



UNIVERSIDAD SAN FRANCISCO DE QUITO

COLEGIO DE CIENCIAS E INGENIERÍA

Optimizing Analysis Software for Single Wire Proportional  
Counters

David Alberto Hervás Aguilar

César Zambrano, PhD., Director de Tesis

Beatrice Mandelli, PhD., Supervisora en CERN

Tesis de grado presentada como requisito  
para la obtención del título de Licenciado en Física

Quito, Abril 2015

Universidad San Francisco de Quito  
Colegio de Ciencias e Ingeniería

## HOJA DE APROBACIÓN DE TESIS

Optimizing Analysis Software for Single Wire Proportional Counters

**David Hervas Aguilar**

César Zambrano, PhD. ....  
Decano de la Escuela de Ciencias  
Colegio de Ciencias e Ingeniería  
Director de la Tesis

Edgar Carrera, PhD. ....  
Miembro del Comité de Tesis

Carlos Montúfar, PhD. ....  
Miembro del Comité de Tesis

Alessandro Veltri, PhD. ....  
Miembro del Comité de Tesis

Darío Niebieskikwiat, PhD. ....  
Director Dep. Física

Quito, Mayo 2015

## © DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído la Política de Propiedad Intelectual de la Universidad San Francisco de Quito y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo de investigación quedan sujetos a lo dispuesto en la Política.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo de investigación en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma:

-----

Nombre: David Alberto Hervas Aguilar

C.I.: 171217438-0

Fecha: 11 de mayo del 2015

## Acknowledgements

*First and foremost I would like to recognize the invaluable support from USFQ along all my studies and the connections this institution has helped me forge. This thesis would have not been possible without Edgar Carrera's continuous support to send annual groups to CERN and of course without CERN's magnificent Non-Member State Summer Student Programme. I would like to thank my supervisor Beatrice Mandelli along with Roberto Guida and Mar Capeans for the knowledge I gained, their advise, the final review of this manuscript and a great summer at CERN. Specifically, I would like to express my sincere thanks to Beatrice Mandelli for providing the data and original scripts that made this analysis possible and for her daily counsel and teachings. Finally, I am very grateful for César Zambrano's guidance throughout the elaboration of this manuscript and for his valuable time as my thesis director and as chair of the science branch of the Polytechnic School at USFQ.*

## DEDICATION

*I would like to dedicate my thesis to my Mom and Dad, for their unconditional support through my five years at USFQ, three career changes and two years at the University of Illinois.*

*For their love, their enthusiasm, their knowledge and their never ending care.*

*For the Sister who always made me laugh in times of need.*

## Resumen

Cuando radiación ionizante atraviesa las cámaras de gas en un detector de hilo único, los átomos de gas se separan en iones y electrones. Mediante un campo eléctrico fuerte y localizado cerca del hilo una avalancha de electrones se crea y puede ser capturada. La corriente que se produce en el hilo es proporcional a la energía original de la partícula detectada. No obstante, existen varios factores que puede contribuir al envejecimiento del detector. Estos se manifiestan en una pérdida de la ganancia causada por la deposición de contaminantes en el hilo. Este estudio consiste de técnicas de análisis de datos originales que se aplican para procesar grandes cantidades de datos producidos por dos detectores de hilo único corriendo simultáneamente. Varios factores de envejecimiento se analizan y se corrige los efectos causados por fluctuaciones ambientales. Una serie de scripts filtra datos, empareja datos y realiza correcciones y gráficos usando las extensas librerías de ROOT creadas en CERN.

## Abstract

When ionizing radiation passes through gas chambers in single wire detectors gas atoms separate into ions and electrons. By applying a strong localized electric field near the single wire an avalanche of electrons is created and it can be collected. The current produced in the wire is then proportional to the energy of the particle detected. Nevertheless, many factors can contribute to detector aging effects which are visible in a loss of gain caused by deposition of contaminants on the collecting wire. This study consists on novel data analysis techniques used to process large amounts of data produced by two simultaneously running single wire detectors. Aging effects are analyzed while environmental fluctuations are corrected for. A series of scripts carry out data filtering, data matching, corrections, and finally trend plotting by using ROOT's extensive libraries developed at CERN.



# Contents

<b>1. Introduction</b>	<b>11</b>
<b>2. Theoretical Framework</b>	<b>12</b>
2.1. Particle Detectors . . . . .	12
2.1.1. Types of Particle Detectors . . . . .	13
2.1.2. Ionizing Gas Chambers . . . . .	13
2.2. Single Wire Detectors . . . . .	15
2.2.1. Ionization of Gases . . . . .	15
2.2.2. Fill Gas Choice . . . . .	16
2.2.3. Electron Avalanche . . . . .	17
2.3. Single Wire Detector Aging . . . . .	18
2.3.1. Deposition on Anode . . . . .	20
2.4. Analysis Tools: ROOT . . . . .	21
<b>3. Experimental Setup and Measurements</b>	<b>22</b>
3.1. Timeline . . . . .	22
3.1.1. Single Wire 1 (SW1) . . . . .	22
3.1.2. Single Wire 2 (SW2) . . . . .	22
3.2. Radiation Source . . . . .	23
<b>4. Script File Map</b>	<b>23</b>
<b>5. Spectra Generation</b>	<b>25</b>
5.1. Introduction . . . . .	25
5.2. Scripts: Spectra: Histograms and Fits (SWspectra_v1.C) . . . . .	25
5.2.1. Grouping Data into Histograms . . . . .	26
5.2.2. SW Loops: Histogram Arrays . . . . .	26
5.2.3. Fits and Methodology . . . . .	26
5.2.4. Peak Range for Fits . . . . .	27
5.2.5. Month Loop . . . . .	28
5.3. Results and Analysis . . . . .	28
5.3.1. Spectra . . . . .	28
5.3.2. Fit Means . . . . .	30
5.3.3. Relative Gain . . . . .	33
5.3.4. Resolution . . . . .	35
5.4. Conclusions . . . . .	36

	10
<b>6. Environmental Effects</b>	<b>37</b>
6.1. Scripts: Environmental Corrector (chisqtest.C)	37
6.1.1. Data Matching	37
6.1.2. Methodology	38
6.1.3. Executing File	38
6.2. Results	39
6.3. Analysis	41
6.3.1. Townsend Coefficient	41
6.4. Conclusions	46
<b>7. Integrated Charge</b>	<b>46</b>
7.1. Introduction	46
7.2. Results and Analysis	46
7.3. Conclusions	49
<b>8. General Conclusions</b>	<b>50</b>
<b>9. Apendix</b>	<b>51</b>
9.1. AmbSelect.C	52
9.2. chisqtest.C (Lines 149-205, 287-367 [19])	59
9.3. FineoutFilter.C	69
9.4. IntegratedCurrent.C	71
9.5. PeakTend_v3.C	75
9.6. SWspectra_v1.C (Lines 69-239 [20])	88
<b>References</b>	<b>98</b>

## 1. Introduction

The work that follows consists of a the complex interaction of scripts written in order to analyze the aging effects of two simultaneously running single wire detectors. The bulk of the of the programing and analysis was conducted in the PH-DT-DI (Physics - Detector Technologies - Detector Infrastructure) department at The European Organization for Nuclear Research (CERN) in Switzerland and at Universidad San Francisco de Quito in Ecuador, to process a year of data collected by Dr. Beatrice Mandelli of the PH-DT-DI department at CERN. For this purpose ROOT, “an object oriented framework for large scale data analysis” developed at CERN [1], was used. The programing was conducted by myself under Dr. Mandelli’s supervision and was jumpstarted by some of her existing scripts. Concordantly, some useful segments of Dr. Mandelli’s code where included in mine (for more details refer to References). Subsequently, my thesis director in Ecuador, Dr. César Zambrano, guided me through the polishing of my analysis and the writing of this work. This thorough analysis consists of three main parts which complement each other. After a thorough theoretical framework of particle detectors and specifically Gaseous Ionizing detectors working in the Proportional Counter Regime (of which a Single Wire detector is the most basic example) the theory, script methodology, results and analysis of the Spectra Generation, Environmental Effects and Integrated Charge Sections are presented.

The Spectra Generation consists of grouping data into histograms which plot the the number particle detection events at each energy bin (also knows as spectra). The script gathers the data from different files created during the measurement period and generates 4 histograms for each day of the year. Using carefully designed fits to maximize fit accuracy and convergence many parameters are extracted and plotted against time but only the Gain, Resolution and Fit Means trends are analyzed. Here aging effects start to become clear.

The following section consists of the Environmental Corrections of the trends that were generated from the spectra. Specifically, pressure and temperature fluctuations can have a considerable effect on the detector due to the sensibility of its gas to these parameters. The script written for this Section reduces the dependence on these parameters and then supplementary scripts carry out a thorough analysis of the aging process while completely isolating temperature and pressure dependencies.

Finally, the concept of Integrated charge is introduced to briefly analyze the time trends produced in the Spectra Generation cection from a different angle. Instead of plotting the gain against time it is plotted against charge “accumulated” in the detector. Detecting particles generate small amounts of current that pass through the detector; therefore by integrating this current in time a total or “accumulated” charge in the detector is calculated.

## 2. Theoretical Framework

### 2.1. Particle Detectors

Particle detectors are the window into the world of the subatomic. They can range from a simple pocket dosimeter to the enormous ATLAS detector with 25 meters in diameter and 46 meters in length [2]. Particle detectors fundamentally rely on the interactions of particles with matter, hence the variety of particle detectors is as broad as these interactions and the types of particles. Effectively they detect, track, time, identify, and/or measure the energy, amongst many other functions, of incoming particles therefore becoming the most important tool in the field of High Energy Physics. It is important to point out some of the subtleties of particle detectors.

- **Detector Efficiency:** The number of particles detected divided by the total number of particles hitting the detector is known as the intrinsic efficiency of the detector (the absolute efficiency is given by number of particles detected divided by the total number particles produced by the source) . Detectors (Refer to Figure 2.1) have a given dead time in which no particles can be detected after any detection event which limits the efficiency of the detector[3].
- **Saturation:** A detector may become saturated if the rate of incoming particles surpasses the rate of particle detection.
- **Detector Resolution:** The resolution of a detector is given by the ability to discern two particles of different energies as such. While a detector with low resolution might see these two particles as having the same energy, one with a higher resolution will be able to distinguish them. If two peaks are separated by a greater distance than their full width at half maximum (FWHM) the the peaks are said to be resolved. Equation 2.1 gives the relative resolution at energy  $E$ , where  $\Delta E$  is the FWHM[3] .

$$Resolution = \frac{\Delta E}{E} \tag{2.1}$$

Depending on the functionality of the detector, particle detectors are typically characterized within the following groups.

- **Calorimeters:** “A calorimeter measures the energy a particle loses as it passes through. It is usually designed to stop entirely or “absorb” most of the particles coming from a collision, forcing them to deposit all of their energy within the detector.”[4]
- **Tracking Detectors:** “Tracking devices reveal the paths of electrically charged particles as they pass through and interact with suitable substances. Most tracking devices do not make particle tracks directly visible, but record tiny electrical signals that particles trigger as they move through the device.”[4]

- Triggers: “A trigger is a system that uses simple criteria to rapidly decide which events in a particle detector to keep when only a small fraction of the total can be recorded. Trigger systems are necessary due to real-world limitations in data storage capacity and rates.”[5]

When the energy of a particle is measured a spectrum is formed. The spectrum is a simple plot of the number of particles detected per specific energy. Radioactive isotopes may have many decay modes and emit radiation at different energies. These specific energies are called spectral lines. However, this nomenclature is actually misleading. All the detected particles from one spectral line actually do not fall within one exact energy, but within a Gaussian distributed range. The “line” refers to the mean of the distribution which ideally corresponds to the peak. Furthermore, the FWHM of the distribution is the width of the “line” and is related to the resolution of the detector by Equation 2.1. Typically a detector will not measure the energy of a particle directly, but it must be calculated depending on the output of the detector. For example the detector used in this experiment outputs ADC (analog to digital) counts generated by the internal circuitry of the detector. These counts then can be converted to energy by calibrating the detector using radiation sources of known energy.

### 2.1.1. Types of Particle Detectors

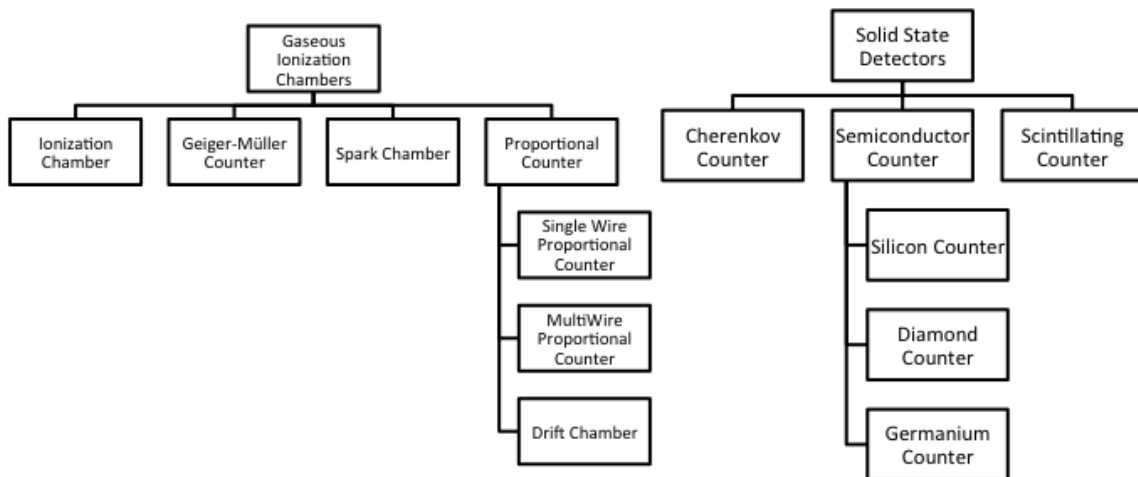


Figure 2.1: Hierarchy of particle detectors. Some gaseous detectors served as tracking systems (as well as calorimeters) in the past but were replaced by the significantly faster silicon detectors [6, 7].

### 2.1.2. Ionizing Gas Chambers

Gaseous ionizing detectors were the first electrical particle detectors and continue to be in widespread use today as radiation monitors. Their basic operation principle consists on the ionization of gas molecules and atoms. As ionizing radiation passes through this medium, ions

and electrons are formed. Meanwhile, an electric field is applied to collect the generated electrons; thus a current is produced in the anode of the detector [3]. Gases are used as ionization media because of the greater mobility of ions and electrons through them. A mixture of mostly noble gases is used in the chamber and the specifications of these mixtures naturally affect the current measured. A basic layout of a cylindrical gaseous ionizing detector is shown in Figure 2.2.

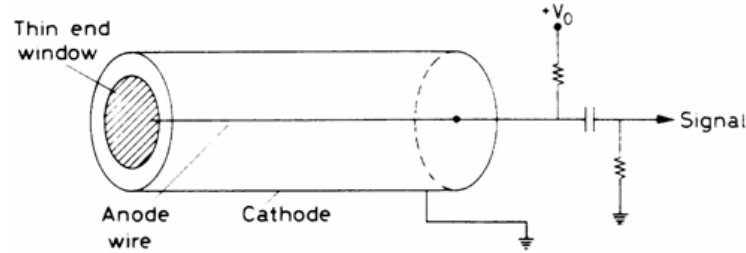


Figure 2.2: Cylindrical gaseous ionizing detector typical layout [3].

The simplicity of this design, the ease of operation, and low cost are the main reasons why these detectors are still in use today. The electric field produced by the wire on which a voltage  $V_0$  is applied is given by:

$$E = \frac{1}{r} \frac{V_0}{\ln\left(\frac{b}{a}\right)} \quad (2.2)$$

Where  $r$  is the distance from the wire,  $b$  the inside radius of the cylinder and  $a$  the radius of the central (anode) wire. This electric field is localized around the wire, which will become of importance in further discussions. As one can expect increasing the voltage applied to the detector will change the magnitude of the electric field and therefore have a major effect in the number of ions collected [3]. Many regimes can be identified as a function of increasing applied voltage. These are shown in Figure 2.3. This basic design coupled with all these working voltage regions encompass all the Gaseous Ionization Chamber detectors described in Figure 2.1.

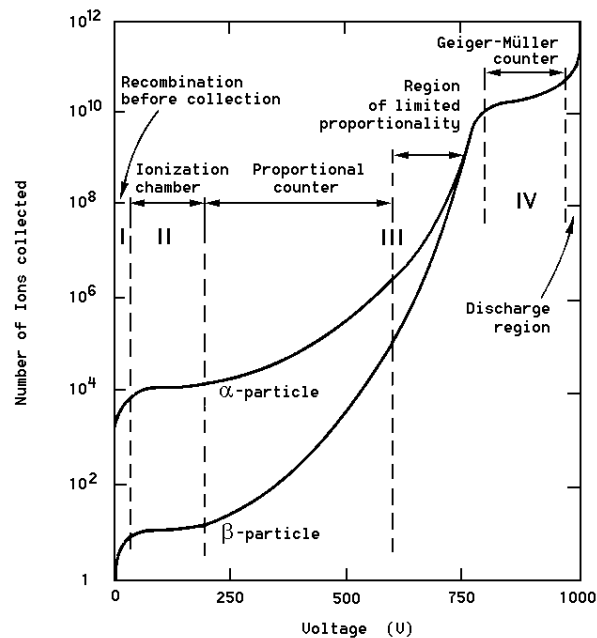


Figure 2.3: Different operating regimes of ionization gas chambers [3].

If the voltage is too low the electron-ion pairs will recombine because of their own attraction forces. By passing this limit the ionization chamber regime is reached. At this working point ion-electron pairs are collected with no intermediate effects. For this reason the generated signal will be very low and is only used with strong radiation sources. Increasing the voltage beyond this limit results in the proportional counter regime. This is the area of interest since this study centers data analysis of single wire gaseous ionizing detectors working in the proportional counter regime. At this voltage range the traveling electrons have enough energy to hit other gaseous particles in their path and cause further ionizations. This effect is referred to as an ionization avalanche. The total amount of ion-electron pairs produced is still proportional to the original ionization events caused by radiation; therefore, the current measured is proportional to the energy of the detected particle. The final voltage regions are the Geiger-Muller regime and then discharge region which corresponds to the working voltage of the Spark Chamber [3].

## 2.2. Single Wire Detectors

Single wire detectors are the simplest type of proportional counter. Having a design identical to that portrayed in Figure 2.2 they operate under the principles described below.

### 2.2.1. Ionization of Gases

Gas molecules and atoms can be readily ionized by ionizing radiation. There are many mechanisms through which this can occur. When a charged particle interacts with matter it can

lose energy in one of two reactions: excitation or ionization. Excitation occurs when a charged particle transfers its energy to an atom as in Equation 2.3.



Where  $p$  is the charged particle and  $X$  is the atom to be excited [3]. Although, no electrons are generated the excited atom may ionize other atoms in further reactions. Furthermore, direct ionizations may occur if the energy of the charged particle is high enough. Equation 2.4 demonstrates the ionization process.



This is known as a primary ionization. If the energy transferred to the emitted electron is high enough, it can participate in the ionization of further atoms, that is creating secondary ionizations. As noted for the excitation reaction, excited atoms may cause further ionizations too. The Penning Effect is an example of such a phenomenon. Certain atoms are not able to de-excite immediately (through emission of a photon) and can collide against other atoms thus starting an ionization reaction. A classical example is that of the interaction between different noble gases. Finally, the positive atoms formed in an ionization reaction can combine with neutral atoms of the same type and form a molecular ion releasing an electron in the process [3].

### 2.2.2. Fill Gas Choice

For the detector to work the electron ion pairs must remain intact till they are collected. Recombination and electron attachment come into play in this scenario. As explained in Section 2.1.2 while referencing Figure 2.3 if the working voltage of the detector is too low, recombination of the electron and ion will occur and a photon will be emitted.



Electron attachment is a similar process where the freed electron is captured by an atom with a high electron affinity [3].



It is evident that gases with low electron affinities must be used, such as the noble gases Ar, He and Ne which have negative electron affinities. Additionally it is important to monitor the levels of gases with high electron affinities. Molecular oxygen in the air is a particular contaminant that must be monitored at all time. Since  $O_2$  has a high electron affinity it can disrupt the operation of the single wire detector significantly[3].

