to Genetic Analysis*. 10th ed. W. H. Freeman, 2011. Chap. 1. ISBN: 9781429276344.

[5] R. W. Ruddon. "Causes of Cancer". In: *Cancer Biology*. 4th ed. Oxford University Press, 2007. Chap. 2, pp. 33–45. ISBN: 9780195175448.

[6] R. A. Weinberg D. Hanahan. "The Hallmarks of Cancer". In: *Cell* 100 (Jan. 2000), pp. 57–70. DOI: https://doi.org/10.1016/S0092-8674(00)81683-9.

[7] R. A. Weinberg D. Hanahan. "The Hallmarks of Cancer: The Next Generation". In: *Cell* 144 (Mar. 2011), pp. 646–674. DOI: https://doi.org/10.1016/j.cell.2011.02.013.

[8] National Cancer Institute. *How Cancer is Diagnosed*. URL: https://www.cancer.gov/about-cancer/diagnosis-staging/diagnosis. [Accessed 8th October 2018].

[9] V. T. DeVita et al. "Cancer Screening". In: *Cancer: Principles and Practice of Oncology*. 10th ed. Wolters Kluwer, 2015. Chap. 34, pp. 370–389. ISBN: 9781451192940.

[10] Cancer Research UK. *Treatment for Cancer*. URL: https://www.cancerresearchuk.org/about-cancer/cancer-in-general/treatment. [Accessed 8th October 2018].

[11] R. Airley. "Principles of Cancer Chemotherapy". In: *Cancer Chemotherapy: Basic Science to the Clinic*. 1st ed. John Wiley & Sons Ltd., 2009. Chap. 7, p. 55. ISBN: 9780470092545.

[12] H. Fanet. "Interactions between Radiation and Matter: Consequences for Detection and Medical Imaging". In: *Photon-based Medical Imaging*. 1st ed. ISTE, 2011. Chap. 1, pp. 17–18. ISBN: 9781848212411.

[13] B. R. Martin. "Experimental Methods". In: *Nuclear and Particle Physics*. 1st ed. John Wiley & Sons Ltd., 2006. Chap. 4, pp. 123–145. ISBN: 9780470019993.

[14] H. Paganetti. "Proton Therapy: History and Rationale". In: *Proton Therapy Physics*. Ed. by H. Paganetti. 1st ed. CRC Press, 2012. Chap. 1, pp. 1–19. ISBN: 9781439836453.

[15] E. B. Podgorsak. "Treatment Machines for External Beam Radiotherapy". In: *Radiation Oncology Physics: A Handbook for Teachers and Students*. 1st ed. International Atomic Energy Agency, 2005. Chap. 5, pp. 123–129. ISBN: 9201073046.

[16] E. B. Podgorsak. "External Photon Beams: Physical Aspects". In: *Radiation Oncology Physics: A Handbook for Teachers and Students*. 1st ed. International Atomic Energy Agency, 2005. Chap. 6, pp. 169–172. ISBN: 9201073046.

[17] J. A. Reisz et al. "Effects of Ionizing Radiation on Biological Molecules — Mechanisms of Damage and Emerging Methods of Detection". In: *Antioxidants Redox Signal* 21 (June 2014), pp. 260–292. DOI: https://doi.org/10.1089/ars.2013.5489.

[18] E. B. Podgorsak. "Clinical Treatment Planning in External Photon Beam Radiotherapy". In: *Radiation Oncology Physics: A Handbook for Teachers and Students*. 1st ed. International Atomic Energy Agency, 2005. Chap. 7, pp. 251–255. ISBN: 9201073046.

[19] CERN. *The Changing Landscape of Cancer Therapy*. Jan. 2018. URL: https://iopp.fileburst.com/ccr/archive/CERNCourier2018JanFeb-digitaledition.pdf.

[20] C. Patrignani et al. "Review of Particle Physics". In: *Chinese Physics* 40.10 (Sept. 2016). DOI: https://doi.org/10.1088/1674-1137/40/10/100001.

[21] B. A. Zimmerman J. B. Marion. "Multiple Scattering of Charged Particles". In: *Nuclear Instruments and Methods* 51.1 (May 1967), pp. 93–101. DOI: https://doi.org/10.1016/0029-554X(67)90367-9.

[22] J. Marshall R. H. Dicke. "Inelastic Scattering of Protons". In: *Phys. Rev.* 63 (3-4 Feb. 1967), pp. 86–90. DOI: https://doi.org/10.1103/PhysRev.63.86.

[23] B. Gottschalk. "Physics of Proton Interactions in Matter". In: *Proton Therapy Physics*. Ed. by H. Paganetti. 1st ed. CRC Press, 2012. Chap. 2, pp. 21–37. ISBN: 9781439836453.

[24] W. Chen D. Jette. "Creating a Spread-out Bragg Peak in Proton Beams". In: *Phys. Med. Biol.* 56.11 (May 2011), N131. DOI: http://dx.doi.org/10.1088/0031-9155/56/11/N01.

[25] A. A. Miller. *Spread-out Bragg Peak*. URL: https://commons.wikimedia.org/wiki/File:BraggPeak.png. [Accessed 9th October 2018].

[26] G. T. Armstrong et al. "Aging and Risk of Severe, Disabling, Life-Threatening, and Fatal Events in the Childhood Cancer Survivor Study". In: *Clinical Oncology* 32.12 (Apr. 2014). DOI: https://doi.org/10.1200/JCO.2013.51.1055.

[27] Z. Li et al. "Quality Assurance and Commissioning". In: *Proton Therapy Physics*. Ed. by H. Paganetti. 1st ed. CRC Press, 2012. Chap. 8, pp. 221–263. ISBN: 9781439836453.

[28] O. Actis et al. "A Comprehensive and Efficient Daily Quality Assurance for PBS Proton Therapy". In: *Phys. Med. Biol.* 62.5 (Feb. 2017), pp. 1661–1675. DOI: https://doi.org/10.1088/1361-6560/aa5131.

[29] G. El Fakhri X. Zhu. "Proton Treatment Verification with PET Imaging". In: *Theranostics* 3.10 (Sept. 2013), pp. 731–740. DOI: https://doi.org/10.7150/thno.5162.

[30] E. Pedroni B. Schaffner. "The Precision of Proton Range Calculations in Proton Radiotherapy Treatment Planning: Experimental Verification of the Relation between CT-HU and Proton Stopping Power". In: *Phys. Med. Biol.* 43 (Feb. 1998), pp. 1579–1592. DOI: https://doi.org/10.1088/0031-9155/43/6/016.

[31] Seattle Cancer Care Alliance Proton Therapy Centre. *Gastrointestinal Cancer Treatments*. URL: https://www.sccaprotontherapy.com/cancers-treated/gastrointestinal-cancer-treatment. [Accessed 24th February 2019].

[32] S. Molinelli et al. "Dosimetric Accuracy Assessment of a Treatment Plan Verification System for Scanned Proton Beam Radiotherapy: One-year Experimental Results and Monte Carlo Analysis of the Involved Uncertainties". In: *Phys. Med. Biol.* 58 (May 2013), pp. 3837–3847. DOI: https://doi.org/10.1088/0031-9155/58/11/3837.

[33] W. R. Leo. "Scintillation Detectors". In: *Techniques for Nuclear and Particle Physics Experiments*. 2nd ed. Springer-Verlag Berlin Heidelberg, 1994. Chap. 7, pp. 157–175. ISBN: 9783642579202.

[34] M. Thomson. "Introduction". In: *Modern Particle Physics*. 1st ed. Cambridge University Press, 2013. Chap. 1, pp. 13–22. ISBN: 9781107034266.

[35] J. B. Birks. "Scintillations from Organic Crystals – Specific Fluorescence and Relative Response to Different Radiations". In: *Proc. Phys. Soc.* 64.10 (Apr. 1951), p. 874. DOI: https://doi.org/10.1088/0370-1298/64/10/303.

[36] J. R. Hook & H. E. Hall. "Semiconductors". In: *Solid State Physics*. 2nd ed. John Wiley & Sons, 1991. Chap. 5, pp. 131–149. ISBN: 9780471928041.

[37] W. R. Leo. "Semiconductor Detectors". In: *Techniques for Nuclear and Particle Physics Experiments*. 2nd ed. Springer-Verlag Berlin Heidelberg, 1994. Chap. 10, pp. 215–246. ISBN: 9783642579202.

[38] J. R. Hook & H. E. Hall. "Semiconductor Devices". In: *Solid State Physics*. 2nd ed. John Wiley & Sons, 1991. Chap. 6, pp. 169–181. ISBN: 9780471928041.

[39] A. Freshville and the SuperNEMO Collaboration. "Calorimeter R&D for the SuperNEMO Double Beta Decay Experiment". In: *Journal of Physics: Conference Series* 293.1 (2011), p. 012037. DOI: https://doi.org/10.1088/1742-6596/293/1/012037.

[40] J. T. Taylor et al. "A New Silicon Tracker for Proton Imaging and Dosimetry". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 831 (May 2016), pp. 362–366. DOI: http://dx.doi.org/10.1016/j.nima.2016.02.013.

[41] M. Esposito et al. "PRaVDA: The First Solid-State System for Proton Computed Tomography". In: *Physica Medica* 55 (Nov. 2018), pp. 149–154. DOI: https://doi.org/10.1016/j.ejmp.2018.10.020.

[42] Eric W. Weisstein. *Newton-Cotes Formulas*. URL: http://mathworld.wolfram.com/Newton-CotesFormulas.html. [Accessed 17th February 2019].

[43] "Integration". In: *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Ed. by M. Abramowitz and I. A. Stegun. 4th ed. Dover, 1972. Chap. 25.4, pp. 885–887. ISBN: 9780486612720.