The electric field applied in the detector forces ions and electrons to accelerate in opposite



directions. Due to collisions with other atoms this acceleration is capped and a maximum average velocity is achieved. The drift velocity is defined as the average speed that is attained in this process. Also note that particles have their own random thermal velocities given by

$$v = \sqrt{\frac{8k_B T}{\pi m}} \quad (2.7)$$

Where  $k_B$  is the Boltzman constant,  $T$  the temperature and  $m$  the mass of the particle. These velocities are much higher than the corresponding drift velocities of the electrons and ions. On the other hand the drift velocity  $u$  depends on the mobility of the charge  $\mu$  and the electric field strength  $E$  [3].

$$u = \mu E \quad (2.8)$$

In turn the mobility is related to the diffusion constant  $D$  by the Einstein relation in ideal gasses:

$$\frac{D}{\mu} = \frac{k_B T}{e} \quad (2.9)$$

Where  $e$  is the charge of the electron. And  $D$  is given by the expression in Equation 2.10.

$$D = \frac{2}{3\sqrt{\pi}} \frac{1}{p\sigma_0} \sqrt{\frac{(k_B T)^3}{m}} \quad (2.10)$$

Where  $p$  is the pressure and  $\sigma_0$  is the total cross Section for collision with a gas molecule [3]. Finally by joining Equations 2.8 - 2.10 a final expression for drift velocity is achieved.

$$u = \frac{2}{3\sqrt{\pi}} \frac{e}{p\sigma_0} \sqrt{\frac{k_B T}{m}} E \quad (2.11)$$

The drift velocities for electrons will be much higher than for positive ions since they are lighter. Also note the environmental dependancies (temperature and pressure) of Equation 2.11. Higher drift velocities are desired to avoid electron attachment and recombination. The addition of certain polyatomic gasses such as  $CO_2$ ,  $CH_4$ , or  $CF_4$  leads to larger electron drift velocities [8] and are therefore readily used in single wire detectors.

### 2.2.3. Electron Avalanche

As mentioned in Section 2.2.1 secondary ionizations may occur if the energy of the incoming particle is high enough. Hence if the primary ionization electrons gain enough energy from being accelerated by the electric field secondary ionizations are caused. If this effect continues an electron avalanche will occur. Since the electric field is highly localized along the central wire of the detector, ionization avalanches only occur within a few radii of this wire [3]. As explained in the previous Section since electrons have higher drift velocities than positive ions the electron avalanche will form the particular drop shape represented in Figure 2.2.3.

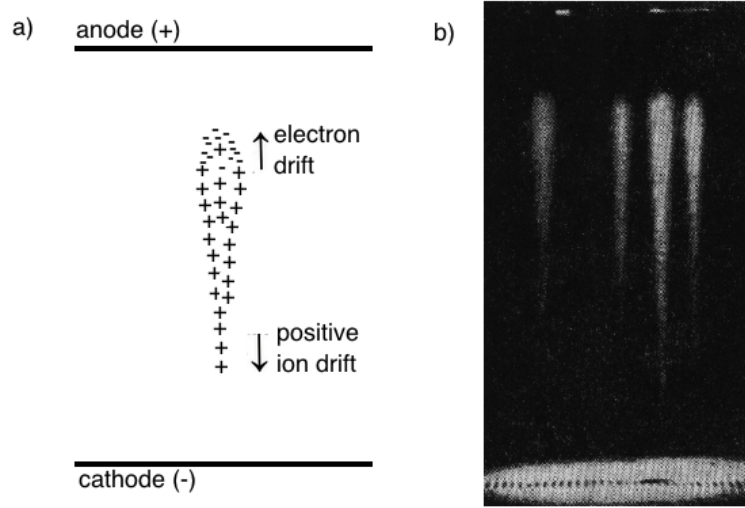


Figure 2.4: a) Symbolical representation of electron avalanche with liquid drop shape. b) Actual photograph of electron avalanche formation [9].

To further characterize the electron avalanche,  $\alpha$  is defined as the mean free path of an electron for a secondary ionizing collision. Therefore,  $1/\alpha$  is the probability of an ionization per unit path length. Hence if there are  $n$  electrons, there will be  $dn$  new electrons created in the path  $dx$  [3].

$$dn = n\alpha(x) dx \quad (2.12)$$

Since the electric field in Equation 2.2 is non-uniform  $\alpha$  is a function of  $x$ . Integrating the expression in Equation 2.12 and defining  $n_0$  as the number of primary ionization electrons then an avalanche multiplication factor  $M$  can be calculated.

$$M \equiv \frac{n}{n_0} = \exp\left(\int_{r_1}^{r_2} \alpha(x) dx\right) \quad (2.13)$$

Where  $r_1$  and  $r_2$  are the initial and final points in the path. This multiplication factor is known as the gas gain or gain, a fundamental property of all proportional counters and specifically of the studied single wire detectors [3]. Recall that the current produced in the anode of the detector is directly proportional to the energy of the incoming particles. Therefore, an increase in gain will cause an increase in measured energy. Note that the energy of the particle has remained constant, therefore it is vital to calibrate a detector with respect to its given gain.

### 2.3. Single Wire Detector Aging

As with any particle detector, single wire gaseous detectors have an apparent aging process. The root of this aging occurs by the deterioration of the anode wire which causes the detector

signal to worsen. Consequently, the measured energy spectrum is affected. It is by studying this effect that the aging effect can be understood. As a single wire detector ages the initially Gaussian peak formed by measuring the energy of incoming particles starts to spread out. This aging process eventually forms what appears to be a second peak in the energy spectrum as in Figure 2.5. Note that no new particles are impaling the detector at a different energy, just that detector itself is not measuring correctly the energy of the particles emitted by the source. This spreading out of the peak has three main effects that are of interest in this discussion.

1. Reduction of gain. The appearance of a second peak lowers the mean of the peak measured as a whole. The individual means of the peaks measured separately are also lowered. The loss of gain is reflected as a loss of measured energy (peak mean).
2. Change of overall peak width (when both peaks are measured as one). The FWHM varies; this change along with the drop in energy affects the resolution (refer to Equation 2.1) of the detector.
3. Fitting problems. This is a practical effect of the aging effect. In general a code for fitting this peak should be able to fit it at any point in the aging process. However the smooth transition between one peak and two leads to coding challenges.

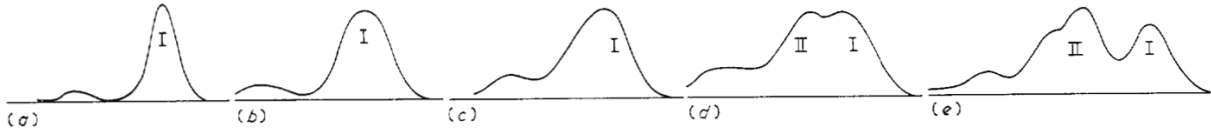


Figure 2.5: “Five successive pulse-height distributions as they develop during irradiation with a 5.9 KeV source of a proportional counter filled with argon + 10 % methane, demonstrating the ageing effect” [10].

In general there are three major causes that alter measurements made by this type of detector. First, variation of environmental parameters such as temperature and pressure cause an immediate effect on the measurements. This does not contribute to aging but rather to instantaneous measurements. The environmental fluctuations should be corrected for since they can be considerable. Pressure changes can change the density of medium inside the ionizing gas chamber in the detector effectively increasing the amount of particles that can be ionized. The same is true for temperature fluctuations. Second, the construction of the detector has a major impact on its performance. Leaks can alter the gas mixture sufficiently and therefore alter the current measured. Oxygen is a particular contaminant that is of interest so its levels are always monitored within the chamber. Finally, aging itself alters measurements. The deposition of contaminants on the central wire is the most important cause for aging. Contaminants can come in through leaks or be produced inside the detector itself by the outgassing of the materials used to build it.

### 2.3.1. Deposition on Anode

Many studies [10, 11, 12, 13] have concluded that aging effects in single wire detectors occur mainly because of deposits in the anode wire. This is discussed in detail in [10]. Two single wire detectors that allow a free exchange of gas between them were run simultaneously forming what is referred to as a twin counter. One of the anodes is irradiated continuously while the other is not. While applying the same voltage to both anodes it was discovered that the anode being irradiated demonstrated the aging effect while the anode that was not irradiated did not. Since they share the same gas, differences in temperature and pressure or the composition of the gas should affect both equally; hence, the difference must be in the anode itself. When the irradiated anode was annealed (heat treated) the original single gaussian shape was obtained. This eliminated the possibility of permanent damage to the anode such as that of sputtering (when atoms of a target material are ejected when bombarded by energetic particles [14]). The conclusion was that there must be deposition on the anode wire mostly composed of negative ions. Within this particular study it was found that deposition on an anode wire with a radius of  $12.5 \mu m$  amounted to  $1-3 \mu m$  resulting in a loss of gain by a factor of 2. Since the drift velocity is directly proportional to the electric field  $E$  as given by Equation 2.8 and in turn  $E$  is highly dependent on the anode wire's radius  $a$  (Equation 2.2) a slight change in  $a$  due to contaminant buildup will affect the gain. This was also demonstrated during a characterization of ATLAS's radiation trackers (also single wire detectors), where actual deposition is depicted.

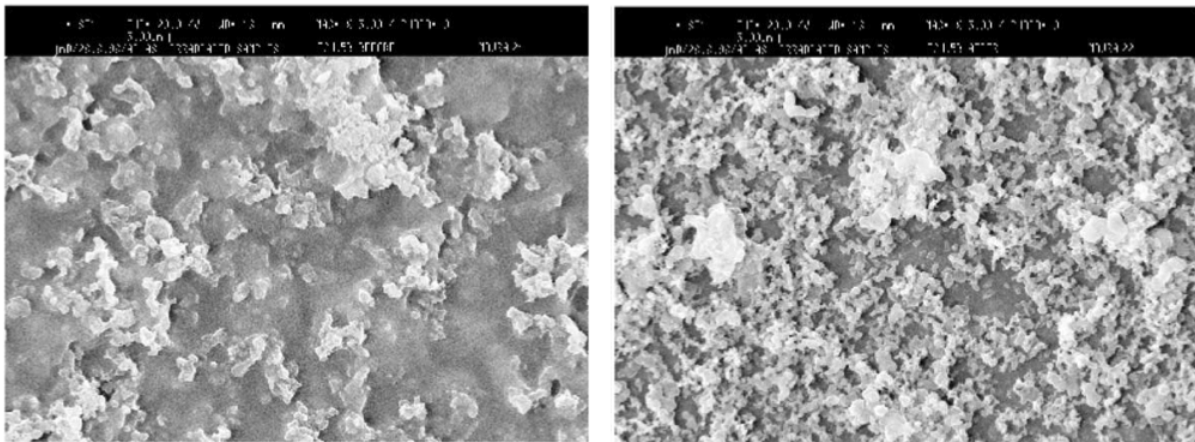


Figure 2.6: Micro-photograph of the cathode surface before irradiation  $25 \times 20 \mu m$  (left). Micro-photograph of the cathode surface after irradiation  $25 \times 20 \mu m$  (right) [11].

Furthermore, several other studies [15, 16] have characterized the aging of single wire detectors with respect to the materials that were used to build them. This has answered the question with respect to what materials are being deposited on the anode wire. For example in [15] a comparison between a detector built with a stainless steel box and one made with fiberglass is made.

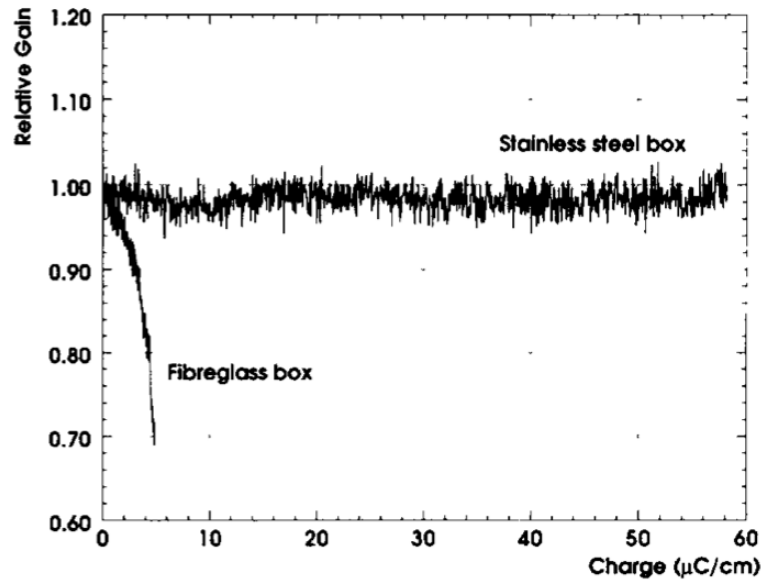


Figure 2.7: Gain dependence on charge measured for the same MSGC plate assembled in a clean, stainless steel box, and in a fibreglass box with rubber 0-rings and Araldit epoxy. [15]

In Figure 2.7 it is evident that the relative gain (instantaneous gain divided by initial gain) drops rapidly for the fibreglass box due to increased deposition on the anode caused by the materials that were used to build it. The x-axis in this Figure is integrated charge, a concept that will be analyzed in detail in Section 7. In short, it is the charge that has been accumulated in the detector by integrating the current over time. In [10] the deposits on the wire were mainly found to be made of hydrocarbons. This is also confirmed in [15] in the fibreglass box case where the deposits originate from the outgassing of the glues used to build the detector. In this very same study different glues are compared since the least outgassing glue is desired. At higher temperatures glues tend to outgas more so the detectors were tested at room temperatures and above.

#### 2.4. Analysis Tools: ROOT

ROOT is an object oriented framework that was designed to process large amounts of data efficiently. It provides the user with a vast array of objects and fast access to their attributes ideal for the high volume data processing needed in high energy physics. In fact, it is the most widespread software in use in the field [17]. ROOT's native language is C++ which makes it relatively easy for new users to learn. Many of the objects designed in this framework are essential to the analysis that follows. The main advantage of using ROOT for this study is the way it can treat clusters of data as objects; therefore providing a complete set of methods that can be implemented or attributes that can be extracted for each object (data cluster). It provided the histogram construct, fit capabilities and graphic visualization tools needed for the creation of the

scripts presented in Section 4.

Section 5 deals mainly with the spectra generation and fits. The spectra corresponds to histograms created by using ROOT. With the provided capabilities the histogram objects packaged six hours of data each and extracting statistical information about this data was as easy as accessing the object's attributes. For example the histogram mean could be extracted with one line of code. Fits can be efficiently created and are also treated as objects; therefore the fit parameters can be extracted with ease as well.

### 3. Experimental Setup and Measurements

To observe the aging effects in single wire detectors two of these detectors were run at the same time over the course of a year. Except for the periods with  $CF_4$  concentration changes the mixture of gasses in the chambers is composed of 45%  $Ar$ , 15%  $CO_2$  and 40%  $CF_4$  <sup>1</sup>. One of them is irradiated by a source at a constant position while in the second the detector is moved to different positions throughout the year.

#### 3.1. Timeline

##### 3.1.1. Single Wire 1 (SW1)

1. 04/09/13: Start of Irradiation. Fixed position.
2. 07/01/14: Unexpected gain drop
3. 28/04/14:  $CF_4$  gas mixture changes start
4. 28/05/14:  $CF_4$  gas mixture changes end
5. 14/07/14: End of Irradiation

##### 3.1.2. Single Wire 2 (SW2)

1. 04/09/13: Start of Irradiation. Position 3.
2. 01/10/13: Move to position 4.
3. 01/11/13: New SW2. Position 3.
4. 21/11/13: Soldering on pin connector.
5. 07/01/14: Position 3 (bottom).

---

<sup>1</sup>The reason this 45%  $Ar$ , 15%  $CO_2$  and 40%  $CF_4$  mixture was chosen is that this single wire detector will be used to monitor the gas mixture sent to a detector that uses exactly this gas mixture composition. The standard gas mixture used is composed of 70%  $Ar$  and 30%  $CO_2$ .

6. 27/02/14: New SW2 position 4.
7. 28/04/14:  $CF_4$  gas mixture changes start
8. 28/05/14:  $CF_4$  gas mixture changes end
9. 14/07/14: End of Irradiation

### 3.2. Radiation Source

- $^{55}Fe$  source producing 5.9 KeV photons.
- Activity (measured in becquerels: [Bq] = 1 decay per second): 1 MBq

## 4. Script File Map

Raw data from the detectors was collected continuously for a year alongside many environmental parameters (such as temperature, pressure and oxygen levels within the detectors). This constituted a challenge for the analysis that follows since problems with the data acquisition software arose and all this data had to be carefully matched. The following series of scripts were created to carry out this process (Refer to Figure 4). The general idea is to group clumps of data into histograms from which different trends are to be extracted through carefully designed fits. These trends, such as FWHM, RMS, peak position or gain, fit means, integral and integral noise are plotted as a function of time or integrated charge to analyze how aging affects them. Raw environmental data is included to correct for fluctuations in temperature and pressure. Finally, supplementary scripts calculate specific relations between different parameters to give a better understanding of the aging effects.

# File Map

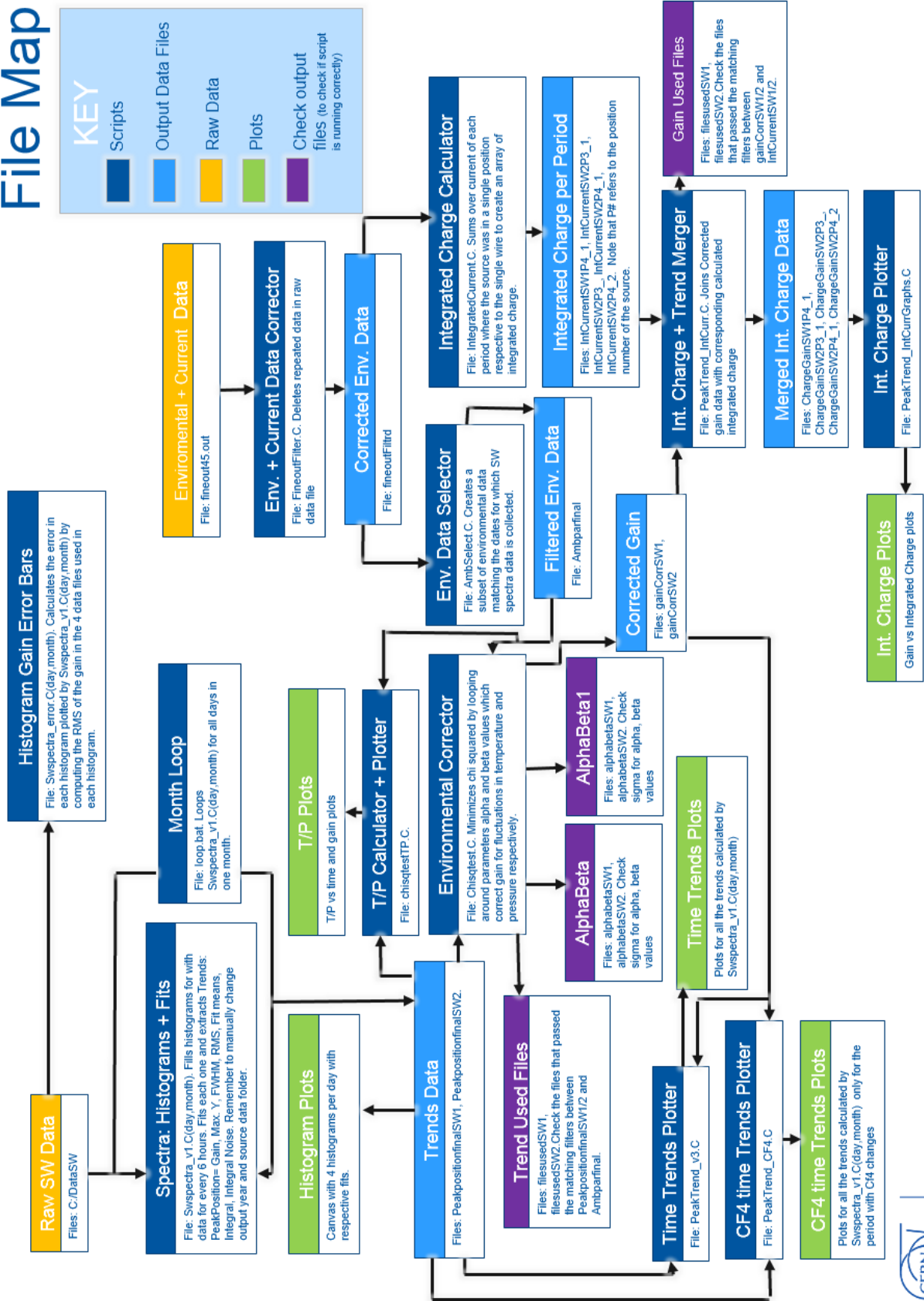


Figure 4.1: File and script map



## 5. Spectra Generation

The energy spectrum is of great importance to understand the detector performance. Depending on the radioactive source strength and its distance from the detector itself the spectra (with a statistically significant number of counts) can be generated within seconds to hours of irradiation. However the quality of these measurements are only as good as the detector itself. In this Section the effects of aging in the detector will start to become apparent through a systematic generation of statistically significant histograms, their fits and the evolution of fit parameters over time.