[44] N. D. Gagunashvili. "Comparison of Weighted and Unweighted Histograms". In: *PoS (ACAT)* (May 2006). DOI: arXiv:physics/0605123.

[45] CERN. *ROOT: TH1 Class Reference*. Nov. 2018. URL: https://root.cern.ch/doc/v614/classTH1.html. [Accessed 17th March 2019].

[46] I. G. Hughes & T. P. A. Hase. "Error Propagation". In: *Measurements and their Uncertainties*. 1st ed. Oxford University Press, 2010. Chap. 4, p. 44. ISBN: 9780199566327.

[47] W. R. Leo. "General Characteristics of Detectors". In: *Techniques for Nuclear and Particle Physics Experiments*. 2nd ed. Springer-Verlag Berlin Heidelberg, 1994. Chap. 5, pp. 115–127. ISBN: 9783642579202.

A Analysis Code

What follows is the C++ code written for this project for the analysis of calorimeter data (from both the Caen digitiser and LeCroy oscilloscope) and PRaVDA tracker data, which is based on code written in previous years.

Caen.h

```
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <TCanvas.h>
#include <TApplication.h>
#include <TGraph.h>
#include <TH1.h>
using namespace std;

#ifndef CAEN_H_
#define CAEN_H_

class Caen {
public:
        vector<double> bins;
        vector<double> counts;

        Caen(vector<double> bins, vector<double> counts) {
                this->bins = bins;
                this->counts = counts;
        }
        ~Caen() {
                this->bins.clear();
                this->counts.clear();
        }
        vector<double> getBins() {
                return this->bins;
        }
        vector<double> getCounts() {
                return this->counts;
        }
};
Caen readCaen(string filename);
```

```
void plotCaen(Caen run, string name, string dest, int xhigh,
    TApplication* app);
void plotCaenHist(Caen run, string name, string dest, int xhigh,
    TApplication* app);

#endif /* CAEN_H_ */
```

Caen.cpp

```
#include "Caen.h"

Caen readCaen(string filename) {
        ifstream file(filename,ifstream::in);
        vector<double> bins;
        vector<double> counts;
        string line;
        while (getline(file, line)) {
                stringstream ss(line);
                int a, b;
                ss>>a>>b;
                bins.push_back(a);
                counts.push_back(b);
        }
        Caen run = Caen(bins,counts);
        return run;
}
void plotCaen(Caen run, string name, string dest, int xhigh,
    TApplication* app) {
        TGraph* graph = new TGraph(run.getBins().size(), run.getBins().data(),
            run.getCounts().data());
        graph->GetXaxis()->SetLimits(0, xhigh);
        graph->GetXaxis()->SetTitle("ADC Counts (mV ns)");
        graph->GetYaxis()->SetTitle("Frequency");
        TCanvas* c1 = new TCanvas("Caen ADC Spectrum", "Caen ADC Spectrum",
            1920, 1080);
        dest = dest+"/"+name+".pdf";
        const char* dest_c = dest.c_str();
        const char* name_c = name.c_str();
        graph->SetNameTitle(name_c,name_c);
        graph->Draw();
        c1->Update();
        c1->Print(dest_c);
```

```
        app->Run(graph);
        delete graph;
        delete c1;
}
void plotCaenHist(Caen run, string name, string dest, int xhigh,
    TApplication* app) {
        string title = name+";ADC Counts (mV ns);Frequency";
        const char* title_c = title.c_str();
        TH1D* hist = new TH1D("ADC Counts", title_c, run.getBins().size(),
            0, xhigh);
        for (int i=0; i<run.getBins().size(); i++) {
                int bin = run.getBins()[i];
                int freq = run.getCounts()[i];
                for (int j=0; j<freq; j++) {
                        hist->Fill(bin);
                }
        }
        TCanvas* c1 = new TCanvas("Caen ADC Spectrum", "Caen ADC Spectrum",
            1920, 1080);
        dest = dest+"/"+name+".pdf";
        const char* dest_c = dest.c_str();
        hist->Draw();
        c1->Update();
        c1->Print(dest_c);
        app->Run(hist);
        delete hist;
        delete c1;
}
```

Tracker.h

```
#include <vector>
#include <sstream>
#include <string>
#include <iostream>
#include <fstream>
#include <TApplication.h>
#include <TCanvas.h>
#include <TH2.h>
using namespace std;

#ifndef TRACKER_H_
```

```
#define TRACKER_H_

class Tracker {
public:
        double timestamp;
        vector<double> coords; //A vector of 60 numbers: 4 trackers,
            up to 5 protons each, 3 coords. 3rd coord always 198.
        bool goodHit; //Only want tracker data with one recorded hit

        Tracker(double timestamp, vector<double> coords, bool goodHit) {
                this->timestamp = timestamp;
                this->coords = coords;
                this->goodHit = goodHit;
        }
        ~Tracker() {
                this->coords.clear();
        }
        double getTimestamp() {
                return this->timestamp;
        }
        vector<double> getCoords() {
                return this->coords;
        }
        bool getGoodHit() {
                return this->goodHit;
        }
};
vector<Tracker> readTracker(string filename);
void histTracker(vector<Tracker> run, int bins, string name, string dest,
    TApplication* app);

#endif /* TRACKER_H_ */
```

Tracker.cpp

```
#include "Tracker.h"

vector<Tracker> readTracker(string filename) {
        cout<<"Loading tracker data..."<<endl;
        ifstream file(filename,ifstream::in);
        vector<Tracker> run;
        string line;
```

```
        while (getline(file, line)) {
                bool goodHit = false;
                stringstream ss(line);
                double timestamp;
                ss>>timestamp;
                vector<double> coords;
                for (int i=0; i<60; i++) {
                        double a;
                        ss>>a;
                        coords.push_back(a);
                }
                if (coords[3] == -1) {
                        goodHit = true;
                }
                Tracker hit = Tracker(timestamp, coords, goodHit);
                run.push_back(hit);
        }
        cout<<"The tracker file contained a total of "<<run.size()<<
            " hits."<<endl;
        return run;
}
void histTracker(vector<Tracker> run, int bins, string name, string dest,
    TApplication* app) {
        string title = name+";x (mm);y (mm)";
        const char* name_c = name.c_str();
        const char* title_c = title.c_str();
        TH2D* hist = new TH2D("Tracker Data", title_c, bins, -20.0, 20.0,
            bins, -20.0, 20.0);
        TCanvas* c1 = new TCanvas(name_c, name_c, 1920, 1080);
        for(int i=0; i<run.size(); i++) {
                if (run[i].getGoodHit()) { //Removing this check will plot
                    bad acqusitions too
                        hist->Fill(run[i].getCoords()[0],
                            run[i].getCoords()[1]);
                }
        }
        const char* dest_c;
        dest = dest+"/"+name+".pdf";
        dest_c = dest.c_str();
        hist->Draw("CONT0, COLZ");
        c1->Update();
```

```
        c1->Print(dest_c);
        app->Run(hist);
        delete hist;
        delete c1;
}
```

LeCroy.h

```
#include <TMath.h>
#include <TH1.h>
#include <TH3.h>
#include <TCanvas.h>
#include <TColor.h>
#include <TPad.h>
#include <TGraph.h>
#include <TROOT.h>
#include <TStyle.h>
#include <TApplication.h>
#include <boost/lexical_cast.hpp>
#include <vector>
#include <cmath>
#include <fstream>
#include <string>
#include <cstring>
#include <sstream>
#include <time.h>
#include <iostream>
#include "Caen.h"
#include "Tracker.h"
using namespace std;

#ifndef LECROY_H_
#define LECROY_H_

class Parameters {
public:
        //USER-DEFINED VARIABLES
        //tolerance for what can be considered background
        double maxBaselineSigma;
        double window; //length of time to integrate over
        int bins; //Number of histogram bins
        int xlow; //Upper and lower limits of histogram x-axis
```

```
        int xhigh;

        Parameters(double maxBaselineSigma, double window, int bins,
            int xlow, int xhigh) {
                this->maxBaselineSigma = maxBaselineSigma;
                this->window = window;
                this->bins = bins;
                this->xlow = xlow;
                this->xhigh = xhigh;
        }
        double getMaxSigma() {
                return this->maxBaselineSigma;
        }
        double getWindow() {
                return this->window;
        }
        int getBins() {
                return this->bins;
        }
        int getXlow() {
                return this->xlow;
        }
        int getXhigh() {
                return this->xhigh;
        }
        void setMaxSigma(double s) {
                this->maxBaselineSigma = s;
        }
        void setWindow(double w) {
                this->window = w;
        }
        void setBins(int b) {
                this->bins = b;
        }
        void setXlow(int l) {
                this->xlow = l;
        }
        void setXhigh(int h) {
                this->xhigh = h;
        }
};
```

```
class LeCroyData {
public:
        string fileName; //File directory on disk
        string vertUnit; //Vertical axis unit
        string horizUnit; //Horizontal axis unit
        string timestamp; //Time of trigger

        short commType; //either _0 for byte, or _1 for word

        float verticalGain;
        //floating values from raw data: verticalGain*data - verticalOffset
        float verticalOffset;
        float horizUncertainty;

        int pointsPerWindow; //number of points in the integration window
        double dx;

    //array of trigger times, relative to the first trigger
        double *trigtimeArray;
        //array of start of data collection for each acquisition, relative
            to that acquisition trigger
        double *horizOffsetArray;
        //array of time values for all acquisitions before baseline
            subtraction
        double *timeArray;
        //array of p.d. values for all acquisitions before baseline
            subtraction
        double *waveArray;

        //LIST OF VARIABLES TO CALCULATE

        int pointsPerAcq; //number of points per acquisition
        long subArrayCount; //number of acquisitions
        vector<bool> goodHits; //index of good/bad labels for acquisitions
    //list of acquisitions in vector format, baseline subtracted
        vector<double*> Acqs;
        vector<double*> Times;
        vector<double> spectrumADCCounts; //integrals of acquisitions
        vector<double> spectrumTime; //times of integrated acquisitions

        LeCroyData(string fileName, string vertUnit, string horizUnit,
```

```
    string timestamp, short commType, float verticalGain,
    float verticalOffset, float horizUncertainty,
    double* trigtimeArray, double* horizOffsetArray,
    double* waveArray, double* timeArray, int pointsPerAcq,
    long subArrayCount, double dx, int pointsPerWindow,
    vector<double*> Acqs, vector<double*> Times,
vector<bool> goodHits, vector<double> spectrumTime,
        vector<double> spectrumADCCounts) {
        this->fileName = fileName;
        this->vertUnit = vertUnit;
        this->horizUnit = horizUnit;
        this->timestamp = timestamp;
        this->commType = commType;
        this->verticalGain = verticalGain;
        this->verticalOffset = verticalOffset;
        this->horizUncertainty = horizUncertainty;
        this->trigtimeArray = trigtimeArray;
        this->horizOffsetArray = horizOffsetArray;
        this->waveArray = waveArray;
        this->timeArray = timeArray;
        this->pointsPerAcq = pointsPerAcq;
        this->subArrayCount = subArrayCount;
        this->dx = dx;
        this->pointsPerWindow = pointsPerWindow;
        this->Acqs = Acqs;
        this->Times = Times;
        this->spectrumTime = spectrumTime;
        this->spectrumADCCounts = spectrumADCCounts;
        this->goodHits = goodHits;
}
~LeCroyData() {
        this->Acqs.clear();
        this->goodHits.clear();
        this->spectrumTime.clear();
        this->spectrumADCCounts.clear();
}
string getFileName() {
        return this->fileName;
}
string getVertUnit() {
        return this->vertUnit;
```

```
}
string getHorizUnit() {
        return this->horizUnit;
}
string getTimestamp() {
        return this->timestamp;
}
short getCommType() {
        return this->commType;
}
float getVerticalGain() {
        return this->verticalGain;
}
float getVerticalOffset() {
        return this->verticalOffset;
}
float getHorizUncertainty() {
        return this->horizUncertainty;
}
double* getTrigTimeArray() {
        return this->trigtimeArray;
}
double* getHorizOffsetArray() {
        return this->horizOffsetArray;
}
//Getter for vector of oscilloscope voltage values
double* getWaveArray() {
        return this->waveArray;
}
//Getter for vector of oscilloscope time values
double* getTimeArray() {
        return this->timeArray;
}
int getPointsPerAcq() {
        return this->pointsPerAcq;
}
long getSubAcqCount() {
        return this->subArrayCount;
}
double getDx() {
        return this->dx;
```

```
}
int getPointsPerWindow() {
        return this->pointsPerWindow;
}
void setPointsPerWindow(int p) {
        this->pointsPerWindow = p;
}
vector<double*> getAcqs() {
        return this->Acqs;
}
void appendAcqs(double* p) {
        this->Acqs.push_back(p);
}
vector<double*> getTimes() {
        return this->Times;
}
void appendTimes(double* p) {
        this->Times.push_back(p);
}
vector<double> getSpectrumTime() {
        return this->spectrumTime;
}
void appendSpectrumTime(double i) {
        this->spectrumTime.push_back(i);
}
vector<double> getSpectrumADCCounts() {
        return this->spectrumADCCounts;
}
void appendSpectrumADCCounts(double i) {
        this->spectrumADCCounts.push_back(i);
}
vector<bool> getGoodHits() {
        return this->goodHits;
}
void appendGoodHits(bool b) {
        this->goodHits.push_back(b);
}
string getHeader(){
        //Returns a string describing the file
        ostringstream header;
        header<<"File name: "<<fileName<<endl;
```

```
                header<<"Timestamp: "<<timestamp<<endl;
                header<<"Number of acquisitions: "<<subArrayCount<<endl;
                header<<"Points per acquisition: "<<pointsPerAcq<<endl;
                header<<"Vertical unit: "<<vertUnit<<endl;
                header<<"Timing uncertainty: "<<horizUncertainty<<endl;
                header<<"Horizontal unit: "<<horizUnit<<endl;
                double a, b;
                a = timeArray[0];
                b = timeArray[pointsPerAcq*subArrayCount-1];
                header<<"First and last timepoints: "<<a<<", "
                    <<b<<" (span: "<<(b-a)<<horizUnit<<" )"<<endl;
                return header.str();
        }
};
class Hit {
public:
        double timestamp;
        double x;
        double y;
        double energy;

        Hit(double timestamp, double x, double y, double energy) {
                this->timestamp = timestamp;
                this->x = x;
                this->y = y;
                this->energy = energy;
        }
        double getTimestamp() {
                return this->timestamp;
        }
        double getX() {
                return this->x;
        }
        double getY() {
                return this->y;
        }
        double getEnergy() {
                return this->energy;
        }
};
//----------------METHODS----------------\\
```

```
//Data conversion
short getShort(int byteLocation);
//Creates a long from the fileBuffer, starting at byteLocation
long getLong(int byteLocation);
//Creates a float from the fileBuffer, starting at byteLocation
float getFloat(int byteLocation);
//Creates a double from the fileBuffer, starting at byteLocation
double getDouble(int byteLocation);
//Creates an int from the fileBuffer, starting at byteLocation
int getInt(int byteLocation);
//Returns the byte at byteLocation in the fileBuffer as a signed char
signed char getByte(int byteLocation);
//Creates a string of 16 characters from the fileBuffer,
    starting at byteLocation
string get16CharString(int byteLocation);
//Creates a string of byteLength characters from the fileBuffer,
    starting at byteLocation
string getText(int byteLocation, long byteLength);

//Reading
bool isTrace(string fileName);
LeCroyData readFile(string fileName, Parameters P);
vector<LeCroyData> Load(int n, string path, Parameters P);

//Analysis
double* getRawAcqWave(int segment, LeCroyData lcd);
double* getRawAcqTime(int segment, LeCroyData lcd);
double meanEstimateBaseline(double* data, double* time, LeCroyData &lcd,
    Parameters P);
void baselineSubtraction(LeCroyData &lcd, Parameters P);
double compositeIntegrate(double* goodAcq, int pointsPerWindow,
    int pointsPerAcq, double dx);
void buildSpectrum(LeCroyData &lcd);
vector<LeCroyData> Analysis(int n, string path, Parameters P);
void plotRawWaveform(LeCroyData lcd, int index, string name, string dest,
    TApplication* app);
void plotSubtractedWaveform(LeCroyData lcd, int index, string name,
    string dest, TApplication* app);
void Histogram(vector<LeCroyData> run, Parameters P, string name,
    string dest, TApplication* app);
void timeHistogram(vector<LeCroyData> run, Parameters P, string name,
```

```
    string dest, TApplication* app);
void Write(vector<LeCroyData> run, string name, string dest);
void program(TApplication* app, Parameters P);

void plotBoth(vector<LeCroyData> run, Caen caen, Parameters P, string name,
    string dest, TApplication* app);
vector<Hit> Match(vector<double> leCroyEnergies, vector<double> leCroyTime,
    vector<bool> goodHits, vector<Tracker> tracker);
void writeMatch(vector<Hit> run, string name, string dest);
void plot3D(vector<Hit> hits, int bins, string name, string dest,
    TApplication* app);

#endif /* LECROY_H_ */
```