### 5.1. Introduction

In order to understand the following methodology a few terms should be defined:

- **Histogram:** a graphical representation of the Single Wire data distribution. Effectively the number of counts per energy value in the detector; also known as the Spectrum.
- **Bins:** The energy range (represented as ADC counts: refer to x-axis of Figure 5.2) is divided into subranges called bins. The histogram is the representation of particle counts incoming with an energy that falls within this energy subrange (bin).
- **Binning:** The size of the bin or number of bins for a given range. Increasing bin size effectively decreases resolution but also eliminates variability within that bin (only a computational artifact).
- **Pedestrial:** An artificial signal (created by the electronics in the setup) that is present throughout most of the data. This signal is purposefully inserted at a lower energy (than the detected particles) in the spectrum to serve as a marker to detect electronic shifts. If the electronics themselves are failing (ADC shifts can occur due to these malfunctions) the real signal can be extrapolated by subtracting the signal coming from electronics

### 5.2. Scripts: Spectra: Histograms and Fits (SWspectra\_v1.C)

This script constitutes the first step in processing the Raw SW Data. Through a series of techniques it groups data into histograms and produces their corresponding fits. For every histogram created a series of parameters is collected and stored as Trend Data in the PeakpositionfinalSW1.txt and PeakpositionfinalSW2.txt files. The collected parameters are as follows: Peak Position or Gain (histogram mean), Fit Means (two Gaussian are used to fit the histogram, refer to Section 5.2.3), and Full Width at Half Maximum (FWHM), Root Mean Squared (RMS), Maximum, Integral, and Integral Noise of the histogram. This script by itself only runs the data collected from one day at a time <sup>2</sup>.

---

<sup>2</sup>There is a supplementary script that runs this script for multiple days at a time. Refer to Section 5.2.5

### 5.2.1. Grouping Data into Histograms

Since data is being collected continuously, an appropriate choice of data sampling and grouping is needed. The quantity of data grouped into a histogram has to be sufficient to provide the correct statistics for the time slot that that group corresponds to. Nevertheless it can't be too large since parameters can fluctuate too much within that time period. In fact, a script was created just for this purpose. Data is sampled every six hours resulting in four histograms a day each containing one hour of data. This is enough to be statistically significant; however, the question that arises is that if environmental (among other) fluctuations can significantly change measurements during the course of an hour. The output of this script shows that the error ranges within 1% of the measured parameters and therefore the sampling size and rate are appropriate.

### 5.2.2. SW Loops: Histogram Arrays

SWspectra\_1.C runs all the data (mean, max, RMS, FWHM, integral...) collection code from the histograms (h0\_1[d][m], h6\_1[d][m], h12\_1[d][m], h18\_1[d][m], h0\_2[d][m], h6\_2[d][m], h12\_2[d][m], h18\_2[d][m]) in one loop for each single wire instead of running each histogram independently. This has a slight disadvantage in that parameters (such as fit parameters) can not be set specifically for each histogram but rather have to work in general for each day. Ultimately a general approach is what we are looking for since we want to run this data once over the course of a year; nevertheless, it makes individual corrections more cumbersome to achieve. To carry out the loops for the single wires a single array of histograms was created containing all the already filled (h0\_1[d][m], ...) histograms: testarray. Thus data collection operations are conducted on the individual elements of this array looping over its indexes.

### 5.2.3. Fits and Methodology

The fit methodology is of great importance for the purposes of this study. Effectively during the creation of the scripts, major differences in output data files was observed when fit methodology was changed. Ideally the observed spectra should be Gaussian; nevertheless in reality this spectrum can become asymmetric over time eventually resulting in the appearance of a second peak. The wire diameter is changing due to aging; therefore, different electric fields are present in the two sides of the single wire. The second peak is a direct result of this. Each histogram is thus fit as the sum of two Gaussians. This method was effective for clear double peak spectrum but failed when the spectra was mostly a single Gaussian peak. Therefore a guided two Gaussian fit method was developed as shown in equation 5.1.

$$f(x) = a_1 e^{-\frac{(x-b_1)^2}{2c_1^2}} + a_2 e^{-\frac{(x-b_2)^2}{2c_2^2}} \quad (5.1)$$

Where  $a_1$ ,  $b_1$ ,  $c_1$ ,  $a_2$ ,  $b_2$ , and  $c_2$  are the fit parameters. It is important to understand what the effects of a guided fit may be. If no guidelines (the parameters in equation 5.1) are set the

fits will fail for an unacceptable percentage of cases. However if not enough liberty is given the fits wont resemble the data. Therefore it is important to have a careful balance between what is set and what is left to fit calculation.

Due to aging the peaks (however small) will drift towards opposite sides of the histogram mean. Therefore the Gaussian fits were forced to be on opposite sides of the histogram mean by setting the corresponding range for the parameters  $b_1$  and  $b_2$ :  $Min(Range) < b_1 < Mean(histogram)$ ,  $Mean(histogram) < b_2 < Max(Range)$ . Here the Range is that calculated by the algorithm explained in Section 5.2.4. This constraint alone increased the fit rate from around 50% to around 80%. Furthermore more evident fit limits were set, such as matching the maximum of the fit with the maximum of the histogram.

#### 5.2.4. Peak Range for Fits

Originally, the range for which all the data was collected from the histograms was fixed. This made some fits fail when the peak moved out of range and it creates the need for an algorithm to find where the main peak is. The main method is located in lines 273-317 in the code included in Appendix 9.6. This method consists on setting a threshold y-value on line 268 (needs to be above the noise). If values are found above this value they are considered part of the main peak, unless they are part of the pedestrian. Hence, the first two points where the threshold is passed are discarded since they are part of the pedestrian. The third and last points where the threshold is passed is considered the limits of the main peak. If no pedestrian is present (not typical) line 305 must be activated. Now the first and last instance where the threshold is passed are considered the limits of the main peak. Notice that however necessary the presence of the pedestrian, it complicates the programing and flow of the data processing.

A few bins are added on each side of the range to recapture the data missed by the threshold setting. Since a higher threshold will result in a higher percentage of fits working an optimal setting of threshold and additional bins (on each side) must be searched for depending on the noise in the data and the chosen binning. Note: FWHM calculation uses same algorithm but with the half maximum as a threshold. Due to the subtleties of this methodology the FWHM only has the accuracy of the size of the bin, thus the calculated FWHM values are somewhat clumped into bins as demonstrated in Figure 5.1.

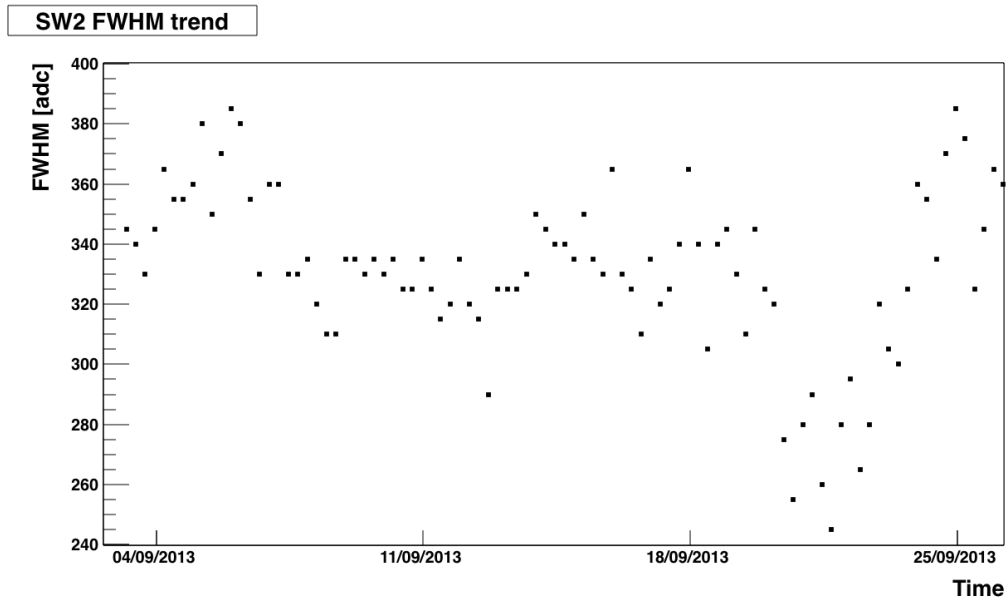


Figure 5.1: FWHM time trend over first month of detector operation of SW2. Values of FWHM can only be multiples of 5 due to the set bin size of 5 ADC counts.

### 5.2.5. Month Loop

Recall that the `SWspectra_1.C(d,m)` script just generates histograms for one specific day of data. To generate the histograms for the entire year this code must be run iteratively by changing the date in each iteration. However since this code can not run inside another root script it must run inside the Unix Shell Bash. The month loop is a very simple bash script that runs this code in the described manner for each month. Hence, instead of running the code for each day of the year individually the bash script is run for each month of the year drastically saving user time [18].

## 5.3. Results and Analysis

### 5.3.1. Spectra

The methodology presented in Section 5.2.3 is exemplified by Figures 5.2 and 5.3. Here both discussed situations are shown for which the fits were effective. Figure 5.2 shows no sign of aging since just one gaussian peak is observed. The fits had to be fine tuned so a high percentage of single peak spectra like this one could be fit.

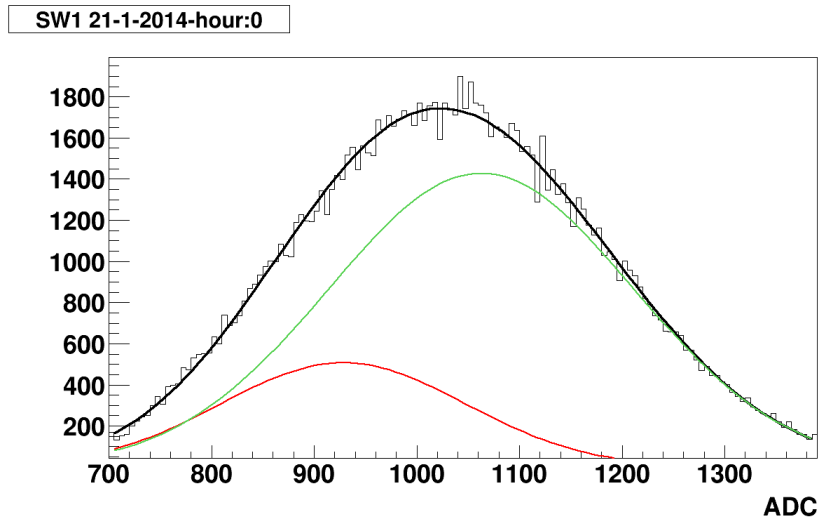


Figure 5.2: Two Gaussian fits applied to a mostly Gaussian peak. Note the positions of the peaks with respect to the mean of the histogram. The sum of both Gaussian peaks is shown in black.

On the other hand, Figure 5.3 shows clear signs of aging. The fits worked exceptionally well for these cases since there is a clear double peak.

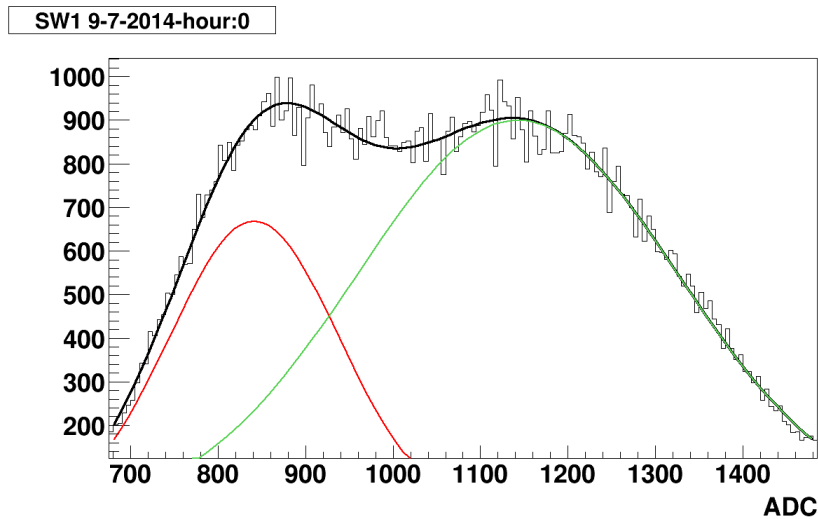


Figure 5.3: Same as in Figure 5.2 but with a clear double peak histogram.

To compare to the illustration in Figure 2.5 a series of histograms are set side by side in Figure 5.3.1. It is clear that the aging effect is manifested in this set of 5 histograms evenly selected over the course of a month.

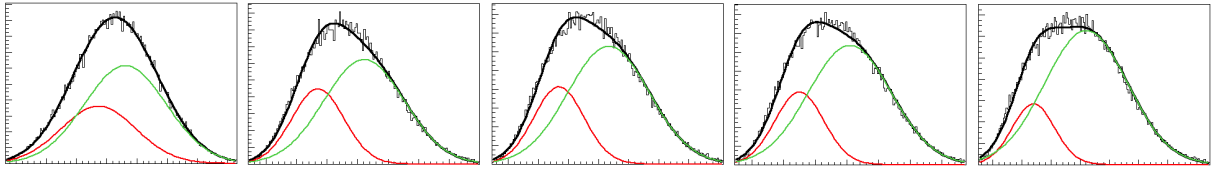


Figure 5.4: Five pulse-height distributions as they develop during irradiation of SW2 spread out over a month (between the dates 1 and 2 in Section 3.1.2).

With the techniques described in Section 5.2.3 the code was able to fit around 95% of all histograms. This is a vast improvement over just fitting the histograms with the unconstrained sum of two gaussians fit given in Equation 5.1. In fact the latter only produced a fit rate of 50%.

Even though the aging effect is not as dramatic as in Figure 2.5 the gradual transition to two peaks is observed. This becomes evident by analyzing carefully the trend of the individual peak means as follows.

### 5.3.2. Fit Means

The yearlong fit means trend can be constructed by extracting the means from the two Gaussian calculated through the guided fit described in Section 5.2.3. These are presented for SW1 and SW2. Each of the means are normalized with respect to their original values where no aging is present.

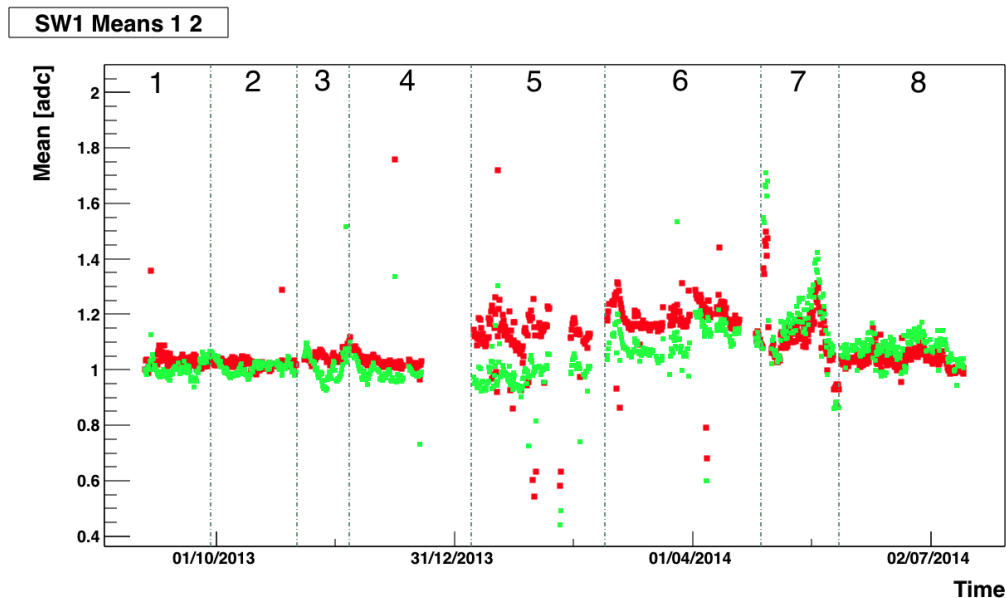


Figure 5.5: Trends of the normalized means of the two Gaussian fits applied to each histogram over one year of detector operation of SW1. The dashed lines mark the dates corresponding to those in Section 3.1.2 for SW2 for this plot and all that follow.

In this case we are interested in the in the trend for the first two periods of SW1 compered with

that of SW2. In general this will be the case for most of this study because either more complex variables come into play during some portions of the year (like the month of  $CF_4$  changes) or there is not a clear trend. In SW1 the means hover around each other depending on how the fit converged. However for the first two periods in SW2 there is a separation of the means over time.

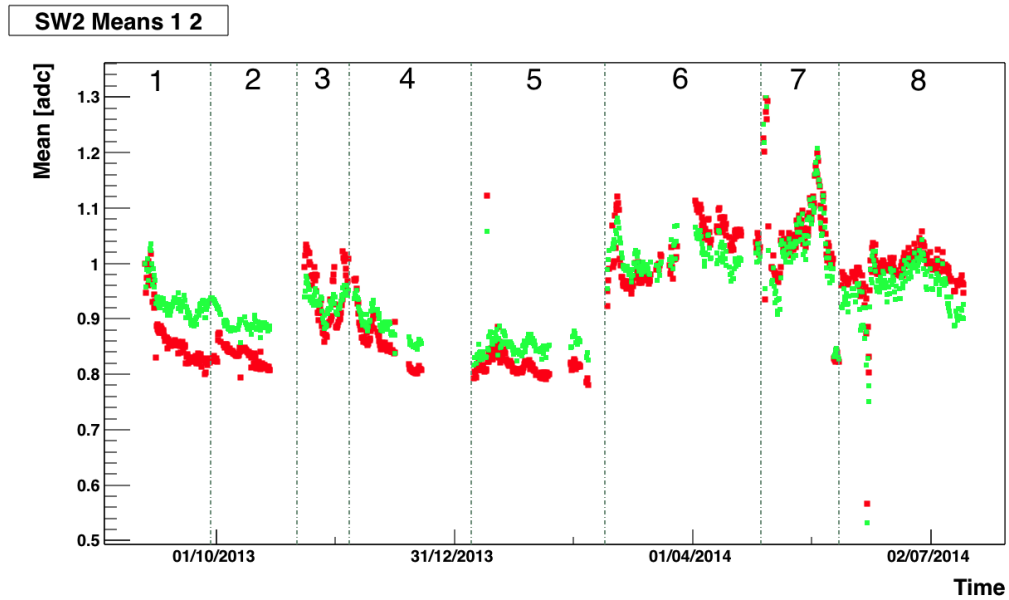


Figure 5.6: Trends of the normalized means of the two Gaussian fits applied to each histogram over one year of detector operation of SW2. The colors used for the different means match those in Figure 5.3.1.

Zooming into the first period of the year this becomes even more apparent.