LeCroy.cpp

```
#include "LeCroy.h"

char* fileBuffer; //Dynamic variable for storing the contents of the file

bool isTrace(string fileName) {
        //Identifies whether a file name represents a LeCroy trace file based
            on the file extension
        string lastFour = fileName.substr(fileName.length() - 4);
        if(lastFour == ".trc"){
                return true;
        }
        return false;
}
short getShort(int byteLocation){
        //Get the value of a float from byteLocation in the fileBuffer
        short s;
        char bytes[sizeof s];
        for(int n = 0; n<(sizeof s); n++){
        //Select bytes to form the short
                bytes[n] = fileBuffer[n+byteLocation];
        }
        memcpy(&s, &bytes, sizeof s); //Copy the bit pattern into the short
        return s;
}
long getLong(int byteLocation){
        //Get the vlaue of a long from byteLocation in the fileBuffer
```

```
        long l;
        char bytes[sizeof l];
        for(int n = 0; n<(sizeof l); n++){
        //Select bytes to form the long
                bytes[n] = fileBuffer[n+byteLocation];
        }
        memcpy(&l, &bytes, sizeof l); //Copy the bit pattern into the long
        return l;
}
int getInt(int byteLocation){
        //Get the value of an int from byteLocation in the fileBuffer
        int i;
        char bytes[sizeof i];
        for(int n = 0; n<(sizeof i); n++){
                //Select bytes to form the int
                bytes[n] = fileBuffer[n+byteLocation];
        }
        memcpy(&i, &bytes, sizeof i); //Copy the bit pattern into the int
        return i;
}
float getFloat(int byteLocation){
        //Get the value of a float from byteLocation in the fileBuffer
        float f;
        char bytes[sizeof f];
        for(int n = 0; n<(sizeof f); n++){
            //Select bytes to form the float
                bytes[n] = fileBuffer[n+byteLocation];
        }
        memcpy(&f, &bytes, sizeof f); //Copy the bit pattern into the float
        return f;
}
double getDouble(int byteLocation){
        //Get the value of a double from byteLocation in the fileBuffer
        double d;
        char bytes[sizeof d];
        for(int n = 0; n<(sizeof d); n++){
                //Select bytes to form the double
                bytes[n] = fileBuffer[n+byteLocation];
        }
        memcpy(&d, &bytes, sizeof d); //Copy the bit pattern into the double
        return d;
```

```
}
signed char getByte(int byteLocation){
        //Get the value of a signed char from byteLocation in the fileBuffer
        signed char b = fileBuffer[byteLocation];
        return b;
}
string getText(int byteLocation, long byteLength){
        //Converts byteLength bytes from the fileBuffer, starting from
            byteLocation, into a string
        ostringstream text;
        //For each character in the specified region of the fileBuffer
        for(int n = 0; n<byteLength; n++){
            //Append the character to the string
                text<<fileBuffer[byteLocation+n];
        }
        return text.str();
}
string get16CharString(int byteLocation){
        //Converts bytes from the fileBuffer, starting from byteLocation,
            into a 16-character string
        return getText(byteLocation, 16);
}
string makeTimestamp(int byteLocation){
        //Constructs a string describing the date and time at which the
            file was created as specified in the LeCroy X-Stream manual
        ostringstream stamp;

        //Get timing info
        double sec = getDouble(byteLocation);
        signed char min = getByte(byteLocation + 8);
        signed char hrs = getByte(byteLocation + 9);
        signed char day = getByte(byteLocation + 10);
        signed char mon = getByte(byteLocation + 11);
        short yrs = getShort(byteLocation + 12);

        //Construct the string
        stamp<<(int)day<<"/"<<(int)mon<<"/"<<yrs<<" @ "<<(int)hrs<<":"<<
            (int)min<<":"<<sec;
        return stamp.str();
}
LeCroyData readFile(string fileName, Parameters p) {
```

```
if(!isTrace(fileName)) {
        throw "Given filename is not a LeCroy binary trace file!";
}
//open the file at the last byte to determine file size in bytes
ifstream target(fileName, ios::in|ios::binary|ios::ate);
int fileSize = target.tellg();

//Identify the byte corresponding to the start of "WAVEDESC"
char memblock[50];
target.seekg(0);
target.read(memblock, 50);
string blockString(memblock);
int wavedescIndex = blockString.find("WAVEDESC");

//read the file into memory
target.seekg(0);
fileBuffer = new char[fileSize];
target.read(fileBuffer, fileSize);
target.close();

//read in each data field from its known byte location
short commType = getShort(wavedescIndex + 32);
long waveDescriptorSize = getLong(wavedescIndex + 36);
long userTextSize = getLong(wavedescIndex + 40);
long trigtimeArraySize = getLong(wavedescIndex + 48);
long waveArraySize = getLong(wavedescIndex + 60);
float verticalGain = getFloat(wavedescIndex + 156);
float verticalOffset = getFloat(wavedescIndex + 160);
float horizInterval = getFloat(wavedescIndex + 176);
string vertUnit = getText(wavedescIndex + 196, 48);
string horizUnit = getText(wavedescIndex + 244, 48);
float horizUncertainty = getFloat(wavedescIndex + 292);
string timestamp = makeTimestamp(wavedescIndex + 296);

int subArrayCount = trigtimeArraySize/(2*sizeof(double));

//Loads the TRIGTIME_ARRAY block into a pair of double[]s
double* trigtimeArray = new double[subArrayCount];
double* horizOffsetArray = new double[subArrayCount];
for(int i = 0; i<subArrayCount; i++){
        trigtimeArray[i] = 1e9*(getDouble(wavedescIndex +
```

```
            (int)waveDescriptorSize + (int)userTextSize +
            2*i*sizeof(double)));
        horizOffsetArray[i] = 1e9*(getDouble(wavedescIndex +
            (int)waveDescriptorSize + (int)userTextSize +
            (2*i+1)*sizeof(double)));
}
double* waveArray;
if(commType == 0){ //If the WAVEARRAY is expressed in bytes...
        waveArray = new double[waveArraySize];
        //Load the waveArray using bytes
        for(int i = 0; i<waveArraySize; i++){
                waveArray[i] = 1e3*((double)verticalGain*getByte(
                    wavedescIndex + waveDescriptorSize + userTextSize
                    + trigtimeArraySize + i) -
                    (double)verticalOffset);
        }
}
else {
        waveArray = new double[waveArraySize/2];
        //If the WAVEARRAY is expressed in words, load the waveArray
            using words
        for(int i = 0; i<waveArraySize/2; i++){
                waveArray[i] = 1e3*((double)verticalGain*getShort(
                    wavedescIndex + waveDescriptorSize + userTextSize
                    + trigtimeArraySize + 2*i) -
                    (double)verticalOffset);
        }
}
//Calculate the number of points in each triggered acquisition
int pointsPerAcq = waveArraySize/(subArrayCount*(commType+1));
//Calculate the time in seconds of each data point in the
    waveArray and append it to the timeArray
double* timeArray = new double[subArrayCount*pointsPerAcq];
for(int i = 0; i<subArrayCount; i++){
        for(int j = 0; j<pointsPerAcq; j++){
                timeArray[j + i*pointsPerAcq] = (horizOffsetArray[i]
                + trigtimeArray[i] + 1e9*j*horizInterval);
        }
}
double dx = timeArray[1] - timeArray[0]; //Integration timestep
int pointsPerWindow = (int)(p.getWindow()/dx);
```

```
        vector<double*> Acqs;
        vector<double*> Times;
        vector<bool> goodHits;
        vector<double> spectrumTime;
        vector<double> spectrumADCCounts;

        LeCroyData lcd = LeCroyData(fileName, vertUnit, horizUnit,
            timestamp, commType, verticalGain, verticalOffset,
            horizUncertainty, trigtimeArray, horizOffsetArray, waveArray,
            timeArray, pointsPerAcq, subArrayCount, dx, pointsPerWindow,
            Acqs, Times, goodHits, spectrumTime, spectrumADCCounts);

        //Clean up
        delete fileBuffer;
        return lcd;
}
vector<LeCroyData> Load(int n, string path, Parameters P) {
        vector<LeCroyData> run;
        for(int i=0; i<n; i++) {
                cout<<"Please input the filename ("<<i+1<<"/"<<n<<"):"<<endl;
                string filename;
                getline(cin, filename, '\n');
                LeCroyData lcd = readFile(path+"/"+filename, P);
                run.push_back(lcd);
        }
        return run;
}
```