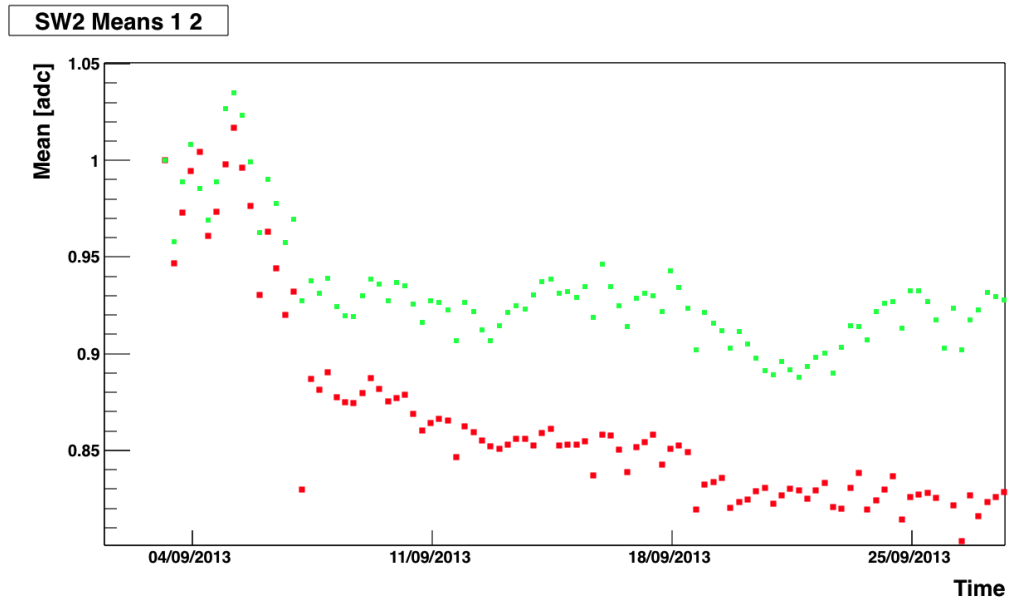


Figure 5.7: Trends of the means of the two Gaussian fits applied to each histogram over one aging period of SW2. This data corresponds to the period between the dates 1 and 2 in Section 3.1.2.

To quantify this difference the relative separation between the peaks can be calculated.

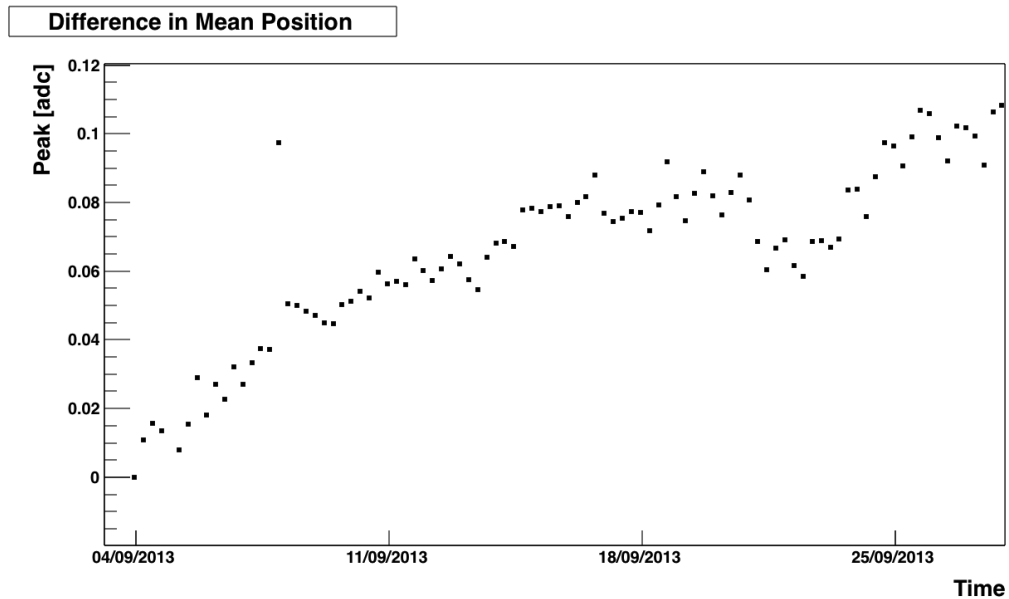


Figure 5.8: The distance between the original peak and the emergent peak (the two fit means) for the period described in Figure 5.7. Since the means in Figure 5.7 are normalized to the initial value of the period this plot represents the fractional difference between the two peaks.

The separation grows rapidly at the beginning and then this rate tapers off. This is a clear sign that the aging effect rate decreases over time. As stated in Section 2.3.1 the aging effect



occurs due to deposition in the wire. Since the rate of the aging effect decreases then the initial deposits have a greater effect than the ones that follow. Setting all the constants equal to 1 except  $a$  and eliminating the  $r$  dependence in Equation 2.2 results in:

$$f(a) = \frac{1}{\ln\left(\frac{1}{a}\right)} \quad (5.2)$$

Where  $a$  is the now variable radius of the anode. The derivative of  $f(a)$  provides insight into the sensitivity of the electric field with respect to this value.

$$\frac{d}{da}f(a) = \frac{1}{a \ln^2(a)} \quad (5.3)$$

Since we set  $b$  (the inner radius in the cylinder of the detector) to 1, an approximate value for  $a$  would be  $a = 0.001$  given the standard specifications of this type of detector. Henceforth we can study Equation 5.3 in this region.

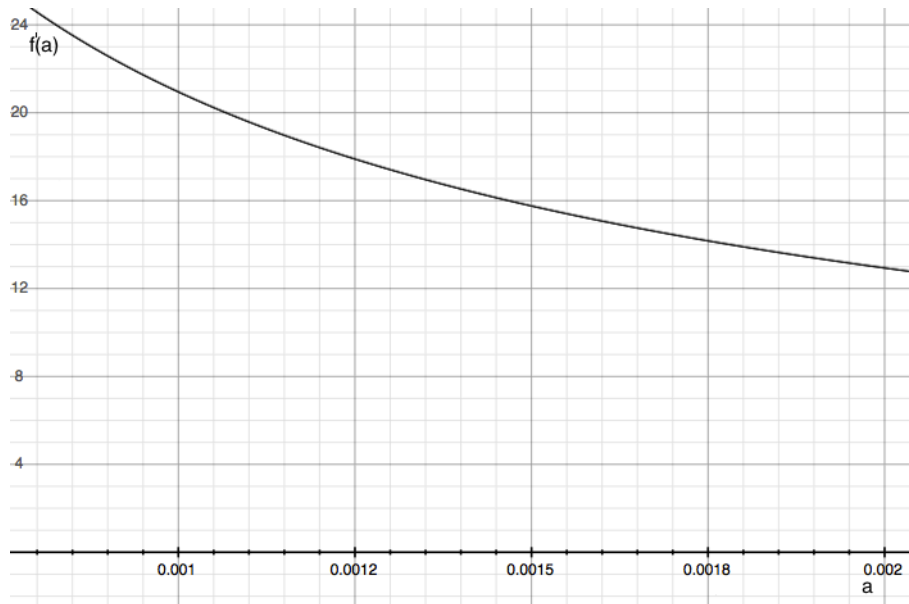


Figure 5.9: A graph of Equation 5.3 within the range defined by standard single wire detector specifications.

The derivative of  $f(a)$  is decreasing, since the electric field changes more dramatically at lower values of  $a$ . This explains why initial deposits have larger effects than further ones thus slowing down the aging effect.

### 5.3.3. Relative Gain

As opposed to extracting the means of the two peaks from the fits themselves as in the previous section the mean of the peak as a whole is gathered from the actual data within the histogram.

This mean is what could be converted to the energy of the incoming particles. However, this is not of interest in this discussion. Since the energy of the particles is constant the change in this mean will reflect a change in the gain of the detector. Thus the relative gain is defined as:

$$\text{Relative Gain} = \frac{\text{instantaneous Gain}}{\text{Initial Gain}} = \frac{\text{instantaneous Peak Mean}}{\text{Initial Peak Mean}} \quad (5.4)$$

The Relative gain is now plotted for the entire year for SW1 and SW2 in Figures 5.10 and 5.11 respectively.

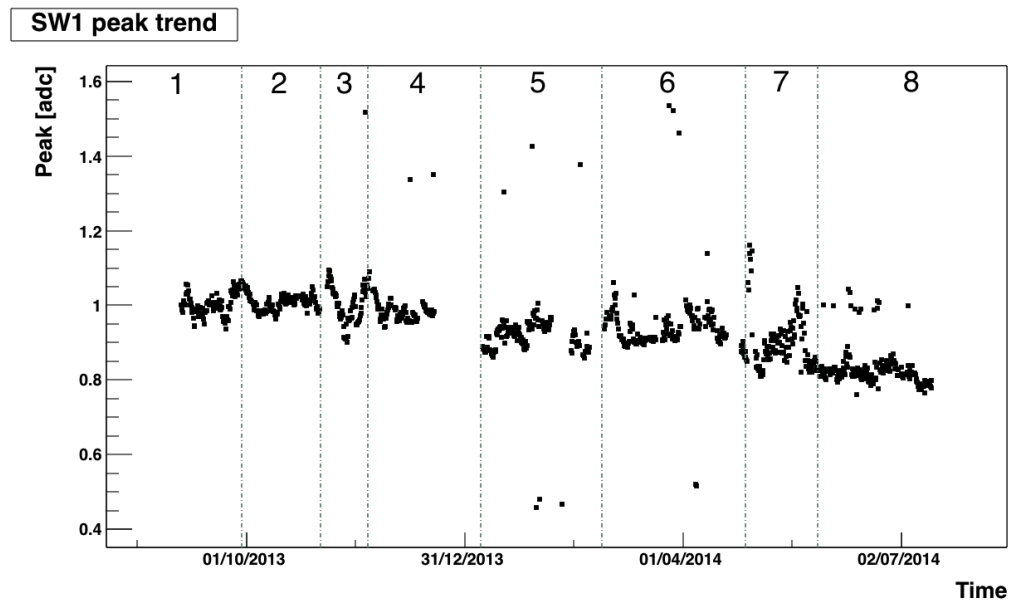


Figure 5.10: Gain time trend over one year of detector operation of SW1

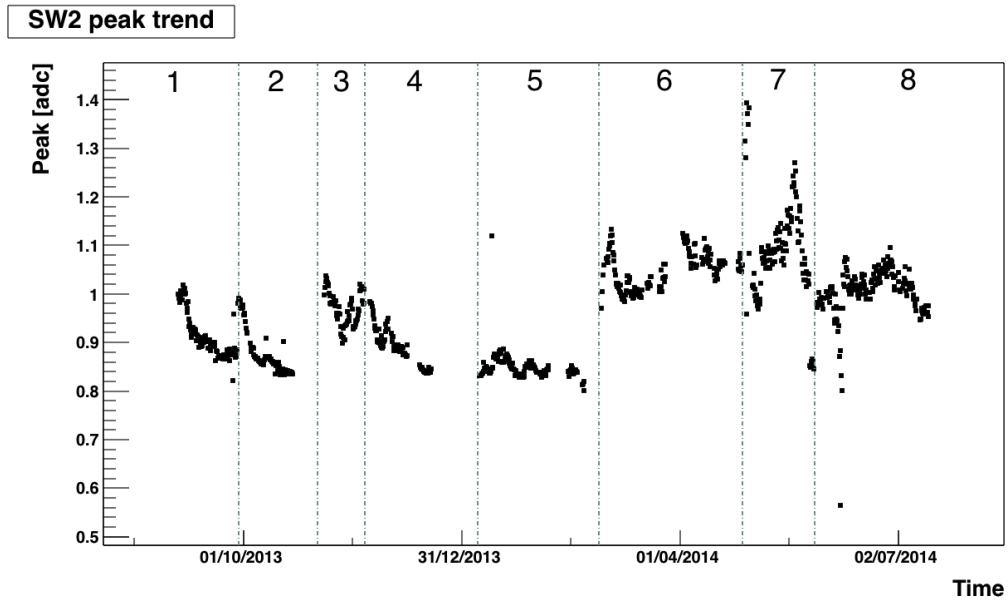


Figure 5.11: Gain time trend over one year of detector operation of SW2

Again the focus will be on the first 2 periods of this trend. As expected the gain follows the same trend as the individual means in the previous Section. SW2 undergoes an aging process during the first month of irradiation. Then the detector is moved to a different position (refer to the timeline in Section 3.1.2). The relative gain shoots up after this change. This occurs because the deposits in the anode are not evenly spread out [10] and this spread depends on the position of the source. Since the detector was moved, a section of the wire with less deposits must be responsible for the recovery of the gain almost to its initial value. This followed by a similar aging period. In 01/11/13 (the start of period 3) the wire was swapped for a new one and the original gain was completely recovered as observed.

#### 5.3.4. Resolution

The resolution is calculated by plugging in the instantaneous peak mean from the previous Section and the FWHM calculated as explained in Section 5.2.4 into Equation 2.1. This is done for the entire year of data to produce the plots for SW1 and SW2. To illustrate this Figure 5.12 follows.

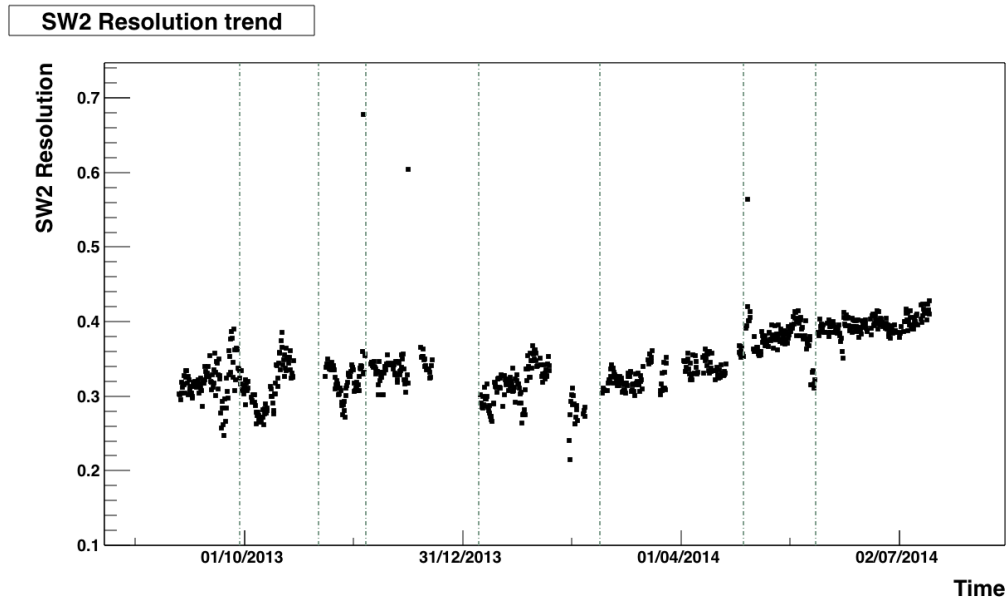


Figure 5.12: Resolution time trend over one year of detector operation of SW2

Since there are two changing variables coming into play for the resolution this trend is harder to analyze. The two initial periods that in SW2 that exhibit the aging effect in previous sections show a distinct pattern in the resolution trend. There resolution first decreases and then increases for each aging period.

#### 5.4. Conclusions

Massive amounts of data require careful packaging and processing. By grouping the particle count data in 6 hour packages, 4 spectra or histograms can be generated per day. It is important to remark on why the data was chosen to be analyzed at this level. For example one histogram could have been generated every day or every 10 minutes, so why was one generated every 6 hours? This relates to the variance of the different variables at play during each chosen histogram period. As for environmental conditions in a laboratory setting, they vary very little and an average over 6 hours is sufficient. As for the aging process, it has been observed to occur in the order of a month. Therefore with the chosen data analysis level, around 120 data points are produced for study for the periods that present aging. This value is sufficient while not being computationally overwhelming. By taking advantage of ROOT's ability to process data in histograms (which are treated as objects), statistical parameters such as the histogram mean (which is converted to relative gain) can be accessed easily by calling one of the histograms attributes. Each one of these histograms are fit by a function resulting from the sum of two gaussians. Nevertheless this fit must be guided for the fits to work for the entire year of operation. Since aging effects are present the fit must be able to handle single peak histograms and double peak histograms as aging causes the single peak to separate over time. This increase in separation becomes evident

by studying the means of the individual peaks produced by the fit and plotting their difference over time. Furthermore the aging effect is more pronounced at the beginning of each period due to the dependence of the electric field on the changing anode radius due to deposits. These deposits are the cause of the aging effect. The relative gain gives further insight into this process. Every time the anode is changed or the detector is moved the detector recovers its initial gain (or a large fraction of it for the later). A “V” shaped pattern in resolution emerges in the periods with aging effects. Unfortunately all the effects are studied just for the initial 2 periods of SW2 which exhibits clear aging periods and SW1 which does not for that time frame. A more complex analysis is required for the rest of the year since other variables come into play.

## 6. Environmental Effects

As seen in Equation 2.11 temperature and pressure fluctuations can significantly affect instantaneous measurements in the detector. Thus to produce accurate and reliable results the effects of these fluctuations must be removed. This Section contains different techniques for doing so.

### 6.1. Scripts: Environmental Corrector (`chisqtest.C`)

Fluctuations in temperature and pressure can induce significant changes in the behavior of the detector as explained before. Therefore, environmental corrections are very important since they eliminate the temperature and pressure dependence of the gain. This script cycles through two parameters to minimize the standard deviation of the gain in periods with no aging where gain variance is caused in part by these environmental fluctuations. The corrected gain is then printed alongside the uncorrected gain and the FWHM.

#### 6.1.1. Data Matching

Since raw data is coming from 2 sources, the spectra data has to be correctly matched to the environmental data. This is done to some extent independently of this script. The raw environmental data contains many repeated sequences caused by errors in data acquisition. The Environmental and Current Data Corrector (`FineoutFilter.C`) cleans up these errors. The corrected data then passes to the Environmental Data Selector (`AmbSelect.C`), a script that samples the environmental data every six hours to match the spectra data sampling. Nevertheless, `chisqtest.C` does a final check in lines 238-248 (Appendix 9.2). This final check runs through all the lines in the environmental data file for each of the lines in the spectra data file. When it finds the corresponding line in the environmental data file the values are pasted together and passed to the environmental correction section. If no match is found for a line in the spectra file this data is discarded. Note that the environmental data file does not need to be in chronological order for this to work.

### 6.1.2. Methodology

A correction term for temperature and one for pressure is introduced in the calculation of the gain as shown in Equation 6.1.

$$G_i = G_0 \left( \frac{T_i}{T_0} \right)^\alpha \left( \frac{P_0}{P_i} \right)^\beta \quad (6.1)$$

Here  $G_0$  is the uncorrected gain,  $T_0$  and  $P_0$  the average temperature and pressure respectively over the correction period,  $\alpha$  and  $\beta$  are the correction factors, and  $G_i$ ,  $T_i$  and  $P_i$  the instantaneous corrected gain, temperature and pressure respectively.

The methodology for the correction is a minimization of the standard deviation of gain over a period where no aging effects are present (where the gain is constant except for the fluctuations induced by temperature and pressure). To reduce the standard deviation the gain was calculated for a wide range of  $\alpha$  and  $\beta$  values over the selected period. The value to be minimized is then:

$$\sum_i (G_i - \overline{G_i})^2 \quad (6.2)$$

Where  $G_i$  is the corrected gain from equation 6.1 and  $\overline{G_i}$  is the average corrected gain over the selected period. With this methodology an appropriate period is chosen for each single wire detector (SW1 and SW2) and values of  $\alpha$  and  $\beta$  are looped over in a range from 0 to 5 in steps of 0.1.

### 6.1.3. Executing File

1. The standard deviation is supposed to be minimized in a period where the gain is constant except for environmental factors (no aging effects). Therefore by referring to the Time Trend Plots for the Gain (of the desired single wire to be analyzed) chose a period of “constant” gain.
2. Input this period in the period selector by entering the start date in lines 253-255 and the end date in lines 263-265 (Appendix 9.2).
3. Run the file in ROOT. A prompt will appear asking for SW1 or SW2. Chose the one corresponding to the period entered.
4. Two lines will be printed on the console. The first one is the average temperature and average pressure of the entire period.
5. An output file is created. The last line signals the selected period. 1 signals a date within the period and 0 a date outside the period.