Analysis.cpp

```
#include "LeCroy.h"

double* getRawAcqWave(int segment, LeCroyData lcd) {
        //Getter method for an individual waveform acquisition,
            indexed by the int segment
        int n = lcd.getPointsPerAcq();
        double* waveform = new double[n];
        for(int i = 0; i<n; i++) { //For each point in the acquisition
                //Fetch the point from the specified segment
                waveform[i] = lcd.getWaveArray()[segment*n + i];
        }
        return waveform;
```

```
}
double* getRawAcqTime(int segment, LeCroyData lcd) {
        //Getter method for the timing data for the individual waveform
            acquisition indexed by the int segment
        int n = lcd.getPointsPerAcq();
        double* segmentTimes = new double[n];
        for(int i = 0; i<n; i++) { //For each point in the acquisition
        //Fetch the point from the specified segment
                segmentTimes[i] = lcd.getTimeArray()[segment*n + i]-
                    lcd.getTrigTimeArray()[segment];
        }
        return segmentTimes;
}
double meanEstimateBaseline(double* data, double* time, LeCroyData &lcd,
    Parameters p) {
        //Uses the voltage recorded up to the trigger event to
            estimate the baseline
        //Labels value for discarding if the standard deviation of voltage
            region in both pre- and post- trigger is higher than sigLimit
        int startPoints = 0;
        int n = lcd.getPointsPerAcq();
        bool check = false; //Assume acquisition is bad
        //Identify the number of points before the trigger
        while(time[startPoints+1] < 0) {
                startPoints++;
        }
        //To avoid the triggering signal skewing the value of the baseline
        startPoints = (startPoints*8)/10;
        double startRegion[startPoints];
        //Collect the voltage values before the trigger in an array
        for(int i = 0; i<startPoints; i++) {
                startRegion[i] = data[i];
        }
        //And evaluate the mean
        double baseLine = TMath::Mean<double>(startPoints, startRegion);
        //If first baseline test is FAILED
        if(TMath::StdDev<double>(startPoints, startRegion) > p.getMaxSigma()
            ) {
                //Try the latter half of the acquisition to estimate baseline,
                    in case pulse is in startRegion
                double secondRegion[startPoints];
```

```
                //Collect voltage values from end of acquisition into array
                for(int i = 0; i<startPoints; i++) {
                        secondRegion[i] = data[n-i];
                }
                //recalculate mean for new region
                baseLine = TMath::Mean<double>(startPoints, secondRegion);
                //If second baseline test is PASSED
                if(TMath::StdDev<double>(startPoints, secondRegion) <
                    p.getMaxSigma()) {
                    //search for a negative value of pd, indicative of signal
                        for (int i=0; i<n; i++) {
                                if (data[i] < 0) {
                                    //if found, then likely that a single
                                    pulse is present
                                        check = true;
                                        break;
                                }
                        }
                }
        }
        else { //If first baseline test is PASSED
        //search for a negative value of pd, indicative of a signal
                for (int i=0; i<n; i++) {
                        if (data[i] < 0) {
                        //if found, then it is likely that a single pulse
                            is present
                                check = true;
                                break;
                        }
                }
        }
        //add result of baseline & signal testing into a vector of checks
        if (check) {
                lcd.appendGoodHits(true);
        }
        else {
                lcd.appendGoodHits(false);
        }
        return baseLine;
}
void baselineSubtraction(LeCroyData &lcd, Parameters P) {
```

```
        lcd.getAcqs().clear(); //clear any previous baseline subtraction
        lcd.getGoodHits().clear();
        int n = lcd.getSubAcqCount();
        int p = lcd.getPointsPerAcq(); //changed from PointsPerWindow
        for(int i = 0; i<n; i++) { //For each acquisition
                double* data = getRawAcqWave(i, lcd);
                double* time = getRawAcqTime(i, lcd);
                //Calculate the baseline for the acquisiton
                double baseline = meanEstimateBaseline(data, time, lcd, P);
                //Make a record of the acquisitions:
                double* acq = new double[p];
                for(int j = 0; j<p; j++) {
                //Store a copy of the acquisition after baseline subtraction
                        acq[j] = baseline - data[j];
                }
                lcd.appendAcqs(acq);
                lcd.appendTimes(time);
        }
}
double compositeIntegrate(double* Acq, int pointsPerWindow, int pointsPerAcq,
    double dx) {
        //Locate pulse by finding the highest pd
        int max = 0;
        double minValue = -DBL_MAX;
        for (int i=0; i<pointsPerAcq; i++) {
                if (Acq[i] > minValue) {
                        minValue = Acq[i];
                        max = i;
                }
        }
        //start integration 20ns before peak of pulse
        int windowStart = max - (20/dx);
        //An n-point composite Newton-Cotes formula
        //Pretty good for nPoints > 8
        double sum = 0;
        for(int i = 0; i<pointsPerWindow; i++) {
                if(i == 0 || i == pointsPerWindow-1) {
                        sum += 17*Acq[windowStart+i]/48.;
                }
                else if(i == 1 || i == pointsPerWindow-2) {
                        sum += 59*Acq[windowStart+i]/48.;
```

```
                }
                else if(i == 2 || i == pointsPerWindow-3) {
                        sum += 43*Acq[windowStart+i]/48.;
                }
                else if(i == 3 || i == pointsPerWindow-4) {
                        sum += 49*Acq[windowStart+i]/48.;
                }
                else {
                        sum+=Acq[windowStart+i];
                }
        }
        return sum*dx;
}
void buildSpectrum(LeCroyData &lcd) {
        lcd.getSpectrumADCCounts().clear(); //Clear any previous spectra
        lcd.getSpectrumTime().clear();
        int n = lcd.getSubAcqCount();
        double step = lcd.getDx();
        //Calculate a point in the spectrum from every acquisition
        for(int i = 0; i<n; i++) {
                double integral = compositeIntegrate(lcd.getAcqs()[i],
                    lcd.getPointsPerWindow(), lcd.getPointsPerAcq(), step);
                lcd.appendSpectrumADCCounts(integral);
                lcd.appendSpectrumTime(lcd.getTrigTimeArray()[i]);
        }
}
vector<LeCroyData> Analysis(int n, string path, Parameters P) {
        vector<LeCroyData> run = Load(n, path, P);
        cout<<"Analysing trace file(s)..."<<endl;
        int count = 0;
        for(int i=0; i<run.size(); i++) {
                baselineSubtraction(run[i], P);
                buildSpectrum(run[i]);
                count += run[i].getSubAcqCount();
        }
        cout<<"The trace file(s) contained a total of "<<count<<"
            acquisitions."<<endl;
        return run;
}
void Write(vector<LeCroyData> run, string name, string dest) {
        dest = dest+"/"+name+".txt";
```

```
        cout<<dest<<endl;
        const char* dest_c = dest.c_str();
        ofstream file;
        file.open (dest_c);
        file << "Spectrum Time          Spectrum ADC Counts
            Label\n";
        for (int i=0; i<run.size(); i++) {
                for (int j=0; j<run[i].getSpectrumADCCounts().size(); j++) {
                        string time = boost::lexical_cast<string>(
                            run[i].getSpectrumTime()[j]);
                        string count = boost::lexical_cast<string>(
                            run[i].getSpectrumADCCounts()[j]);
                        string label;
                        if (run[i].getGoodHits()[j]) {
                                label = "Good";
                        }
                        else {
                                label = "Bad";
                        }
                        string entry = time+"   "+count+"       "+label+"\n";
                        file << entry;
                }
        }
        file.close();
}
vector<Hit> Match(vector<double> leCroyEnergies, vector<double> leCroyTime,
    vector<bool> goodHits, vector<Tracker> tracker) {
        cout<<"Filtering tracker data..."<<endl;
        //filter tracker data, taking only good hits
        vector<Tracker> tracker_f;
        for (int i=0; i<tracker.size(); i++) {
                if (tracker[i].getGoodHit()) {
                        tracker_f.push_back(tracker[i]);
                }
        }
        cout<<"Filtered tracker data contains "<<tracker_f.size()<<"
            events."<<endl;
        cout<<"Filtering leCroy data..."<<endl;
        //filter leCroy data, taking only good acquisitions
        vector<double> energies_f;
        vector<double> leCroyTime_f;
```

```
        for (int i=0; i<leCroyEnergies.size(); i++) {
                if (goodHits[i]) {
                        energies_f.push_back(leCroyEnergies[i]);
                        leCroyTime_f.push_back(leCroyTime[i]);
                }
        }
        cout<<"Filtered leCroy data contains "<<energies_f.size()<<"
            events."<<endl;
        cout<<"Matching events..."<<endl;
        vector<Hit> hits_f;
        double agreement = 500.0; //initial tolerance for matching events (ns)
        for (int i=0; i<leCroyTime_f.size(); i++) {
                for (int j=0; j<tracker_f.size(); j++) {
                    //clocks gradually shift further out of sync
                        if (abs(leCroyTime_f[i]-tracker_f[j].getTimestamp()) <
                            agreement + (3.42E-6*tracker_f[j].getTimestamp()))
                            {
                                Hit hit = Hit(tracker_f[j].getTimestamp(),
                                    tracker_f[j].getCoords()[0],
                                    tracker_f[j].getCoords()[1],
                                    energies_f[i]);
                                hits_f.push_back(hit);
                                //cout<<"Match at "<<leCroyTime_f[i]<<","<<
                                    tracker_f[j].getTimestamp()<<endl;
                                break;
                        }
                }
        }
        cout<<"Matched "<<hits_f.size()<<" events from filtered data."<<endl;
        return hits_f;
}
void writeMatch(vector<Hit> run, string name, string dest) {
        dest = dest+"/"+name+".txt";
        cout<<dest<<endl;
        const char* dest_c = dest.c_str();
        ofstream file;
        file.open (dest_c);
        file << "Time            X               Y               Energy\n";
        for (int i=0; i<run.size(); i++) {
                string time = boost::lexical_cast<string>(
                    run[i].getTimestamp());
```

```
                string x = boost::lexical_cast<string>(run[i].getX());
                string y = boost::lexical_cast<string>(run[i].getY());
                string energy = boost::lexical_cast<string>(
                    run[i].getEnergy());
                string entry = time+"   "+x+"   "+y+"   "+energy+"\n";
                file << entry;
        }
        file.close();
}
void Histogram(vector<LeCroyData> run, Parameters P, string name, string dest,
    TApplication* app) {
        string title = name+";ADC Counts (mV ns);Frequency";
        const char* name_c = name.c_str();
        const char* title_c = title.c_str();
        TH1D* hist = new TH1D("ADC Counts", title_c, P.getBins(), P.getXlow(),
            P.getXhigh());
        for(int i=0; i<run.size(); i++) {
                for (int j=0; j<run[i].getSpectrumADCCounts().size(); j++) {
                        //Removing this check will plot bad acqusitions too
                        if (run[i].getGoodHits()[j]) {
                                hist->Fill(run[i].getSpectrumADCCounts()[j]);
                        }
                }
        }
        //find the bin with the largest number of counts
        int maxLeCroyBin = hist->GetMaximumBin();
        //convert bin number into ADC count
        double conversion = (double) (P.getXhigh()-P.getXlow())/P.getBins();
        cout<<"The peak energy was: "<<maxLeCroyBin*conversion<<" mV ns"<<
            endl;
        const char* dest_c;
        dest = dest+"/"+name+".pdf";
        dest_c = dest.c_str();
        TCanvas* c1 = new TCanvas(name_c, name_c, 1920, 1080);
        hist->Draw();
        c1->Update();
        c1->Print(dest_c);
        app->Run(hist);
        delete hist;
        delete c1;
}
```

```
void timeHistogram(vector<LeCroyData> run, Parameters P, string name,
    string dest, TApplication* app) {
        string title = name+";Time Interval (ns);Frequency";
        const char* name_c = name.c_str();
        const char* title_c = title.c_str();
        TH1D* hist = new TH1D("Time Interval Between Pulses", title_c,
            P.getBins(), P.getXlow(), P.getXhigh());
        for(int i=0; i<run.size(); i++) {
                for (int j=0; j<run[i].getSpectrumTime().size(); j++) {
                        hist->Fill(run[i].getSpectrumTime()[j+1]-run[i].
                            getSpectrumTime()[j]);
                }
        }
        const char* dest_c;
        dest = dest+"/"+name+".pdf";
        dest_c = dest.c_str();
        TCanvas* c1 = new TCanvas(name_c, name_c, 1920, 1080);
        hist->Draw();
        c1->Update();
        c1->Print(dest_c);
        app->Run(hist);
        delete hist;
        delete c1;
}
void plotRawWaveform(LeCroyData lcd, int index, string name, string dest,
    TApplication* app) {
        int p = lcd.getPointsPerAcq();
        double* ADC = getRawAcqWave(index, lcd);
        double* time = getRawAcqTime(index, lcd);
        TGraph* graph = new TGraph(p, time, ADC);
        graph->GetXaxis()->SetTitle("Time (ns)");
        graph->GetYaxis()->SetTitle("Potential Difference (mV)");
        TCanvas* c1 = new TCanvas("Waveform", "Waveform", 1920, 1080);
        dest = dest+"/"+name+".pdf";
        const char* dest_c = dest.c_str();
        const char* name_c = name.c_str();
        graph->SetNameTitle(name_c, name_c);
        graph->Draw();
        c1->Update();
        c1->Print(dest_c);
        app->Run(graph);
```

```
        delete graph;
        delete c1;
}
void plotSubtractedWaveform(LeCroyData lcd, int index, string name,
    string dest, TApplication* app) {
        int p = lcd.getPointsPerAcq();
        double* ADC = lcd.getAcqs()[index];
        double* time = lcd.getTimes()[index];
        TGraph* graph = new TGraph(p, time, ADC);
        graph->GetXaxis()->SetTitle("Time (ns)");
        graph->GetYaxis()->SetTitle("Potential Difference (mV)");
        TCanvas* c1 = new TCanvas("Waveform", "Waveform", 1920, 1080);
        dest = dest+"/"+name+".pdf";
        const char* dest_c = dest.c_str();
        const char* name_c = name.c_str();
        graph->SetNameTitle(name_c, name_c);
        graph->Draw();
        c1->Update();
        c1->Print(dest_c);
        app->Run(graph);
        delete graph;
        delete c1;
}
void plotBoth(vector<LeCroyData> run, Caen caen, Parameters P, string name,
    string dest, TApplication* app) {
        string title = name+";ADC Counts (mV ns);Frequency";
        const char* name_c = name.c_str();
        const char* title_c = title.c_str();
        const char* dest_c;
        dest = dest+"/"+name+".pdf";
        dest_c = dest.c_str();
        //load LeCroy data into histogram
        TH1D* leCroy = new TH1D("LeCroy Scope", title_c, P.getBins(),
            P.getXlow(), P.getXhigh());
        for(int i=0; i<run.size(); i++) {
                for (int j=0; j<run[i].getSpectrumADCCounts().size(); j++) {
                        //Removing this check will plot bad acqusitions too
                        if (run[i].getGoodHits()[j]) {
                                leCroy->Fill(run[i].getSpectrumADCCounts()[j]
                                    );
                        }
```

```
        }
}
//scale Caen data to match LeCroy data
double maxCaenBin;
double maxCaenFreq = DBL_MIN;
for (int i=0; i<caen.getBins().size(); i++) {
        if (caen.getCounts()[i] > maxCaenFreq) {
            //find the bin with the largest number of counts
                maxCaenFreq = caen.getCounts()[i];
                maxCaenBin = i;
        }
}
//find the bin with the largest number of counts
int maxLeCroyBin = leCroy->GetMaximumBin();
double maxLeCroyFreq = leCroy->GetBinContent(maxLeCroyBin);
//convert bin number into ADC count
double conversion = (double) (P.getXhigh()-P.getXlow())/P.getBins();
//find approximate x and y scaling factors
double xChange = maxCaenBin/(maxLeCroyBin*conversion);
double yChange = maxCaenFreq/maxLeCroyFreq;
//Compare number of events in peaks of histograms to find
    better y-scale
double caenPeakCounts = 0.0;
double leCroyPeakCounts = 0.0;
//the scale between no. of LeCroy bins vs. Caen bins
int factor = (P.getXhigh()-P.getXlow())/P.getBins();
if (factor == conversion) { //check if whole number
        int peakWidth = 320; //width of peak in mVns
        int halfRange = peakWidth/(2*factor);
        //count events +- 320 mVns of peak
        for (int i=-halfRange; i<halfRange+1; i++) {
                leCroyPeakCounts += leCroy->
                    GetBinContent(maxLeCroyBin+i);
                //otherwise for i=range-1, the caen data goes on for
                    3 mVns too many
                if (i == halfRange) {
                        caenPeakCounts += caen.getCounts()
                            [maxCaenBin+(factor*i)];
                }
                else {
                        for (int j=0; j<factor; j++) {
```

```
                                        caenPeakCounts += caen.getCounts()
                                            [maxCaenBin+(factor*i)+j];
                                }
                        }
                }
        }
        //Do not know why we need to multiply by factor to get a good result
        double yChangePeak = caenPeakCounts/(leCroyPeakCounts*factor);
//Check for noisy peaks by checking if y-scale factors are within 20%
    of each other
        if (TMath::Abs(yChangePeak-yChange)/yChange > 0.2) {
                cout<<"The approximate y-scale factor was used: check plot
                    for noisy peak."<<endl;
        }
        else {
                yChange = yChangePeak;
                cout<<"The peak event counting y-scale factor was used."<<
                    endl;
        }
        cout<<"Would you like to find the x-scale by minimising the
        Chi-Square? (Takes a few minutes) (Y/N)"<<endl;
        string o;
        getline(cin, o, '\n');
        if (o=="Y" || o=="Yes" || o=="y") {
                //Test x and y scale factors within +/- 1.5% in steps of 0.05%
                double increment = 0.0005;
                int range = 30;
                vector<double> xFactors;
                vector<double> yFactors;
                for (int i=0; i<range; i++){
                        xFactors.push_back((1-(((range/2)-i)*increment))*
                            xChange);
                        yFactors.push_back((1-(((range/2)-i)*increment))*
                            yChange);
                }
                //For each factor, create a scaled histogram of Caen data &