## 6.2. Results

For SW1 the period 1/10/13 - 1/11/13 was selected and the  $\alpha$  and  $\beta$  values obtained are:

$$G_i = G_0 \left( \frac{T_i}{T_0} \right)^{1.8} \left( \frac{P_0}{P_i} \right)^{3.4} \quad (6.3)$$

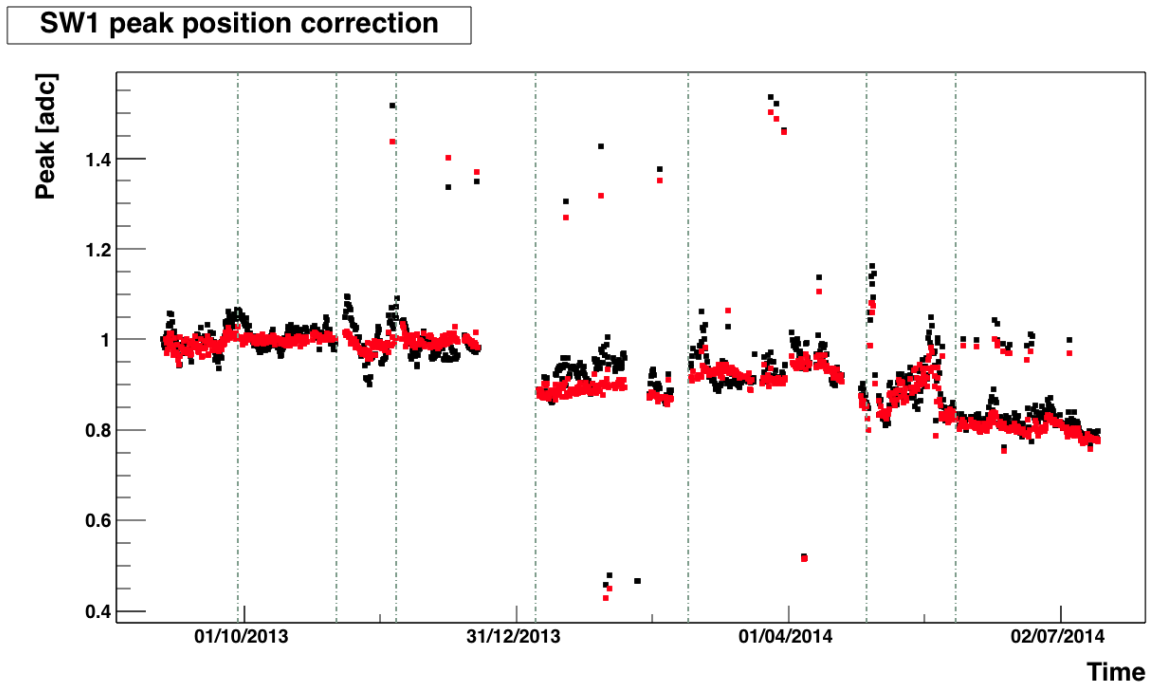


Figure 6.1: Corrected and uncorrected normalized gain for one year of detector operation of SW1. Corrected gain is plotted in red.

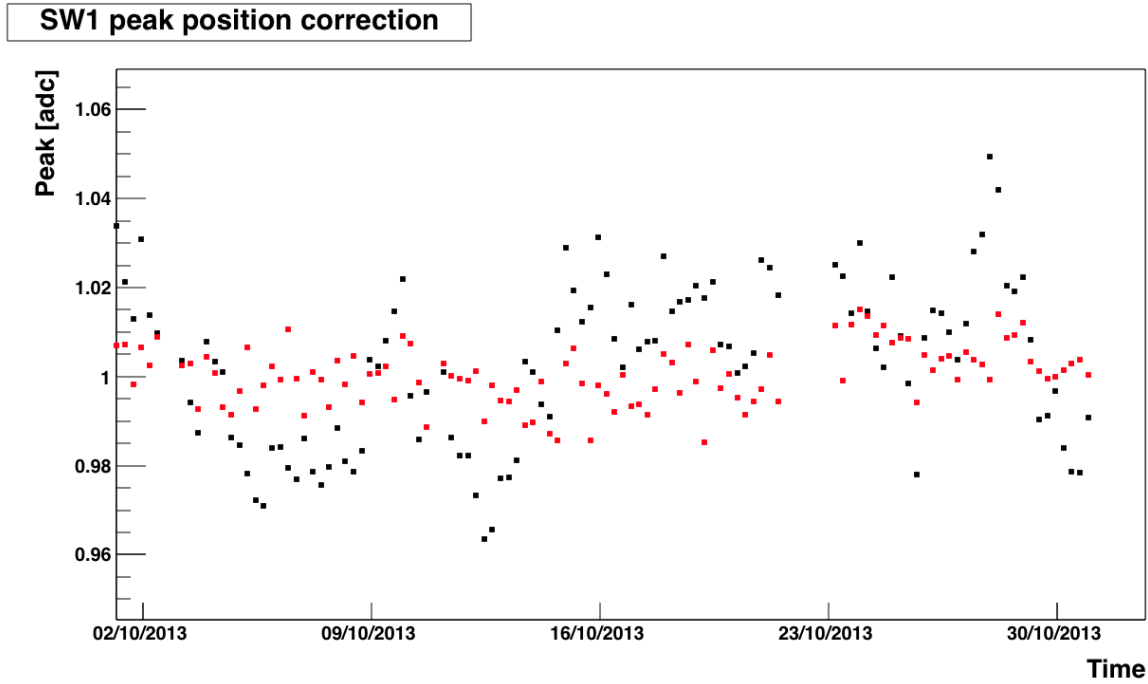


Figure 6.2: Corrected and uncorrected normalized gain the second month of irradiation for SW1. Corrected gain is plotted in red.

As it is shown by Figure 6.1 and the zoomed in version in Figure 6.2 the gain fluctuations are significantly reduced over the entire year and especially for the period from which these calculations were based. Note that even though the corrections are calculated from a subset of data they are applied to all of it.

For SW2 the period 7/1/14 - 27/2/14 was selected and the  $\alpha$  and  $\beta$  values obtained are:

$$G_i = G_0 \left( \frac{P_0}{P_i} \right)^{0.8} \quad (6.4)$$

Note the absence of the temperature related correction factor. The script calculated  $\alpha = 0$ .



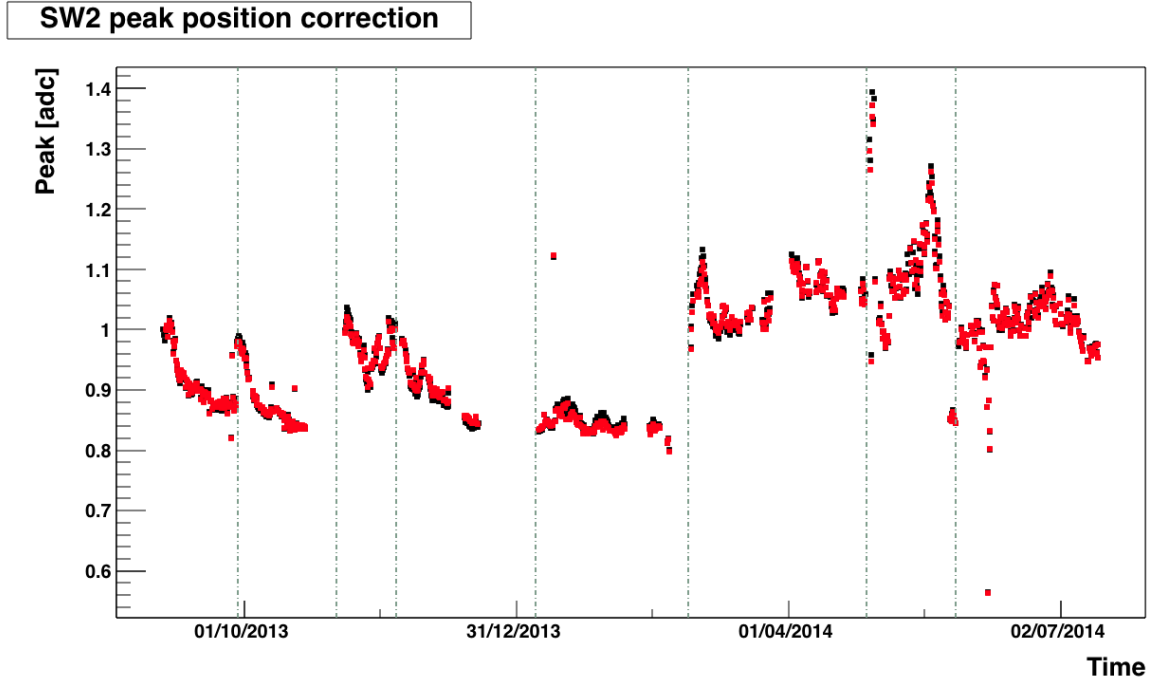


Figure 6.3: Corrected and uncorrected normalized gain for one year of detector operation of SW2. Corrected gain is plotted in red.

### 6.3. Analysis

In this Section the terms peak position, peak mean and gain are used interchangeably. The peak position refers to the peak mean in the spectra. Recall that these values are then directly proportional to the relative gain or simply gain as in Equation 5.4.

#### 6.3.1. Townsend Coefficient

The Townsend coefficient and the gain are strictly correlated and it is interesting to study the effect of temperature and pressure fluctuations on the gain. The effect of temperature and pressure on the gain is modeled by the following equation.

$$G_i = G_0 e^{B \left( \frac{T_i}{P_i} - \frac{T_0}{P_0} \right)} \quad (6.5)$$

Where  $G_0$  is the uncorrected gain,  $T_0$  and  $P_0$  the average temperature and pressure,  $G_i$ ,  $T_i$  and  $P_i$  the instantaneous corrected gain, temperature and pressure respectively and  $B$  is the so called Townsend coefficient. In laboratory conditions temperature and pressure have small fluctuations, therefore to first order Equation 6.5 can be approximated with the following:

$$G_i = G_0 \left( 1 - B + B \left( \frac{T_i}{T_0} \frac{P_0}{P_i} \right) \right) \quad (6.6)$$

Therefore to study the direct effect to temperature and pressure variations on gain we generate plots of gain vs T/P. For this purpose we start selecting periods without aging. In this way the effect of T and P variations on gain can be understood independently. Here SW2 is of specific interest (refer to Figure 6.2).

There are two periods where the gain is constant (no aging effects): the fifth time subdivision and the final time subdivision. The peak position can now be plotted against T/P for these two periods.

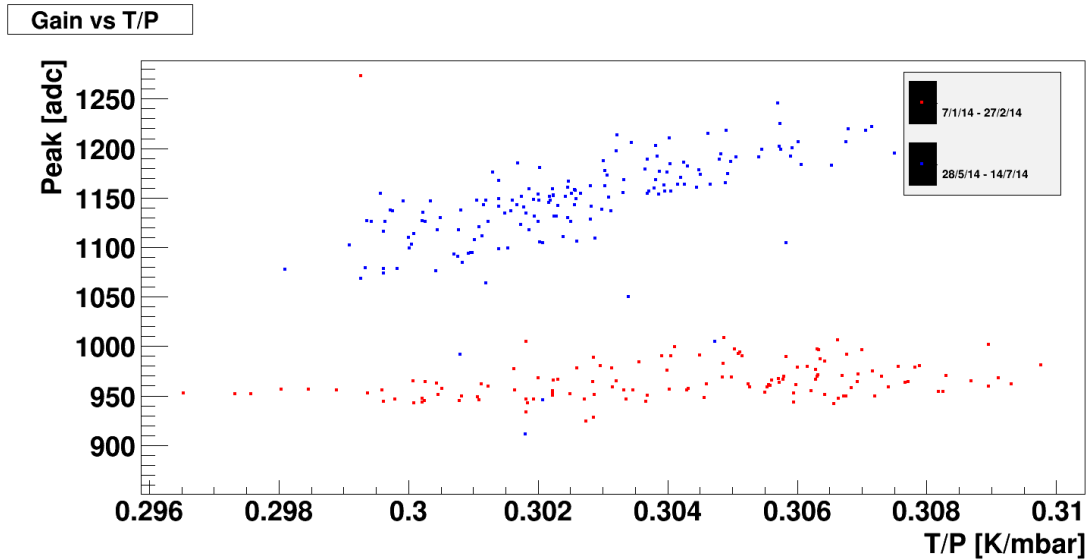


Figure 6.4: Peak Position (Uncorrected peak mean) vs Temperature/Pressure for the two periods in SW2 with no aging effects.

A similar plot but with three periods of constant gain is included for SW1 for completeness.

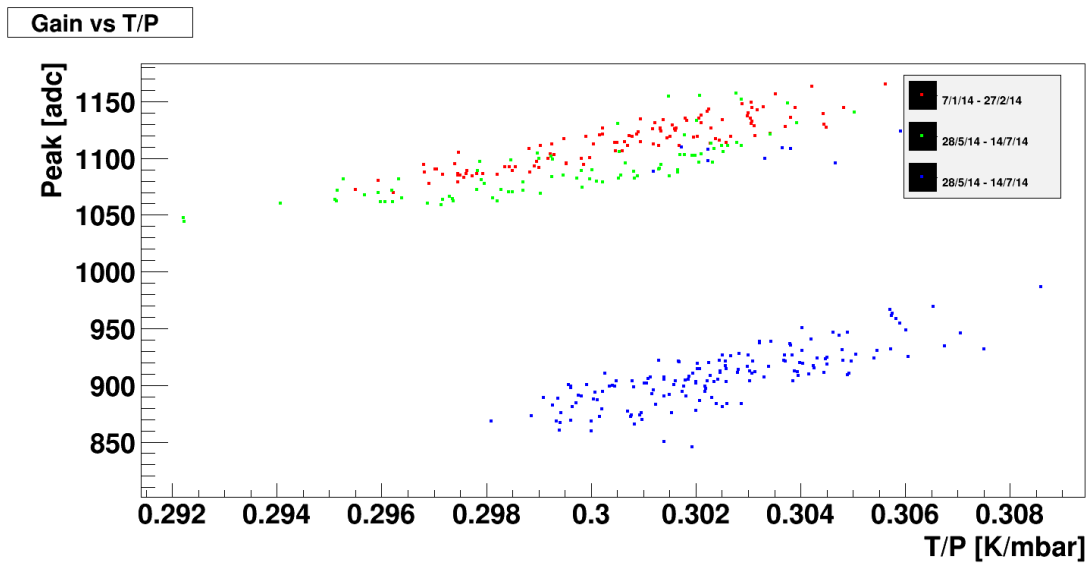


Figure 6.5: Peak Position (Uncorrected peak mean) vs Temperature/Pressure for the two periods in SW1 with no aging effects.

The slope of each one of these clusters will correspond to the Townsend coefficient. It is evident that the approximation made in Equation 6.6 is applicable because of the high linear correlation in each data cluster. Nevertheless, this plot does not explain how this relation changes over time. It is useful to plot these two parameters against time, this three dimensional plot is represented in the Figure 6.7 using a color scale to represent the z-axis. For ease of comparison to the original gain trend over a period with no apparent aging effects, Figure 6.8 is included.

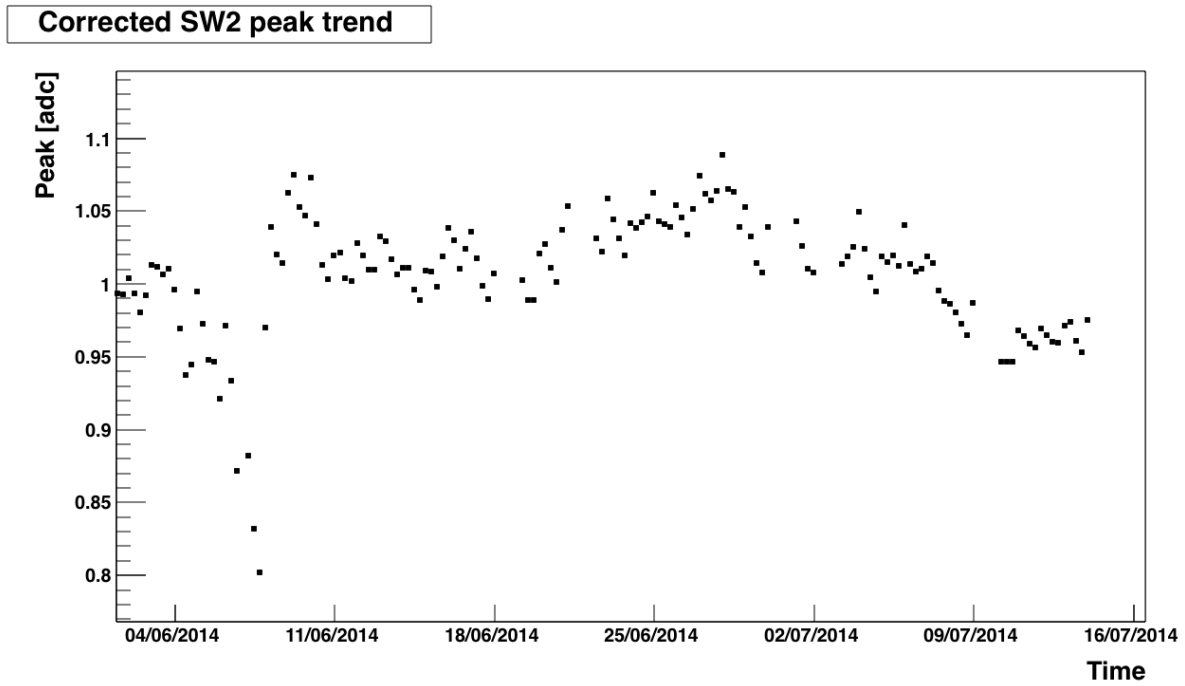


Figure 6.6: Corrected normalized gain for the period between the dates 8 and 9 in subSection 3.1.2 for SW2.

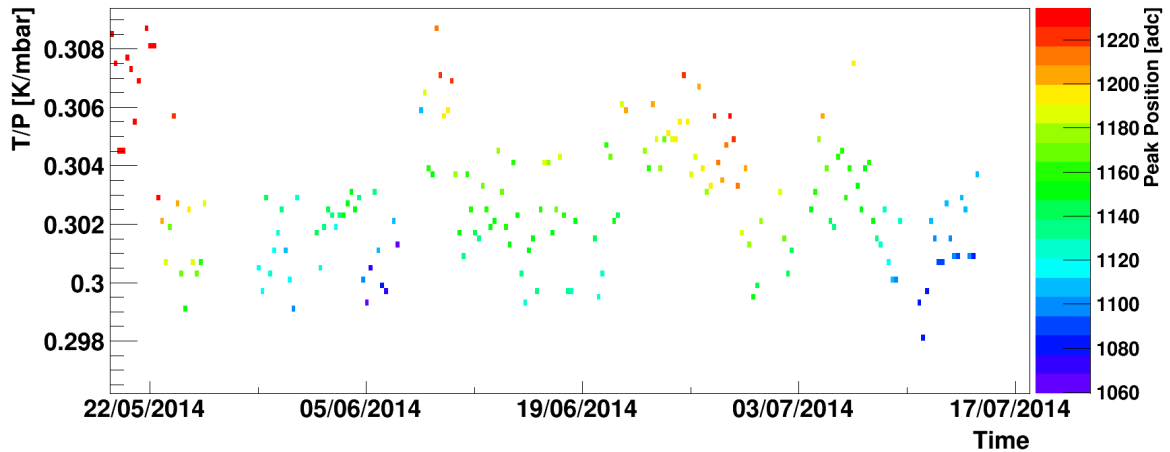


Figure 6.7: Peak Position (uncorrected peak mean (gain) (non normalized)) vs Time and T/P for the period described in Figure 6.6 in SW2.

This plot has to be read in different ways. Here horizontal color stripes can be observed throughout the whole time period. Going vertically one read off the data that is being represented in Figure 6.7: as T/P increases the peak position increases linearly. However, this plot adds another dimension to Figure 6.6, demonstrating that this trend holds constant over time. The

opposite can also be done. Periods where aging is present can be plotted in the same fashion as in Figure 6.7. For ease of comparison to the original gain trend over the aging periods, Figure 6.8 is included.

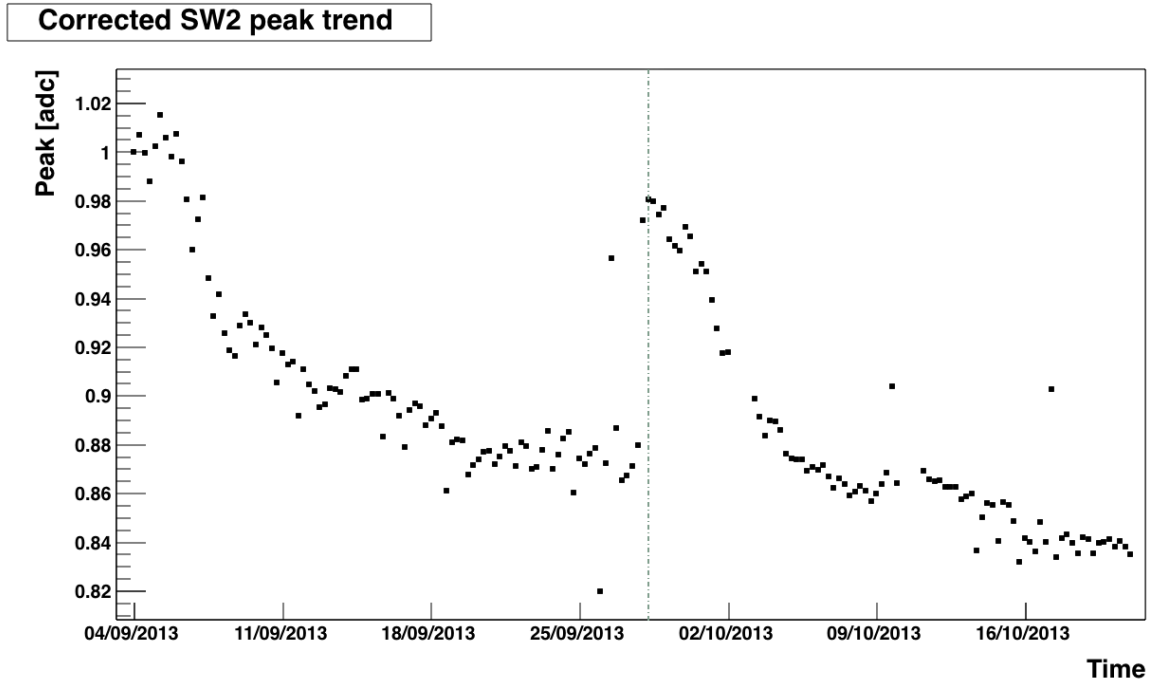


Figure 6.8: Corrected normalized gain for the periods between the dates 1 and 2 and 2 and 3 in subSection 3.1.2 for SW2.