                    evaluate chi-square
                vector<double> xChiSquares;
                vector<TH1D*> xHistograms;
                double bestXChiSquare = DBL_MAX;
                double newXChange;
```

```
//Create histograms for Caen data
for (int i=0; i<xFactors.size(); i++) {
        string xID = to_string(xFactors[i]);
        const char* xID_c = xID.c_str();
        //Caen data in the same number of bins as leCroy
        TH1D* xCaenHist = new TH1D(xID_c, title_c,
        P.getBins(), P.getXlow(), P.getXhigh());
        xHistograms.push_back(xCaenHist);
}
//Fill histograms
cout<<"Testing x scale factors..."<<endl;
cout<<"Filling Caen histograms..."<<endl;
for (int k=0; k<caen.getBins().size(); k++) {
        for (int j=0; j<caen.getCounts()[k]; j++) {
                for (int i=0; i<xFactors.size(); i++) {
                        xHistograms[i]->Fill(
                            caen.getBins()[k]/xFactors[i]);
                }
        }
}
//Since we have rebinned Caen data into fewer bins,
    find new approximate y scale factor
//Should not matter which histogram we use
int maxCaenBin2 = xHistograms[0]->GetMaximumBin();
double maxCaenFreq2 = xHistograms[0]->GetBinContent(
    maxCaenBin2);
double yChange2 = maxCaenFreq2/maxLeCroyFreq;
//Suppress information output from ROOT
gErrorIgnoreLevel = kError;
//Test x chi-square
for (int i=0; i<xFactors.size(); i++) {
        xHistograms[i]->Scale(1/yChange2);
        //Evaluate chi-square and keep track of best value
        double chisquare = xHistograms[i]->Chi2Test(
            leCroy, "UU, CHI2/NDF");
        if (chisquare < bestXChiSquare) {
                bestXChiSquare = chisquare;
                newXChange = xFactors[i];
        }
        cout<<"The Chi-Square between the LeCroy and Caen
            histograms for an x-scale factor "<<
```

```
                        xFactors[i]<<" is: "<<chisquare<<endl;
                    xChiSquares.push_back(chisquare);
                    delete xHistograms[i];
                }
                cout<<"The difference between the approximate x-scale factor
                    and the Chi-Square x-scale factor is: "<<
                    xChange-newXChange<<endl;
                xChange = newXChange;
                //Restore information output from ROOT
                gErrorIgnoreLevel = kPrint;
        }
        cout<<endl;
        cout<<"The peak (LeCroy) energy was: "<<maxLeCroyBin*conversion<<"
            mV ns"<<endl;
        cout<<"The best x-scale factor was found to be: "<<xChange<<endl;
        cout<<"The best y-scale factor was found to be: "<<yChange<<endl;
        vector<double> x; //scale original Caen data with best factors
        vector<double> y;
        for (int i=0; i<caen.getBins().size(); i++) {
                x.push_back(caen.getBins()[i]/xChange);
                y.push_back(caen.getCounts()[i]/yChange);
        }
        TGraph* graph = new TGraph(x.size(), x.data(), y.data());
        TCanvas* c1 = new TCanvas(name_c, name_c, 1920, 1080);
        graph->GetXaxis()->SetLimits(0, P.getXhigh());
        graph->GetXaxis()->SetTitle("ADC Counts (mV ns)");
        graph->GetYaxis()->SetTitle("Frequency");
        graph->SetNameTitle(name_c,name_c);
        graph->Draw();
        leCroy->Draw("same");
        c1->Update();
        c1->Print(dest_c);
        app->Run(leCroy);
        delete leCroy;
        delete c1;
}
void plot3D(vector<Hit> hits, int bins, string name, string dest,
    TApplication* app) {
        //cout<<"Would you like to write average energy histogram data to
            file? (Y/N)"<<endl;
        string o = "N";
```

```
//getline(cin, o, '\n');
string title = name+";x (mm);y (mm);Range (mm)";
string title1 = name+";x (mm);y (mm);Frequency";
string title2 = name+";x (mm);Energy (mV ns);Frequency";
string title3 = name+";y (mm);Energy (mV ns);Frequency";
const char* name_c = name.c_str();
const char* title_c = title.c_str();
const char* title1_c = title1.c_str();
const char* title2_c = title2.c_str();
const char* title3_c = title3.c_str();
TH2D* hist = new TH2D("3D Dose Deposition", title_c, bins, -20.0,
    20.0, bins, -20.0, 20.0);
TH2D* hist1 = new TH2D("X-Y Dose Deposition (Full)", title1_c, bins,
    -20.0, 20.0, bins, -20.0, 20.0);
TH2D* hist2 = new TH2D("X-Y Dose Deposition (Stripped)", title1_c,
    bins, -20.0, 20.0, bins, -20.0, 20.0);
TH2D* hist3 = new TH2D("X-E Dose Deposition", title2_c, bins, -20.0,
    20.0, bins, 0, 2200.0);
TH2D* hist4 = new TH2D("Y-E Dose Deposition", title3_c, bins, -20.0,
    20.0, bins, 0, 2200.0);
TCanvas* c1 = new TCanvas(name_c, name_c, 1080, 1080);
for(int i=0; i<hits.size(); i++) {
//deliver global bin number of added entry
        int b = hist1->Fill(hits[i].getX(),hits[i].getY());
        hist3->Fill(hits[i].getX(),hits[i].getEnergy());
        hist4->Fill(hits[i].getY(),hits[i].getEnergy());
        //add up energies of all events in each bin
        hist->SetBinContent(b, hist->GetBinContent(b)+hits[i].
            getEnergy());
}
//Calculate energy to range conversion
double alpha = 0.0194; //power law of form range = alpha*energy^p
double p = 1.795;          //values found from Geant4 simulation
double peakEnergy = 1816.0; //found from energy spectrum
double beamEnergy = 36.0;       //beam energy in MeV
double conversion = beamEnergy/peakEnergy;
//Transform 3D XY to XYRange histogram
int minCount = 20;
for (int i=0; i<bins; i++) { //iterate over all bins
        for (int j=0; j<bins; j++) {
        //return global bin number (+1 ignores underflow bin)
```

```
                int binNum = hist->GetBin(i+1,j+1);
                //find the number of counts
                double count = hist1->GetBinContent(binNum);
                if (count > minCount) {
                        hist->SetBinContent(binNum,
                        //divide energy by counts to find average,
                            then convert to range
                        alpha*(TMath::Power(((hist->GetBinContent(
                            binNum)/count)*conversion),p)));
                        hist2->SetBinContent(binNum,
                            hist1->GetBinContent(binNum));
                }
                else {
                //Set bins with insufficient events to 0
                        hist->SetBinContent(binNum, 0);
                }
        }
}
//Write data of XYRange histogram to file
if (o=="Y" || o=="Yes" || o=="y") {
        const char* dest_c;
        dest = dest+"/"+name+".txt";
        dest_c = dest.c_str();
        ofstream file;
        file.open (dest_c);
        file << "X               Y               Range\n";
        for (int i=0; i<bins; i++) { //iterate over all bins
                for (int j=0; j<bins; j++) {
                //return global bin number (+1 ignores underflow bin)
                        int binNum = hist->GetBin(i+1,j+1);
                        string x = boost::lexical_cast<string>(
                            hist->GetXaxis()->GetBinLowEdge(i+1)+
                            hist->GetXaxis()->GetBinWidth(i+1)/2);
                        string y = boost::lexical_cast<string>(
                            hist->GetYaxis()->GetBinLowEdge(j+1)+
                            hist->GetYaxis()->GetBinWidth(j+1)/2);
                        string energy = boost::lexical_cast<string>(
                            hist->GetBinContent(binNum));
                        string entry = x+"      "+y+"   "+energy+"\n";
                        file << entry;
                }
```

```
        }
        file.close();
}
//Draw 4D XYRange Histogram
hist->SetStats(kFALSE); //Hide legends
hist2->SetStats(kFALSE);
//stops axis labels overlapping with scale
hist->GetXaxis()->SetTitleOffset(1.5);
hist->GetYaxis()->SetTitleOffset(1.7);
hist->GetZaxis()->SetTitleOffset(1.2);
hist->SetContour(99);
//Make surface plot invisible in 3D space
hist2->SetLineColorAlpha(22,0.);
hist2->SetContour(60);
//Change 0 level to white instead of blue
hist2->SetContourLevel(0, 0.1);
hist->Draw("LEGO2Z0"); //run40_2mmEPCol_40pA_4mmPMMA.trc
c1->Update();
app->Run(hist);
//Draw 2D histogram onto a new pad and then overlay: semi-hacky way
    to make it work
//Before Using File->Quit ROOT to add the 2D histogram, make sure you
    have completed any adjustments to 3D histogram, otherwise colour
    will change!
TPad* pad2 = new TPad("pad2","",0,0,1,1);
int ci = 1181;
TColor* color = new TColor(ci, 0.52, 0.76, 0.64, " ", 0);
pad2->SetFillColor(ci);
pad2->SetFillStyle(4000);
pad2->SetBorderMode(0);
pad2->SetBorderSize(2);
pad2->SetFrameFillColor(0);
pad2->SetFrameBorderMode(0);
pad2->Draw();
pad2->cd();
gStyle->SetPalette(kRainBow);
hist2->Draw("SURF3 FB BB A");
c1->Update();
app->Run(hist);
delete color;
delete pad2;
```

```
//Draw Stripped 2D XY Histogram
hist2->GetXaxis()->SetTitleOffset(1.4);
hist2->GetYaxis()->SetTitleOffset(1.4);
hist2->GetZaxis()->SetTitleOffset(1.4);
hist2->SetContour(60);
hist2->Draw("CONT0, COLZ");
c1->Update();
app->Run(hist2);
//Draw Full 2D XY Histogram
gStyle->SetPalette(kBird);
hist1->GetXaxis()->SetTitleOffset(1.4);
hist1->GetYaxis()->SetTitleOffset(1.4);
hist1->GetZaxis()->SetTitleOffset(1.4);
hist1->SetContour(60);
hist1->Draw("CONT0, COLZ");
c1->Update();
app->Run(hist1);
//Draw Full 3D XY Histogram
hist1->SetContour(99);
hist1->Draw("LEGO2Z 0");
c1->Update();
app->Run(hist1);
//Draw 3D XE Histogram
hist3->GetXaxis()->SetTitleOffset(1.4);
hist3->GetYaxis()->SetTitleOffset(1.4);
hist3->GetZaxis()->SetTitleOffset(1.4);
hist3->SetContour(99);
hist3->Draw("LEGO2Z 0");
c1->Update();
app->Run(hist3);
//Draw 3D YE Histogram
hist4->GetXaxis()->SetTitleOffset(1.4);
hist4->GetYaxis()->SetTitleOffset(1.4);
hist4->GetZaxis()->SetTitleOffset(1.4);
hist4->SetContour(99);
hist4->Draw("LEGO2Z 0");
c1->Update();
app->Run(hist4);
delete hist;
delete hist1;
delete hist2;
```

```
        delete hist3;
        delete hist4;
        delete c1;
}
```