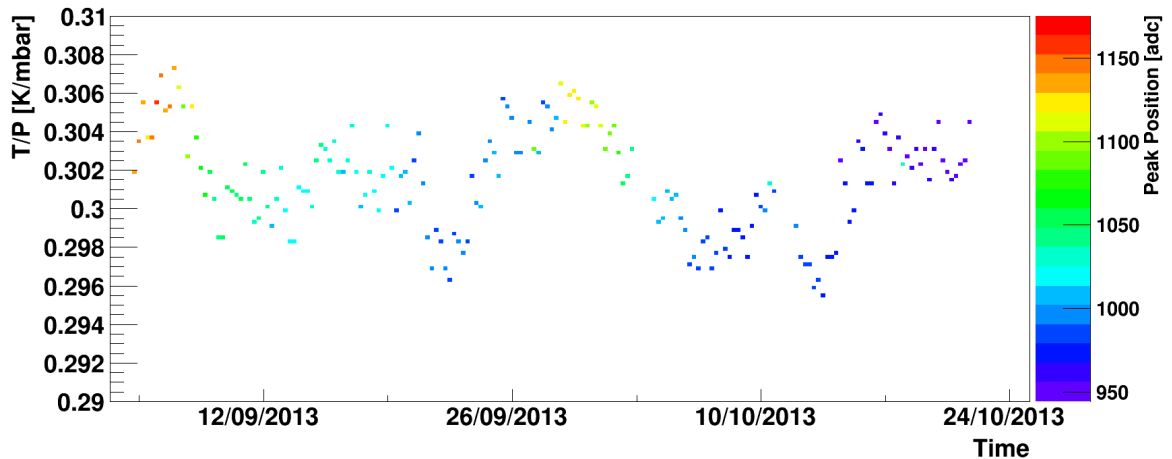


Figure 6.9: Peak Position (uncorrected peak mean (gain) (non normalized)) vs Time and T/P for the periods described in Figure 6.8.

In contrast with Figure 6.7 in Figure 6.9 vertical color stripes are present. By choosing a

constant value of  $T/P$  and reading horizontally one can observe the decrease of the peak position with respect to time repeated for both periods. This tells us that at any value for  $T/P$  the aging trend holds.

## 6.4. Conclusions

Single wire detectors are highly sensitive to environmental fluctuations such as changes in temperature and pressure. This is due to operation mechanism which uses the gases as an ionizing medium for detection of particles. The gas itself is what is sensitive to these environmental fluctuations. Nevertheless, the effects caused by these can be easily removed in their majority. First of all the fluctuations in a period where the gain seems to be constant can be eliminated in their majority by applying Equations 6.3 and 6.4 respectively. These equations were calculated by searching for the equations that minimize the standard deviation for a period with seemingly constant gain. When no aging effects are present the relation between gain and  $T/P$  can be compared to the theoretical model given in Equation 6.6. These are in close agreement since Figures 6.5 and 6.4 have clusters (corresponding to constant gain periods) that have high linear correlations. Finally by adding a time dimension into these plots two very interesting plots are created: Figures 6.7 and 6.9. These completely isolate environmental parameters by analyzing gain trends at specific values of  $T/P$ . Since the same trend is found for all values of  $T/P$  the conclusions with respect to aging periods or non-aging periods in SW2 hold strongly.

## 7. Integrated Charge

### 7.1. Introduction

Integrated charge refers to the accumulation of charge over time in the detector caused by current collection. It effectively describes the usage of a detector in a more absolute way than time. It is of interest to understand how much current a detector can handle before aging settles in rather than how much time has passed. For example, given two identical detectors, if detector 1 is used with a strong radioactive source and detector 2 with a weaker one, detector 1 will show aging first in time. Nevertheless these two detectors will show aging at the same integrated charge.

### 7.2. Results and Analysis

In contrast with Figure 5.10 the gain is plotted against integrated charge in 7.1.

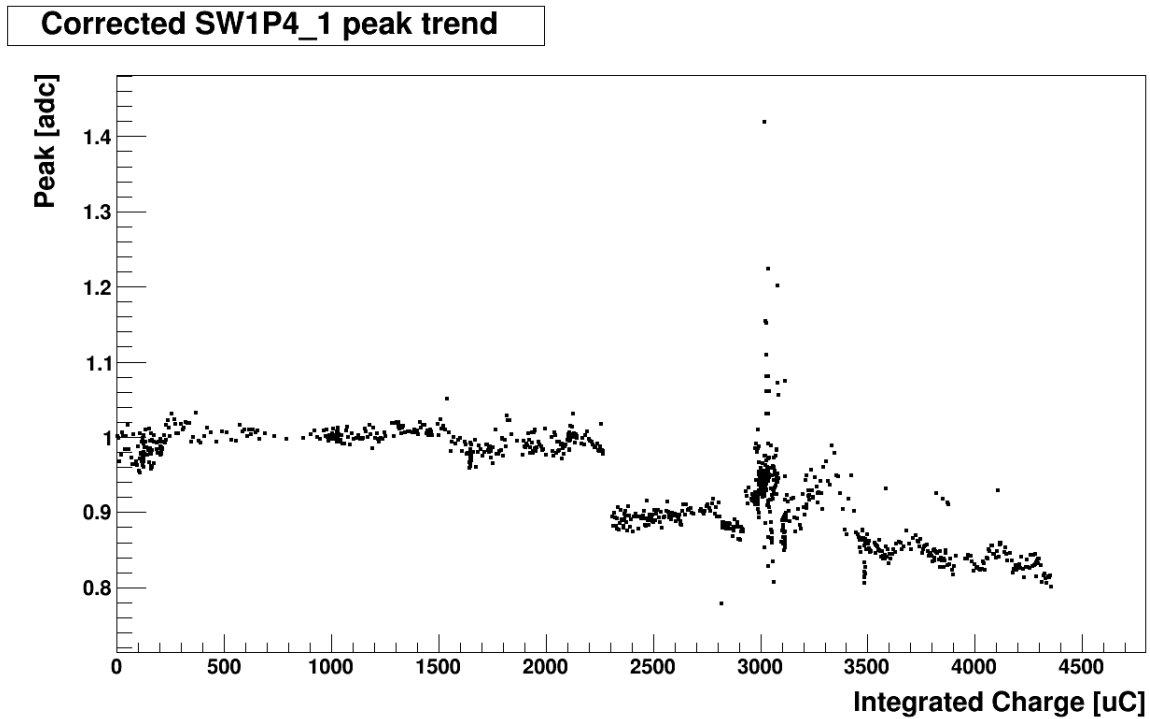


Figure 7.1: Corrected gain vs Integrated Charge for one year of detector operation of SW1

While the y values remain the same, there is a clear distortion in the x-axis since it now represents integrated charge which is a function of time. There is no clear advantage in doing this yet, however by doing the same for periods with aging these advantages become clear. Calculating the integrated charge for the heavily analyzed first two aging periods in SW2 results in Figures [7.2](#) and [7.3](#).

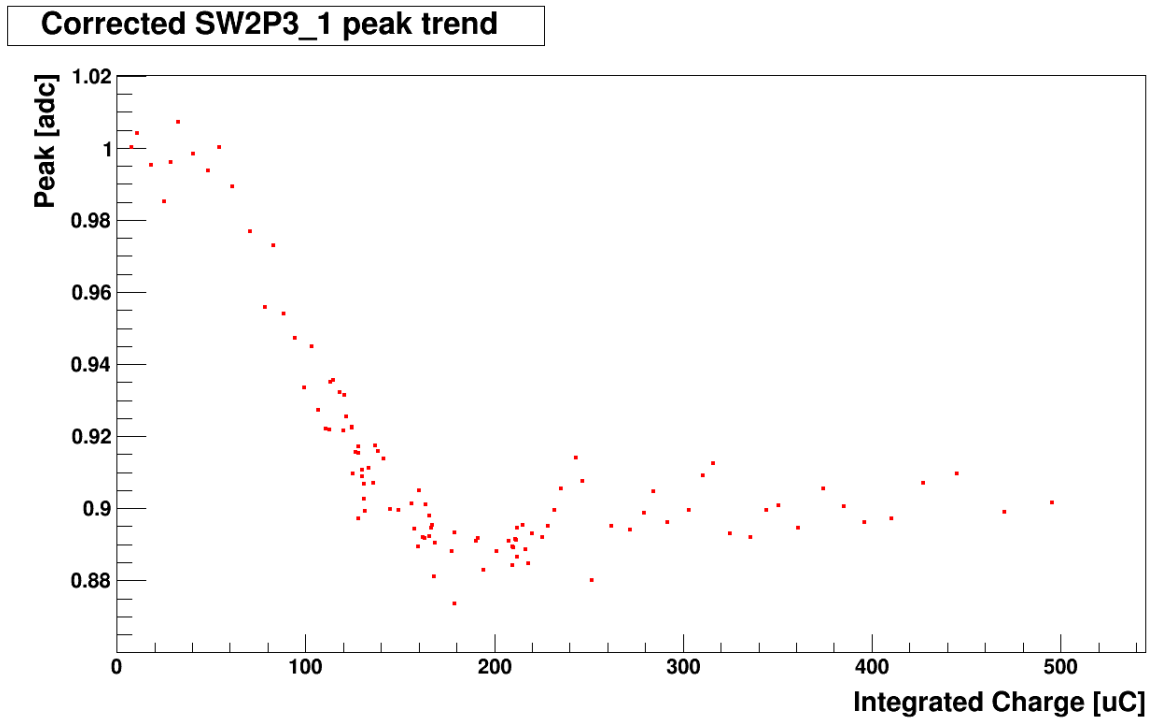


Figure 7.2: Corrected gain vs Integrated Charge for the period between the dates 1 and 2 in subSection3.1.2 for SW2 operation in position 3.



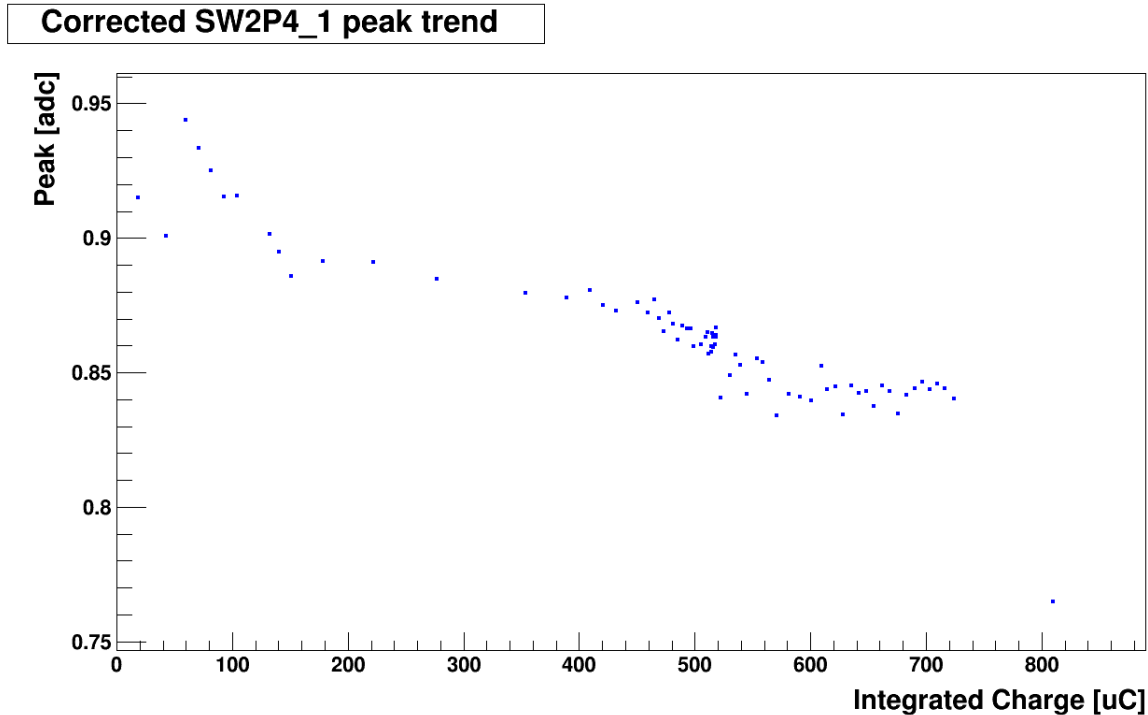


Figure 7.3: Corrected gain vs Integrated Charge for the period between the dates 2 and 3 in subSection 3.1.2 for SW2 operation in position 4.

These two Figures correspond to the periods in Figure 6.8. The difference is dramatic. When the gain is plotted against time the aging process settles in 16 days for the first period (detector in position 3) and 20 days for the second period (detector in position 4). Does this mean the aging process lasts 25% more in position 4? Not exactly. To compute this an absolute comparison parameter is needed. This is where the integrated charge comes in. Referring to Figures 7.2 and 7.3, the 16 days in the first period corresponds to an integrated charge of  $200 \mu C$  while in period the 20 days correspond to an integrated charge of  $600 \mu C$ , a 300% increase.

Dividing these integrated charges by the corresponding number of days an average current for each period is obtained:  $0.14 nA$  for the first period and  $0.35 nA$  for the second. Therefore, there are around 2.5 times more particles counted when the detector is in position 4 since the current generated in the wire is proportional to the incoming number of particles.

### 7.3. Conclusions

Integrated charge is a great tool to characterize the aging in single wire detectors. It is a parameter that allows comparison of aging effects between detectors subject to different radiation sources or a detector that's moved with respect to a single radiation source. Additionally the integrated charge provides information about the current in the wire and the total number of particles detected.

## 8. General Conclusions

Single wire detectors work on the principle of gas particle ionization and electron avalanche production to detect various types of ionizing radiation. Since they work in the proportional counter regime the energy of the detected particle can be easily extrapolated from the measured current. However, many factors can contribute to a change in the measured current. The gas mixture itself is an essential component of this. Environmental fluctuations, especially in temperature and pressure, can affect the density of the gas mixture in the detector changing the mean free path of collected electrons. These environmental parameters are corrected by using a simple minimization of the standard deviation of the gain fluctuations. By applying these corrections, the detector performances are well understood. Furthermore by carefully analyzing the effects of environmental T/P fluctuations on the gain the precise effects were understood over time. The most important outcome of these tests is the characterization of single wire detectors in terms of aging and environmental effects. The gain increases as T/P increases (giving a positive Townsend coefficient) in a consistent manner over time and that at stable T/P values aging effects are clearly observed. The single wire detector aging effects are due to the deposition of contaminants along the different lengths of the wire. The effects are clearly visible by moving the detector relative to the radioactive source as to expose wire sections that have not been irradiated before since the gas gain is restored. Supplementary to this time trend analysis one can plot all the variables with respect to integrated charge giving a more absolute perspective of the aging in the detectors.

## 9. Appendix

## 9.1. AmbSelect.C

```

1 //trend of enviromental parameter, current, corrected current, peak position
2 // plot for only ONE single wire -> choose SW from DataPeak line! (or use other script: PeakTrend_v?.C)
3 // output of enviromental parameter in AmbParCorr.dat
4
5 #include <string>
6 #include <stdlib>
7 #include <fstream>
8 #include <iostream>
9
10
11 void AmbSelect(){
12
13     //-----
14
15     gROOT->SetStyle("Plain");
16     // background is no longer mouse-dropping white
17     gStyle->SetCanvasColor(kWhite);
18     // blue to red false color palette. Use 9 for b/w
19     gStyle->SetPalette(1,0);
20     // turn off canvas borders
21     gStyle->SetCanvasBorderMode(0);
22     gStyle->SetPadBorderMode(0);
23     // What precision to put numbers if plotted with "TEXT"
24     gStyle->SetPaintTextFormat("5.2f");
25
26     // For publishing:
27     gStyle->SetLineWidth(1.5);
28     gStyle->SetTextSize(1.1);
29     gStyle->SetLabelSize(0.03,"xy");
30     gStyle->SetTitleSize(0.04,"xy");
31     gStyle->SetTitleOffset(1.1,"x");
32     gStyle->SetTitleOffset(0.9,"y");
33     gStyle->SetPadTopMargin(0.1);
34     gStyle->SetPadRightMargin(0.05);
35     gStyle->SetPadBottomMargin(0.1);
36     gStyle->SetPadLeftMargin(0.1);
37
38     //-----
39
40
41     //open file
42     //fstream check("C:/root/macros/SingleWire/OutputandPlots/check.out",ios::out);
43     fstream AmbParCorr("C:/root/macros/SingleWire/OutputandPlots/AmbParCorrfinal.dat",ios::out);
44     fstream DataCurrent("C:/root/macros/SingleWire/OutputandPlots/FilteredFineout.dat",ios::in); //merged file
45     of current and enviromental parameters
46     fstream DataPeak("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW1.txt",ios::in); //remember
47     to choose SW1 or SW2!!!
48
49     //file variables
50     int xcheck = 1;
51     float a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19;
52     float alnew;
53     float mean1, mean2, maximumX, FirstValue, integral, integralnoise;
54     float convYearPeak, convMonthPeak, convDayPeak, convHourPeak, convMinutePeak, convSecondPeak, a6fPeak,
55     a12fPeak, a15fPeak; //Bea
56     float
57     a1f[100000],a2f[100000],a3f[100000],a4f[100000],a5f[100000],a6f[100000],a7f[100000],a8f[100000],a9f[100000],
58     a10f[100000];
59     float
60     a11f[100000],a12f[100000],a13f[100000],a14f[100000],a15f[100000],a15f_test[100000],a16f[100000],a17f[100000],
61     a18f[100000],a19f[100000];
62     float alnewf[100000];
63     float CorrFact[100000];
64     float CorrCurr[100000];
65     float PeakADC[100000], PeakADCtmp;
66     float P0 = 970.; //mbar
67     float T0 = 293.; //K
68
69     //time variables
70     char cday[30],cmonth[30],cyear[30],chour[30],cmin[30],csec[30];
71     int day,month,year,hour,min,sec;
72     TDateTime t;
73     UInt_t ctime;
74     int convYear[100000], convMonth[100000], convDay[100000]; //boh
75     int convHour[100000], convMinute[100000], convSecond[100000]; //boh
76     int pTime;
77     float Time, Timef[100000], Timeg[100000];
78
79     //plot variables
80     TCanvas *c[100];
81     TGraph *gr[100];
82     TLegend *Legend;
83
84     char ctitle[30];
85     char titlegif[60];