Main.cpp

```
#include "LeCroy.h"
#include "Caen.h"
#include "Tracker.h"

int inputInt() {
        int n;
        cin>>n;
        cin.ignore();
        return n;
}
double inputDouble() {
        double d;
        cin>>d;
        cin.ignore();
        return d;
}
string inputString() {
        string s;
        getline(cin, s, '\n');
        return s;
}
void subMenu(TApplication* app, Parameters P) {
        cout<<"What would you like to do?"<<endl;
        cout<<endl;
        cout<<"-> Enter 0 to return to previous menu."<<endl;
        cout<<"-> Enter 1 to set maximum baseline sigma. (Current value: "
            <<P.getMaxSigma()<<")"<<endl;
        cout<<"-> Enter 2 to set integration window. (Current value: "
            <<P.getWindow()<<" ns)"<<endl;
        cout<<"-> Enter 3 to set number of histogram bins. (Current value: "
            <<P.getBins()<<")"<<endl;
        cout<<"-> Enter 4 to set histogram x-axis limits. (Current values: "
            <<P.getXlow()<<", "<<P.getXhigh()<<")"<<endl;
        int subOption = inputInt();
        if (subOption == 1) {
```

```
        cout<<"Set the maximum baseline sigma: (Current value: "<<
            P.getMaxSigma()<<")"<<endl;
        cout<<"Or enter 0 to return."<<endl;
        double s = inputDouble();
        if (s!=0) {
                P.setMaxSigma(s);
        }
        subMenu(app, P);
}
else if (subOption == 2) {
        cout<<"Set the integration window (ns): (Current value: "<<
            P.getWindow()<<" ns)"<<endl;
        cout<<"Or enter 0 to return."<<endl;
        double w = inputDouble();
        if (w!=0) {
                P.setWindow(w);
        }
        subMenu(app, P);
}
else if (subOption == 3) {
        cout<<"Set the number of histogram bins: (Current value: "<<
            P.getBins()<<")"<<endl;
        cout<<"Or enter 0 to return."<<endl;
        int b = inputInt();
        if (b!=0) {
                P.setBins(b);
        }
        subMenu(app, P);
}
else if (subOption == 4) {
        cout<<"Set the X-axis limits: (Current values: "<<P.getXlow()
            <<", "<<P.getXhigh()<<")"<<endl;
        cout<<"Or enter 0 to return."<<endl;
        cout<<"Set the upper x limit: "<<endl;
        int upper = inputInt();
        if (upper != 0) {
                cout<<"Set the lower x limit: "<<endl;
                int lower = inputInt();
                P.setXlow(lower);
                P.setXhigh(upper);
        }
```

```
                subMenu(app, P);
        }
        else {
                program(app, P);
        }
}
void program(TApplication* app, Parameters P) {
        cout<<"What would you like to do?"<<endl;
        cout<<endl;
        cout<<"-> Enter 0 to exit."<<endl;
        cout<<"-> Enter 1 to plot waveforms."<<endl;
        cout<<"-> Enter 2 to histogram pulse time intervals."<<endl;
        cout<<"-> Enter 3 to histogram ADC spectra."<<endl;
        cout<<"-> Enter 4 to set LeCroy baseline subtraction, integration
            or histogram parameters."<<endl;
        cout<<"-> Enter 5 to plot Caen digitiser data."<<endl;
        cout<<"-> Enter 6 to plot Caen digitiser data with LeCroy data."
           <<endl;
        cout<<"-> Enter 7 to plot PRaVDA tracker data."<<endl;
        cout<<"-> Enter 8 to match LeCroy and PRaVDA tracker data."<<endl;
        int option = inputInt();
        if (option == 1) {
                cout<<"-> Enter 0 to return to previous menu."<<endl;
                cout<<"-> Enter 1 to plot raw waveforms."<<endl;
                cout<<"-> Enter 2 to plot subtracted waveforms."<<endl;
                int option2 = inputInt();
                if (option2 == 0) {
                        program(app, P);
                }
                if (option2 == 1 || option2 == 2) {
                        cout<<"Please specify the full path of the .trc file:"
                            <<endl;
                        string path = inputString();
                        cout<<"Enter the index of the pulse you would like to
                        plot:"<<endl;
                        int index = inputInt();
                        cout<<"Specify the directory you wish to save the plot
                            to:"<<endl;
                        string dest = inputString();
                        cout<<"Please give the plot a name: "<<endl;
                        string name = inputString();
```

```
                cout<<"Working..."<<endl;
                LeCroyData lcd = readFile(path, P);
                if (option2 == 1) {
                        plotRawWaveform(lcd, index, name, dest, app);
                }
                if (option2 == 2) {
                        baselineSubtraction(lcd, P);
                        plotSubtractedWaveform(lcd, index, name, dest,
                            app);
                }
                cout<<"Analysis complete."<<endl;
                cout<<"Would you like to do something else? (Y/N)"
                    <<endl;
                string o = inputString();
                if (o=="Y" || o=="Yes" || o=="y") {
                        lcd.~LeCroyData();
                        program(app, P);
                }
                else {
                        cout<<"Goodbye."<<endl;
                        exit(0);
                }
        }
        else {
                cout<<"Goodbye."<<endl;
                exit(0);
        }
}
else if (option == 2 || option == 3) {
        cout<<"How many .trc files would you like to analyse?
            (Enter 0 to return to previous menu)"<<endl;
        int n = inputInt();
        if (n==0) {
                program(app, P);
        }
        else {
                cout<<"Please specify the directory containing your
                    .trc files:"<<endl;
                string path = inputString();
                vector<LeCroyData> run = Analysis(n, path, P);
                cout<<"Specify the directory you wish to save the
```

```
                    histogram to:"<<endl;
                string dest = inputString();
                cout<<"Please give the histogram a name: "<<endl;
                string name = inputString();
                cout<<"Save LeCroy data to disk? (Y/N)"<<endl;
                string g = inputString();
                if (g=="Y" || g=="Yes" || g=="y") {
                        cout<<"Writing file to disk..."<<endl;
                        Write(run, name, dest);
                        cout<<"Write complete."<<endl;
                }
                cout<<"Analysis complete."<<endl;
                if (option == 2) {
                        timeHistogram(run, P, name, dest, app);
                }
                if (option == 3) {
                        Histogram(run, P, name, dest, app);
                }
                cout<<"Would you like to do something else? (Y/N)"
                    <<endl;
                string o = inputString();
                if (o=="Y" || o=="Yes" || o=="y") {
                        for (int i=0; i<run.size(); i++) {
                                run[i].~LeCroyData();
                        }
                        program(app, P);
                }
                else {
                        cout<<"Goodbye."<<endl;
                        exit(0);
                }
        }
}
else if (option == 4) {
        subMenu(app, P);
}
else if (option == 5) {
        cout<<"Please specify the path of the digitiser data file:"
            <<endl;
        string path = inputString();
        Caen run = readCaen(path);
```

```
        cout<<"Specify the upper limit of the x-axis."<<endl;
        int xhigh = inputInt();
        cout<<"Specify the directory you wish to save the plot to:"
            <<endl;
        string dest = inputString();
        cout<<"Please give the plot a name: "<<endl;
        string name = inputString();
        cout<<"Working..."<<endl;
        //change to plotCaen for simple graphs
        plotCaenHist(run, name, dest, xhigh, app);
        cout<<"Analysis complete."<<endl;
        cout<<"Would you like to do something else? (Y/N)"
            <<endl;
        string o = inputString();
        if (o=="Y" || o=="Yes" || o=="y") {
                run.~Caen();
                program(app, P);
        }
        else {
                cout<<"Goodbye."<<endl;
                exit(0);
        }
}
else if (option == 6) {
        cout<<"Please specify the path of the digitiser data file:"
            <<endl;
        string caenPath = inputString();
        cout<<"Working..."<<endl;
        Caen caen = readCaen(caenPath);
        cout<<"How many .trc files would you like to analyse?
            (Enter 0 to return to previous menu)"<<endl;
        int n = inputInt();
        if (n==0) {
                program(app, P);
        }
        else {
                cout<<"Please specify the directory containing your
                    .trc files:"<<endl;
                string lecroyPath = inputString();
                vector<LeCroyData> run = Analysis(n, lecroyPath, P);
                cout<<"Specify the directory you wish to save the
```

```
                histogram to:"<<endl;
                string dest = inputString();
                cout<<"Please give the histogram a name: "<<endl;
                string name = inputString();
                cout<<"Save LeCroy data to disk? (Y/N)"<<endl;
                string g = inputString();
                if (g=="Y" || g=="Yes" || g=="y") {
                        cout<<"Writing file to disk..."<<endl;
                        Write(run, name, dest);
                        cout<<"Write complete."<<endl;
                }
                cout<<"Working..."<<endl;
                plotBoth(run, caen, P, name, dest, app);
                cout<<"Analysis complete."<<endl;
                cout<<"Would you like to do something else? (Y/N)"
                    <<endl;
                string o = inputString();
                if (o=="Y" || o=="Yes" || o=="y") {
                        for (int i=0; i<run.size(); i++) {
                                run[i].~LeCroyData();
                        }
                        caen.~Caen();
                        program(app, P);
                }
                else {
                        cout<<"Goodbye."<<endl;
                        exit(0);
                }
        }
}
else if (option == 7) {
        cout<<"Please specify the path of the PRaVDA .txt file:"
            <<endl;
        string path = inputString();
        cout<<"Specify the directory you wish to save the histogram
            to:"<<endl;
        string dest = inputString();
        cout<<"Please give the plot a name: "<<endl;
        string name = inputString();
        cout<<"Please specify the number of bins: "<<endl;
        int bins = inputInt();
```

```
        cout<<"Working..."<<endl;
        vector<Tracker> run = readTracker(path);
        histTracker(run, bins, name, dest, app);
        cout<<"Analysis complete."<<endl;
        cout<<"Would you like to do something else? (Y/N)"<<endl;
        string o = inputString();
        if (o=="Y" || o=="Yes" || o=="y") {
                for (int i=0; i<run.size(); i++) {
                        run[i].~Tracker();
                }
                program(app, P);
        }
        else {
                cout<<"Goodbye."<<endl;
                exit(0);
        }
}
else if (option == 8) {
        cout<<"Please specify the path of the PRaVDA .txt file:"
            <<endl;
        string trackerPath = inputString();
        cout<<"Working..."<<endl;
        vector<Tracker> tracker = readTracker(trackerPath);
        cout<<"Please specify the directory containing your .trc
            file:"<<endl;
        string lecroyPath = inputString();
        vector<LeCroyData> leCroy = Analysis(1, lecroyPath, P);
        vector<Hit> hits = Match(leCroy[0].getSpectrumADCCounts(),
            leCroy[0].getSpectrumTime(), leCroy[0].getGoodHits(),
                tracker);
        cout<<"Please give the file a name: "<<endl;
        string name = inputString();
        cout<<"Writing file to disk..."<<endl;
        writeMatch(hits, name, lecroyPath);
        cout<<"View 3D plots? (Y/N)"<<endl;
        string k = inputString();
        if (k=="Y" || k=="Yes" || k=="y") {
                cout<<"Please specify the number of bins: "<<endl;
                int bins = inputInt();
                cout<<"Working..."<<endl;
                plot3D(hits, bins, name, lecroyPath, app);
```

```
                }
                cout<<"Analysis complete."<<endl;
                cout<<"Would you like to do something else? (Y/N)"<<endl;
                string o = inputString();
                if (o=="Y" || o=="Yes" || o=="y") {
                        for (int i=0; i<tracker.size(); i++) {
                                tracker[i].~Tracker();
                        }
                        leCroy[0].~LeCroyData();
                        program(app, P);
                }
                else {
                        cout<<"Goodbye."<<endl;
                        exit(0);
                }
        }
        else {
                cout<<"Goodbye."<<endl;
                exit(0);
        }
}
void start() {
        cout<<endl;
        cout<<"<UCL PBT Data Analyser>"<<endl;
        cout<<endl;
        cout<<"Exit ROOT windows using File->Quit ROOT to allow program to
            continue."<<endl;
        cout<<endl;
        TApplication* app = new TApplication("App", 0, 0);
        Parameters P = Parameters(5.0, 150.0, 550, 0, 2200);
        program(app, P);
}
int main(int argc, char **argv) {
        start();
}
```