```

```

79 char title[60],title2[60],titleday[60];
80
81 //?
82 int iplt;
83 int i=0;
84 int oldHour = 0;
85
86 while(!DataCurrent.eof())
87 {
88     DataCurrent >> alnew >> a6 >> a7 >> a9 >> a12 >> a15 >> a17 >> a18;
89
90     Timef[i] = alnew;
91     t.Set(alnew);
92     convYear[i] = t.GetYear(); convMonth[i] = t.GetMonth(); convDay[i] = t.GetDay();
93     convHour[i] = t.GetHour(); convMinute[i] = t.GetMinute(); convSecond[i] = t.GetSecond();
94     /* if (i<10) { */
95     /* cout << alnew << "\t" << convYear[i] << "\t" << convMonth[i] << "\t" << convDay[i] << endl;
... */
96     /* cout << convHour[i] << "\t" << convMinute[i] << "\t" << convSecond[i] << endl; */
97     /* } */
98     //T1
99     a6f[i] = a6;
100    //T2
101    a7f[i] = a7;
102    //I (mA)
103    a9f[i] = a9 * 1000.;
104    //dewp
105    a12f[i] = 7.5*(a12*1000)-90.;
106    //Pabs
107    a15f[i] = ((0.075*(1000*a17))-0.3)*1000.; //Prel
108    a17f[i] = 3.125*(a15*1000)-37.5;
109    //O2
110    a18f[i] = 6.25*(a18*1000)-25;
111    //CorrectionFactor
112    CorrFact[i]= ((273.0+a7f[i])/T0)*(P0/a15f[i]);
113    //CorrectionFactor
114    CorrCurr[i]= a9f[i] * CorrFact[i] * CorrFact[i];
115    if (convHour[i] != oldHour && convHour[i]%6==0) //boh
116    {
117        AmbParCorr << convYear[i] << "\t" << convMonth[i] << "\t" << convDay[i] << "\t" << //boh
118        convHour[i] << "\t" << convMinute[i] << "\t" << convSecond[i] << "\t" << a6f[i] << "\t" << a15f[i]
... << "\t" << a18f[i] << "\t" << a12f[i] << endl;
119        oldHour = convHour[i];
120
121    }
122    i++;
123 }
124
125 int npt = i-1;
126
127 /***Start Plots***
128
129 //a6->T1
130 iplt=1;
131 sprintf(title, "plt[%d]",iplt);
132 sprintf(title2, " plt %d",iplt);
133 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
134
135 c[iplt]->SetFillColor(0);
136 c[iplt]->GetFrame()->SetBorderSize(0);
137 gr[iplt] = new TGraph(npt, Timef, a6f);
138
139 gr[iplt]->SetMarkerSize(0.75);
140 gr[iplt]->SetMarkerStyle(21);
141 gr[iplt]->SetMarkerColor(1);
142
143 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
144
145 cout << "time" << Timef[npt-1] << endl;
146
147 gr[iplt]->SetTitle(title);
148 gr[iplt]->GetXaxis()->SetTitle("");
149 gr[iplt]->GetYaxis()->SetTitle("T1 (C)");
150
151 gr[iplt]->GetXaxis()->SetNdivisions(605);
152 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
153 gr[iplt]->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
154 gr[iplt]->Draw("AP");
155 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/T1.jpg");
156
157 //a7->T2
158 iplt=2;
159 sprintf(title, "plt[%d]",iplt);
160 sprintf(title2, " plt %d",iplt);
161 c[iplt] = new TCanvas(title,title2,10,10,1300,800);

```

```

162 c[iplt]->SetFillColor(0);
163 c[iplt]->GetFrame()->SetBorderSize(0);
164 c[iplt]->GetFrame()->SetBorderSize(0);
165 gr[iplt] = new TGraph(npt, Timef, a7f);
166
167 gr[iplt]->SetMarkerSize(0.75);
168 gr[iplt]->SetMarkerStyle(21);
169 gr[iplt]->SetMarkerColor(1);
170
171 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
172
173 gr[iplt]->SetTitle(title);
174 gr[iplt]->GetXaxis()->SetTitle("");
175 gr[iplt]->GetYaxis()->SetTitle("T2 (C)");
176
177 gr[iplt]->GetXaxis()->SetNdivisions(605);
178 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
179 gr[iplt]->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
180 gr[iplt]->Draw("AP");
181 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/T2.jpg");
182
183 //a9->I
184 iplt=3;
185 sprintf(title, "plt[%d]",iplt);
186 sprintf(title2, " plt %d",iplt);
187 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
188
189 c[iplt]->SetFillColor(0);
190 c[iplt]->GetFrame()->SetBorderSize(0);
191 gr[iplt] = new TGraph(npt, Timef, a9f);
192
193 gr[iplt]->SetMarkerSize(0.75);
194 gr[iplt]->SetMarkerStyle(21);
195 gr[iplt]->SetMarkerColor(1);
196
197 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
198
199 gr[iplt]->SetTitle(title);
200 gr[iplt]->GetXaxis()->SetTitle("");
201 gr[iplt]->GetYaxis()->SetTitle("I (mA)");
202
203 gr[iplt]->GetXaxis()->SetNdivisions(605);
204 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
205 gr[iplt]->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
206 gr[iplt]->Draw("AP");
207 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/Current.jpg");
208
209 //a12->dewp
210 iplt=4;
211 sprintf(title, "plt[%d]",iplt);
212 sprintf(title2, " plt %d",iplt);
213 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
214
215 c[iplt]->SetFillColor(0);
216 c[iplt]->GetFrame()->SetBorderSize(0);
217 gr[iplt] = new TGraph(npt, Timef, a12f);
218
219 gr[iplt]->SetMarkerSize(0.75);
220 gr[iplt]->SetMarkerStyle(21);
221 gr[iplt]->SetMarkerColor(1);
222
223 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
224
225 gr[iplt]->SetTitle(title);
226 gr[iplt]->GetXaxis()->SetTitle("");
227 gr[iplt]->GetYaxis()->SetTitle("dewpoint (C)");
228
229 gr[iplt]->GetXaxis()->SetNdivisions(605);
230 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
231 gr[iplt]->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
232 gr[iplt]->Draw("AP");
233 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/dewpoint.jpg");
234
235 //a15->Pabs
236 iplt=5;
237 sprintf(title, "plt[%d]",iplt);
238 sprintf(title2, " plt %d",iplt);
239 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
240
241 c[iplt]->SetFillColor(0);
242 c[iplt]->GetFrame()->SetBorderSize(0);
243 gr[iplt] = new TGraph(npt, Timef, a15f);
244
245 gr[iplt]->SetMarkerSize(0.75);
246 gr[iplt]->SetMarkerStyle(21);

```

```

247 gr[iplt]->SetMarkerColor(1);
248
249 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
250
251 gr[iplt]->SetTitle(title);
252 gr[iplt]->GetXaxis()->SetTitle("");
253 gr[iplt]->GetYaxis()->SetTitle("Pabs (mbar)");
254
255 gr[iplt]->GetXaxis()->SetNdivisions(605);
256 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
257 gr[iplt]->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
258 gr[iplt]->Draw("AP");
259 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/Pabs.jpg");
260
261 //a17->Prel
262 iplt=6;
263 sprintf(title, "plt[%d]",iplt);
264 sprintf(title2, " plt %d",iplt);
265 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
266
267 c[iplt]->SetFillColor(0);
268 c[iplt]->GetFrame()->SetBorderSize(0);
269 gr[iplt] = new TGraph(npt, Timef, a17f);
270
271 gr[iplt]->SetMarkerSize(0.75);
272 gr[iplt]->SetMarkerStyle(21);
273 gr[iplt]->SetMarkerColor(1);
274
275 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
276
277 gr[iplt]->SetTitle(title);
278 gr[iplt]->GetXaxis()->SetTitle("");
279 gr[iplt]->GetYaxis()->SetTitle("Prel (mbar)");
280
281 gr[iplt]->GetXaxis()->SetNdivisions(605);
282 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
283 gr[iplt]->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
284 gr[iplt]->Draw("AP");
285 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/Prel.jpg");
286
287 //a18->O2
288 iplt=7;
289 sprintf(title, "plt[%d]",iplt);
290 sprintf(title2, " plt %d",iplt);
291 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
292
293 c[iplt]->SetFillColor(0);
294 c[iplt]->GetFrame()->SetBorderSize(0);
295 gr[iplt] = new TGraph(npt, Timef, a18f);
296
297 gr[iplt]->SetMarkerSize(0.75);
298 gr[iplt]->SetMarkerStyle(21);
299 gr[iplt]->SetMarkerColor(1);
300
301 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
302
303 gr[iplt]->SetTitle(title);
304 gr[iplt]->GetXaxis()->SetTitle("");
305 gr[iplt]->GetYaxis()->SetTitle("O2 (ppm)");
306
307 gr[iplt]->GetXaxis()->SetNdivisions(605);
308 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
309 gr[iplt]->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
310 gr[iplt]->Draw("AP");
311 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/O2.jpg");
312
313 //CorrFact
314 iplt=10;
315 sprintf(title, "plt[%d]",iplt);
316 sprintf(title2, " plt %d",iplt);
317 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
318
319 c[iplt]->SetFillColor(0);
320 c[iplt]->GetFrame()->SetBorderSize(0);
321 gr[iplt] = new TGraph(npt, Timef, CorrFact);
322
323 gr[iplt]->SetMarkerSize(0.75);
324 gr[iplt]->SetMarkerStyle(21);
325 gr[iplt]->SetMarkerColor(1);
326
327 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]),1.0001*(Timef[npt-1]));
328
329 gr[iplt]->SetTitle(title);
330 gr[iplt]->GetXaxis()->SetTitle("");
331 gr[iplt]->GetYaxis()->SetTitle("CorrFact[i]");

```



```

332
333 gr[iplt]->GetXaxis()->SetNdivisions(605);
334 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
335 gr[iplt]->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
336 gr[iplt]->Draw("AP");
337 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/CorrFact.jpg");
338
339 //CorrCurrent
340 iplt=11;
341 sprintf(title, "plt[%d]", iplt);
342 sprintf(title2, " plt %d", iplt);
343 c[iplt] = new TCanvas(title, title2, 10, 10, 1300, 800);
344
345 c[iplt]->SetFillColor(0);
346 c[iplt]->GetFrame()->SetBorderSize(0);
347 gr[iplt] = new TGraph(npt, Timef, CorrCurr);
348
349 gr[iplt]->SetMarkerSize(0.75);
350 gr[iplt]->SetMarkerStyle(21);
351 gr[iplt]->SetMarkerColor(1);
352
353 gr[iplt]->GetXaxis()->SetLimits(0.9999*(Timef[1]), 1.0001*(Timef[npt-1]));
354
355 gr[iplt]->SetTitle(title);
356 gr[iplt]->GetXaxis()->SetTitle("");
357 gr[iplt]->GetYaxis()->SetTitle("Corr Curr (mA)");
358
359 gr[iplt]->GetXaxis()->SetNdivisions(605);
360 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
361 gr[iplt]->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
362 gr[iplt]->Draw("AP");
363 //c[iplt]->Print("/Applications/root/macros/swc256/jpeg/CorrCurr.jpg");
364
365 //***End Plots***
366
367 //Peak SW1
368 int i=0;
369 int j=0;
370 /*
371 AmbParCorr.close();
372 fstream AmbParCorr2("/Applications/root/macros/SW/OutputAndPlots/AmbParCorr2.dat", ios::in); //boh
373 while(!DataPeak.eof() && !AmbParCorr.eof()){ //Bea
374
375     //DataPeak >> mean1 >> mean2 >> maximumX >> hour >> day >> month >> year; //old settings
376     DataPeak >> mean1 >> mean2 >> maximumX >> integral >> integralnoise >> hour >> day >> month >> year;
377 //new settings
378     year= 2000 + year; //spostata qui da sotto
379
380     if (i==0) {
381         AmbParCorr2 >> convYearPeak >> convMonthPeak >> convDayPeak >> convHourPeak >> convMinutePeak
382 >> convSecondPeak >> a6fPeak >> a12fPeak >> a15fPeak; //Bea
383         cout << " ambi " << convYearPeak << "\t" << convMonthPeak << "\t" << convDayPeak << "\t" << convHourPeak
384 << "\t" << endl ;
385     }
386
387     if (j==1) {
388         //q-if (convMonthPeak != month && convDayPeak != day && convHourPeak != hour) {
389         while (convMonthPeak != month || convDayPeak != day || convHourPeak != hour) {
390             AmbParCorr2 >> convYearPeak >> convMonthPeak >> convDayPeak >> convHourPeak >> convMinutePeak
391 >> convSecondPeak >> a6fPeak >> a12fPeak >> a15fPeak; //Bea
392         }
393
394         cout << "peakw " << year << "\t" << month << "\t" << day << "\t" << hour << endl ;
395         cout << "ambiw " << convYearPeak << "\t" << convMonthPeak << "\t" << convDayPeak << "\t" << convHourPeak <<
396 endl ;
397         cout << endl;
398
399         //CorrectionFactor
400         CorrFact[i]= ((273.0+a6fPeak)/T0)*(P0/(-3.3*a15fPeak));
401
402         PeakADctmp = maximumX;
403         maximumX = maximumX;// / (CorrFact[i] * CorrFact[i] * CorrFact[i] * CorrFact[i] * CorrFact[i]);
404
405         cout << maximumX << "\t" << PeakADctmp << "\t" << CorrFact[i] << "\t" << a6fPeak << "\t" << a15fPeak
406 << endl;
407
408         //q-}
409         //
410     }
411
412     //year= 2000 + year; spostata sopra
413     TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
414     UInt_t ttt;
415     ttt = date->Convert();

```

```

411     pTime = (int)ttt;
412     Timeg[i] = (float)pTime;
413
414     if (i==0) {
415         FirstValue = maximumX; //define the first value for the normalization
416         PeakADC[i]=maximumX/FirstValue;
417         //cout << i << "\t" << PeakADC[i] << endl;
418     }
419     if (i>0) {
420         PeakADC[i] = maximumX/FirstValue;
421     }
422     i++;
423 }
424
425 int nptP = i-1;
426
427 //PeakPlot SW1
428 iplt=12;
429 sprintf(title, "plt[%d]",iplt);
430 sprintf(title2, " plt %d",iplt);
431 c[iplt] = new TCanvas(title,title2,10,10,1300,800);
432
433 c[iplt]->SetFillColor(0);
434 c[iplt]->GetFrame()->SetBorderSize(0);
435 gr[iplt] = new TGraph(nptP, Timeg, PeakADC);
436
437 gr[iplt]->SetMarkerSize(0.75);
438 gr[iplt]->SetMarkerStyle(21);
439 gr[iplt]->SetMarkerColor(1);
440
441 gr[iplt]->SetTitle(title);
442 gr[iplt]->GetXaxis()->SetTitle("");
443 gr[iplt]->GetYaxis()->SetTitle("Peak(adc)");
444
445 gr[iplt]->GetXaxis()->SetNdivisions(605);
446 gr[iplt]->GetXaxis()->SetTimeDisplay(1);
447 gr[iplt]->GetXaxis()->SetTimeFormat("%d\/%m\/%Y %F1970-01-01 00:00:00");
448 gr[iplt]->Draw("AP");
449
450
451 //Time division lines
452
453 year = 2013;
454 month = 10;
455 day = 31;
456 hour = 0; min = 0; sec = 0;
457 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
458 UInt_t ttt;
459 ttt = date->Convert();
460 pTime = (int)ttt;
461 float Time1 = (float)pTime;
462
463 c[iplt]->Update();
464 TLine *l=new TLine(Time1,c[iplt]->GetUymin(),Time1,c[iplt]->GetUymax());
465 l->SetLineColor(32);
466 l->SetLineWidth(2);
467 l->SetLineStyle(5);
468 l->Draw();
469
470 //Finish time line division
471
472 c[iplt]->Print("/Applications/root/macros/SW/OutputAndPlots/Peak_adc.jpg");
473 */
474 }
475

```

## 9.2. `chisqtest.C` (Lines 149-205, 287-367 [[19](#)])

```

1 #include <fstream>
2 bool eof();
3
4 void chisqtestTP()
5 {
6 //-----
7
8 gROOT->SetStyle("Plain");
9 // background is no longer mouse-dropping white
10 gStyle->SetCanvasColor(kWhite);
11 // blue to red false color palette. Use 9 for b/w
12 gStyle->SetPalette(1,0);
13 // turn off canvas borders
14 gStyle->SetCanvasBorderMode(0);
15 gStyle->SetPadBorderMode(0);
16 // What precision to put numbers if plotted with "TEXT"
17 gStyle->SetPaintTextFormat("5.2f");
18
19 // For publishing:
20 gStyle->SetLineWidth(1.5);
21 gStyle->SetTextSize(1.1);
22 gStyle->SetLabelSize(0.05,"xy");
23 gStyle->SetTitleSize(0.05,"xy");
24 gStyle->SetTitleOffset(1.1,"x");
25 gStyle->SetTitleOffset(0.9,"y");
26 gStyle->SetPadTopMargin(0.1);
27 gStyle->SetPadRightMargin(0.1);
28 gStyle->SetPadBottomMargin(0.16);
29 gStyle->SetPadLeftMargin(0.12);
30
31 //-----
32
33 int SingleWire;
34 cout << "SingleWire 1 or 2? ";
35 cin >> SingleWire;
36 if (SingleWire==1)
37 {
38 //fstream alphabeta("C:/root/macros/SingleWire/OutputandPlots/alphabetaSW1.dat", ios::out );
39 //fstream filesused("C:/root/macros/SingleWire/OutputandPlots/FilesUsedSW1.dat", ios::out );
40 //fstream alphabeta1("C:/root/macros/SingleWire/OutputandPlots/alphabeta1SW1.dat", ios::out );
41 //fstream gaincorr("C:/root/macros/SingleWire/OutputandPlots/gaincorrSW1.dat", ios::out );
42 //ifstream ftime("/home/daqrpc/daq-1.0.0/Scan_v0/Backup_pccms2/CAENHVWrapper_2_4/RPCTest/ftime_thp.tmp0",
... ios::in);
43
44 ifstream fgain1("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW1clean.txt", ios::in);//These
... two have to be the same file
45 ifstream fgainP("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW1clean.txt", ios::in);//These
... two have to be the same file
46 }
47 if (SingleWire==2)
48 {
49 //fstream alphabeta("C:/root/macros/SingleWire/OutputandPlots/alphabetaSW2.dat", ios::out );
50 //fstream filesused("C:/root/macros/SingleWire/OutputandPlots/FilesUsedSW2.dat", ios::out );
51 //fstream alphabeta1("C:/root/macros/SingleWire/OutputandPlots/alphabeta1SW2.dat", ios::out );
52 //fstream gaincorr("C:/root/macros/SingleWire/OutputandPlots/gaincorrSW2.dat", ios::out );
53 //ifstream ftime("/home/daqrpc/daq-1.0.0/Scan_v0/Backup_pccms2/CAENHVWrapper_2_4/RPCTest/ftime_thp.tmp0",
... ios::in);
54
55 ifstream fgain1("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW2clean.txt", ios::in);//These
... two have to be the same file
56 ifstream fgainP("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW2clean.txt", ios::in);//These
... two have to be the same file
57 }
58 if (SingleWire!=1 && SingleWire!=2) {cout<< "Selection has to be 1 or 2" << endl;}
59
60 ifstream famb("C:/root/macros/SingleWire/OutputandPlots/AmbParCorrfinal.dat", ios::in);
61 //ifstream fhv[50];
62
63
64 int i;
65 //int startday,stopday;
66 //float iMax;
67 float Avegain1[50];
68 //Int_t idx[50], tmpidx;
69 int idx;
70 char title[256];
71
72 TCanvas *c[50];
73 TGraph *grc[50], *grcc[50];
74 TGraph *grhv[50];
75
76 float FWHM[10000];
77 float gain1[10000],hv[50][10000]; //gain1[50][10000]
78 float dt[10000];
79 int dt2[10000],dt_month[10000],dt_day[10000],dt_year[10000],dt_hour[10000],dt_min,dt_sec;

```

```

80 //Int_t pTime[10000];
81 float Time[10000];
82
83 float Tin[10000], Patm[10000], dewP[10000], O2[10000];
84 float Tsinc[10000], Psinc[10000];
85
86 float T0=0;
87 float P0=0;
88 float Tcorr, Pcorr; //RHcorr;
89 Int_t idx_T0, idx_P0;
90 Int_t alpha, beta, alphaMax, betaMax;
91 Int_t best_alpha, best_beta;
92
93 float sigma[50][50]; //alpha, beta, gap
94 float minsigma; //alpha, beta, gap
95 float gain1_corr[50][50][10000]; //alpha, beta, gap, time
96 //float gain1_i2_corr[50][50][10000]; //alpha, beta, gap, time
97 float Avegain1_corr[50][50]; //alpha, beta, gap
98 float gain1_bestcorr[50][10000]; //best correction
99
100 for(int filli=0; filli<50; filli++)
101 {
102     for(int fillj=0; fillj<50; fillj++)
103     {
104         Avegain1_corr[filli][fillj]=0;
105         sigma[filli][fillj]=0;
106     }
107 }
108
109 //Int_t iday_i1, iday_i2;
110
111 //TCanvas *c[50];
112 //TGraph *grc[50];
113
114 //float Tmin, TMax, Patmmin, PatmMax;
115
116 //TF1 *fGraph[50];
117 //Float_t nP0_reg1, nPOE_reg1;
118
119 //Int_t ITimeZona1, ITimeZona2, ITimeZona3;
120
121 //Tmin= 17;
122 //TMax = 25;
123 //Patmmin = 930;
124 //PatmMax = 1000;
125 //iMax = 60;
126
127 alphaMax = 49;
128 betaMax = 49;
129
130 i = 0;
131 while(!famb.eof())
132 {
133     famb >> dt_year[i] >> dt_month[i] >> dt_day[i] >> dt_hour[i] >> dt_min >> dt_sec >> Tin[i] >> Patm[i] >>
134     O2[i] >> dewP[i];
135
136     //if (i<150) cout<< dt_year[i] <<endl;
137
138     //dt[i] = float(dt2[i]);
139
140     //check << i << "\t" << dt[i] << endl;
141     /*
142     TDate * date = new TDate(dt_year[i], dt_month[i], dt_day[i], dt_hour[i], dt_min, dt_sec);
143     UInt_t ttt;
144     ttt = date->Convert();
145     pTime[i] = (int)ttt;
146     Time[i] = (float)pTime[i];
147     */
148     //THP SHOULD THIS BE INCLUDED????????
149     Patm[i] = Patm[i] + 4.0; //offset sensore GIF (to be checked!)
150     if(Tin[i] < 10)
151     {
152         Tin[i] = 10 ;
153         //RHin[i]= 30 ;
154         //Tout[i]= 10 ;
155         //RHout[i]= 30;
156         Patm[i] = 1000;
157     }
158
159     //T0, RH0, P0, ... average value
160     if(Tin[i]>10)
161     {
162         idx_T0++;
163         T0 = T0 + Tin[i];

```

```

164     }
165     if(Patm[i]>900)
166     {
167         idx_P0++;
168         P0 = P0 + Patm[i];
169     }
170
171     //fine THP
172     //check << dt[i] << " " << dt_month <<" " << dt_day << endl;
173     //if(i==0) startday = Time[i];//dt[i];
174     //if(i>0) stopday = Time[i-1];//dt[i-1];
175     i++;
176 }
177
178
179 T0 = T0 / idx_T0;
180 P0 = P0 / idx_P0;
181 cout << T0 << "\t" << P0 << endl;
182
183 /*
184 int days = i;
185 check << "start stop day "
186     << startday << "\t" << stopday << "\t" << days << "\n";
187
188 for(int k = 0; k <= days; k++)
189 {
190     sPatm[k] = (((Patm[k]-Patmmin)/(PatmMax-Patmmin))*(TMax-Tmin))+Tmin;
191 }
192
193 int i1,i2;
194
195 // for(int ichn=1;ichn<12;ichn++){ //era <8 comm.10/03/2011
196 for(int ichn=4;ichn<5;ichn++)
197 { //era <8
198     i1 = 2*ichn - 1;
199     i2 = 2*ichn;
200
201     if(ichn == 9){
202         i1 = 17;
203         i2 = 20;
204     }
205 */
206 i = 0;
207 int j0 = 0; //
208 int iday;
209
210 float trash;
211 float firstgain;
212 int checkhour, checkday, checkmonth, checkyear;
213 int yearP, monthP, dayP, hourP, minP, secP;
214 float periodstart, periodend, periodday;
215 int pass;
216 int nused=0;
217 idx = 0;
218
219 // *****Main loop needed for chi squared
... reduction to find alpha and beta*****
220 //I removed the indices [i1] to uncompllicate things... basically extracting this loop from the one it is in
... (line 221)
221
222 while(!fgain1.eof())
223 {
224     j0=0;
225     fgain1 >> trash >> trash >> gain1[i] >> trash >> trash >> FWHM[i] >> trash >> trash >> trash >> trash
... >> checkhour >> checkday >> checkmonth >> checkyear;
226     checkyear=2000+checkyear;
227
228     //if(i==0) idx = 0;
229     if (i==0)
230     {
231         firstgain=gain1[i];
232         gain1[i]=gain1[i]/firstgain;
233     }
234     if (i>0) gain1[i]=gain1[i]/firstgain;
235     //cout<< gain1[i] << endl;
236     //if(i>20) break;
237
238     while(checkhour!=dt_hour[j0] || checkday!=dt_day[j0] || checkmonth!=dt_month[j0] ||
... checkyear!=dt_year[j0] )
239     {
240         j0++;
241         if(j0>9999)
242         {
243             pass=0;
244             break;

```

```

245     }
246     else pass=1;
247 }
248 if(checkhour==dt_hour[j0] && checkday!=dt_day[j0] && checkmonth==dt_month[j0] &&
... checkyear==dt_year[j0]) pass=1;
249
250
251
252 //*****Period
... selector*****
253 yearP = 2013;
254 monthP = 10;
255 dayP = 1;
256 hourP = 0; minP = 0; secP = 0;
257 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
258 UInt_t ttt;
259 ttt = dateP->Convert();
260 pTimeP = (int)ttt;
261 periodstart = (float)pTimeP;
262
263 yearP = 2013;
264 monthP = 11;
265 dayP = 1;
266 hourP = 0; minP = 0; secP = 0;
267 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
268 UInt_t ttt;
269 ttt = dateP->Convert();
270 pTimeP = (int)ttt;
271 periodend = (float)pTimeP;
272
273 yearP = checkyear;
274 monthP = checkmonth;
275 dayP = checkday;
276 hourP = 0; minP = 0; secP = 0;
277 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
278 UInt_t ttt;
279 ttt = dateP->Convert();
280 pTimeP = (int)ttt;
281 periodday = (float)pTimeP;
282
283 if(periodday<periodstart || periodday>periodend) pass=0;
284
... //*****
... *****
285 if (pass==1)
286 {
287     for(int alpha=0; alpha< alphaMax; alpha++)
288     {
289         for(int beta=0; beta< betaMax; beta++)
290         {
291             Tcorr = pow(((Tin[j0]+273.1)/(T0+273.1)),-0.1*alpha);
292             Pcorr = pow((P0/Patm[j0]),-0.1*beta);
293             gain1_corr[alpha][beta][nused] = Tcorr * Pcorr * gain1[i];
294             //cout<< gain1[i] << " " << gain1_corr[alpha][beta][i] << endl;
295             if(alpha == 0 && beta == 0) idx++;
296             //Avegain1_corr[alpha][beta]++;
297             Avegain1_corr[alpha][beta] = Avegain1_corr[alpha][beta] + gain1_corr[alpha][beta][nused];
298             //cout << i << " " << alpha << " " << beta << " " << Avegain1_corr[alpha][beta]
... << " " << gain1_corr[alpha][beta][nused]<< " " << idx << endl;
299         }
300     }
301     //filesused << i << "\t" <<nused << "\t" << checkyear << "\t" << checkmonth << "\t" << checkday <<
... "\t" << checkhour << endl;
302     nused++; //i++;
303     //j0++;
304     //if (i>2) break;
305 }
306 i++;
307 }
308
309 iday = nused - 1;
310 //
311 //stopday = Time[iday_i1];
312
313
314 for(int alpha=0; alpha< alphaMax; alpha++)
315 {
316     for(int beta=0; beta< betaMax; beta++)
317     {
318         Avegain1_corr[alpha][beta] = Avegain1_corr[alpha][beta] / idx;
319         //cout << i << " " << alpha << " " << beta << " " << Avegain1_corr[alpha][beta] <<
... endl;
320     }
321 }
322

```

```

323     for(int j=0; j < iday; j++)
324     {
325         for(int alpha=0; alpha< alphaMax; alpha++)
326         {
327             for(int beta=0; beta< betaMax; beta++)
328             {
329                 if(gain1[j] > 0)
330                     sigma[alpha][beta] = sigma[alpha][beta] + pow((gain1_corr[alpha][beta][j] -
... Avegain1_corr[alpha][beta]), 2);
331                     //cout << iday << "      " << alpha << "      " << beta << "      " << sigma[alpha][beta] << endl;
332             }
333         }
334     }
335 }
336
337
338
339
340
341     for(int alpha=0; alpha< alphaMax; alpha++)
342     {
343         for(int beta=0; beta< betaMax; beta++)
344         {
345             sigma[alpha][beta] = sigma[alpha][beta]/ (iday-1);
346             sigma[alpha][beta] = pow(sigma[alpha][beta], 0.5);
347             //alphabet << alpha << "\t" << beta << "\t" << sigma[alpha][beta] << endl;
348
349             if(alpha == 0 && beta == 0)
350             {
351                 minsigma = sigma[alpha][beta];
352                 best_alpha = alpha;
353                 best_beta = beta;
354             }
355
356             //alphabet1 << alpha << "\t" << beta << "\t" << sigma[alpha][beta] << "\t" << minsigma << "\t" <<
... endl;
357
358             if(sigma[alpha][beta] < minsigma)
359             {
360                 best_alpha = alpha;
361                 best_beta = beta;
362                 minsigma = sigma[alpha][beta];
363             }
364         }
365     }
366 }
367     cout << best_alpha << "      " << best_beta << endl;
368
369     float gain_corrP;
370     int iP=0;
371     float gainP[10000];
372     float gainTP[10000], gainTP1[10000], gainTP2[10000], gainTP3[10000], gainTPall[10000];
373     float TP[10000], TP1[10000], TP2[10000], TP3[10000], TPall[10000];
374     float timeTP[10000];
375     nused=0;
376     int selectedperiod;
377     int tp=0;
378     int tp1=0;
379     int tp2=0;
380     int tp3=0;
381     int year, month, day, hour, min, sec;
382     TH2F *TPPeak;
383     TPPeak = new TH2F("TPpeak", "", 50,0.29,0.31,50,850,1400);
384     TH2F *TPPeakall;
385     TPPeakall = new TH2F("TPpeak", "", 25,0.29,0.31,25,850,1400);
386     TH2F *TPtimeG;
387     TPtimeG = new TH2F("TPG", "", 1620,1375000000,1410000000,100,0.29,0.31);
388
389
390     //Periods for gain vs T/P plots
391
392     float Time1[7];
393     year = 2013;
394     month = 10;
395     day = 1;
396     hour = 0; min = 0; sec = 0;
397     TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
398     UInt_t ttt;
399     ttt = date->Convert();
400     pTime = (int)ttt;
401     Time1[0] = (float)pTime;
402
403     year = 2013;
404     month = 11;
405     day = 1;

```



```

406 hour = 0; min = 0; sec = 0;
407 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
408 UInt_t ttt;
409 ttt = date->Convert();
410 pTime = (int)ttt;
411 Time1[1] = (float)pTime;
412
413 year = 2013;
414 month = 11;
415 day = 21;
416 hour = 0; min = 0; sec = 0;
417 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
418 UInt_t ttt;
419 ttt = date->Convert();
420 pTime = (int)ttt;
421 Time1[2] = (float)pTime;
422
423 year = 2014;
424 month = 1;
425 day = 7;
426 hour = 0; min = 0; sec = 0;
427 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
428 UInt_t ttt;
429 ttt = date->Convert();
430 pTime = (int)ttt;
431 Time1[3] = (float)pTime;
432
433 year = 2014;
434 month = 2;
435 day = 27;
436 hour = 0; min = 0; sec = 0;
437 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
438 UInt_t ttt;
439 ttt = date->Convert();
440 pTime = (int)ttt;
441 Time1[4] = (float)pTime;
442
443 year = 2014;
444 month = 4;
445 day = 28;
446 hour = 0; min = 0; sec = 0;
447 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
448 UInt_t ttt;
449 ttt = date->Convert();
450 pTime = (int)ttt;
451 Time1[5] = (float)pTime;
452
453 year = 2014;
454 month = 5;
455 day = 28;
456 hour = 0; min = 0; sec = 0;
457 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
458 UInt_t ttt;
459 ttt = date->Convert();
460 pTime = (int)ttt;
461 Time1[6] = (float)pTime;
462
463 //*****
464 //Print corrected gain alongside uncorrected.
465 while(!fgainP.eof())
466 {
467     j0=0;
468     fgainP >> trash >> trash >> gainP[iP] >> trash >> trash >> FWHM[iP] >> trash >> trash >> trash >> trash
... >> checkhour >> checkday >> checkmonth >> checkyear;
469     checkyear=2000+checkyear;
470     while(checkhour!=dt_hour[j0] || checkday!=dt_day[j0] || checkmonth!=dt_month[j0] ||
... checkyear!=dt_year[j0] )
471     {
472         j0++;
473         if(j0>9999)
474         {
475             pass=0;
476             break;
477         }
478         else pass=1;
479     }
480     if(checkhour==dt_hour[j0] && checkday!=dt_day[j0] && checkmonth==dt_month[j0] &&
... checkyear==dt_year[j0]) pass=1;
481
482     yearP = checkyear;
483     monthP = checkmonth;
484     dayP = checkday;
485     hourP = 0; minP = 0; secP = 0;
486     TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
487     UInt_t ttt;

```

```

488 ttt = dateP->Convert();
489 pTimeP = (int)ttt;
490 periodday = (float)pTimeP;
491
492 if (pass==1)
493 {
494     TPall[nused] = (Tin[j0]+273.1)/Patm[j0];
495     gainTPall[nused] = gainP[iP];
496     TPPeakall->Fill(TPall[nused], gainTPall[nused]);
497     if(periodday>periodstart && periodday<periodend)
498     {
499         selectedperiod=1;
500     }
501     else selectedperiod=0;
502     // peakpos vs T/P plot
503     if(periodday>Time1[0] && periodday<Time1[1])
504     //if(periodday>Time1[6])
505     {
506         TP1[tp1] = (Tin[j0]+273.1)/Patm[j0];
507         gainTP1[tp1] = gainP[iP];
508         tp1++;
509         TP[tp] = (Tin[j0]+273.1)/Patm[j0];
510         gainTP[tp] = gainP[iP];
511         timeTP[tp] = periodday;
512         TPPeak->Fill(TP[tp], gainTP[tp]);
513         int gn=0;
514         while(gn<gainTP[tp])
515         {
516             TPtimeG->Fill(timeTP[tp],TP[tp]);
517             gn++;
518         }
519         tp++;
520     }
521
522     if(periodday>Time1[2] && periodday<Time1[3])
523     {
524         TP2[tp2] = (Tin[j0]+273.1)/Patm[j0];
525         gainTP2[tp2] = gainP[iP];
526         tp2++;
527         TP[tp] = (Tin[j0]+273.1)/Patm[j0];
528         gainTP[tp] = gainP[iP];
529         timeTP[tp] = periodday;
530         TPPeak->Fill(TP[tp], gainTP[tp]);
531         int gn=0;
532         while(gn<gainTP[tp])
533         {
534             TPtimeG->Fill(timeTP[tp],TP[tp]);
535             gn++;
536         }
537         tp++;
538     }
539
540     if(periodday>Time1[6])
541     {
542         TP3[tp3] = (Tin[j0]+273.1)/Patm[j0];
543         gainTP3[tp3] = gainP[iP];
544         tp3++;
545         TP[tp] = (Tin[j0]+273.1)/Patm[j0];
546         gainTP[tp] = gainP[iP];
547         timeTP[tp] = periodday;
548         TPPeak->Fill(TP[tp], gainTP[tp]);
549         int gn=0;
550         while(gn<gainTP[tp])
551         {
552             TPtimeG->Fill(timeTP[tp],TP[tp]);
553             gn++;
554         }
555         tp++;
556     }
557
558     //
559     Tcorr = pow(((Tin[j0]+273.1)/(T0+273.1)),-0.1*best_alpha);
560     Pcorr = pow((P0/Patm[j0]),-0.1*best_beta);
561     gain_corrP = Tcorr * Pcorr * gainP[iP];
562
563     //gaincorr << nused << "\t" << checkyear << "\t" << checkmonth << "\t" << checkday << "\t" <<
... checkhour << "\t" << gainP[iP] << "\t" << gain_corrP << "\t" << FWHM[iP] << "\t" << selectedperiod <<endl;
564     nused++;
565 }
566 iP++;
567 }
568 cout << tp << "      " << tp1 << "      " << tp2 << "      " << tp3 << endl;
569
570
571 //*****END MAIN

```

```

571... LOOP*****
572
573 //Canvas1 = new TCanvas("Gain vs T/P","Gain vs T/P",10,10,1300,800);
574 TPplot1 = new TGraph(tp1-1,TP1, gainTP1);
575 TPplot1->SetMarkerSize(0.6);
576 TPplot1->SetMarkerStyle(21);
577 TPplot1->SetMarkerColor(2);
578 TPplot1->SetTitle("7/1/14 - 27/2/14");
579
580 TPplot2 = new TGraph(tp2-1,TP2, gainTP2);
581 TPplot2->SetMarkerSize(0.6);
582 TPplot2->SetMarkerStyle(21);
583 TPplot2->SetMarkerColor(3);
584 TPplot2->SetTitle("28/5/14 - 14/7/14");
585
586 TPplot3 = new TGraph(tp3-1,TP3, gainTP3);
587 TPplot3->SetMarkerSize(0.6);
588 TPplot3->SetMarkerStyle(21);
589 TPplot3->SetMarkerColor(4);
590 TPplot3->SetTitle("28/5/14 - 14/7/14");
591
592 Canvas1 = new TCanvas("Gain vs T/P","Gain vs T/P",10,10,1600,800);
593 TPplot = new TMultiGraph("Gain vs T/P", "Gain vs T/P");
594 TPplot->Add(TPplot1);
595 TPplot->Add(TPplot2);
596 TPplot->Add(TPplot3);
597 TPplot->Draw("AP");
598 TPplot->SetTitle("Gain vs T/P ");
599 TPplot->GetXaxis()->SetTitle("T/P [K/mbar]");
600 TPplot->GetYaxis()->SetTitle("Peak [adc]");
601 TPplot->Draw("AP");
602 Canvas1->BuildLegend(0.75);
603 /*
604 Canvas2 = new TCanvas("T/P vs time","T/P vs time",10,10,1300,800);
605 TPTime = new TGraph(tp-1, timeTP, TP);
606 //Peak position
607 TPTime->SetMarkerSize(0.6);
608 TPTime->SetMarkerStyle(21);
609 TPTime->SetMarkerColor(1);
610 TPTime->SetTitle("T/P vs time");
611 TPTime->GetXaxis()->SetTitle("Time");
612 TPTime->GetYaxis()->SetTitle("T/P [K/mbar]");
613 //TPTime->GetXaxis()->SetNdivisions(605);
614 //TPTime->GetXaxis()->SetTimeDisplay(1);
615 //TPTime->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
616 TPTime->SetMinimum(0.28);
617 TPTime->SetMaximum(0.32);
618 TPTime->Draw("AP");
619 */
620 Canvas3 = new TCanvas("T/P vs time & gain","T/P vs time & gain",10,10,1300,800);
621 TPTimeG = new TGraph2D(tp-1, timeTP, TP, gainTP);
622 //Peak position
623 TPTimeG->SetMarkerSize(0.6);
624 TPTimeG->SetMarkerStyle(21);
625 TPTimeG->SetMarkerColor(1);
626 TPTimeG->SetTitle("T/P vs time");
627 TPTimeG->GetXaxis()->SetTitle("Time");
628 TPTimeG->GetYaxis()->SetTitle("T/P [K/mbar]");
629 TPTimeG->GetZaxis()->SetTitle("Peak [adc]");
630 TPTimeG->GetXaxis()->SetNdivisions(605);
631 TPTimeG->GetYaxis()->SetNdivisions(605);
632 TPTimeG->GetXaxis()->SetTimeDisplay(1);
633 TPTimeG->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
634 //TPTimeG->SetMinimum(0.28);
635 //TPTimeG->SetMaximum(0.32);
636 TPTimeG->Draw("P");
637
638 Canvas4 = new TCanvas("TP vs peak all","TP vs peak all",10,10,1300,800);
639 gStyle->SetOptStat(00000000);
640 //gStyle->SetPalette()
641 TPPeakall -> Draw("BOX");
642
643 Canvas5 = new TCanvas("TP vs peak","TP vs peak",10,10,1300,800);
644 gStyle->SetOptStat(00000000);
645 //gStyle->SetPalette()
646 TPPeak -> Draw("BOX");
647
648 Canvas6 = new TCanvas("TP vs time C","TP vs time C",10,10,1300,800);
649 gStyle->SetOptStat(00000000);
650 //gStyle->SetPalette()
651 TPtimeG -> Draw("COLZ");
652
653 }
654
655

```

```
656 // exit(0);
657
658
659
660
661
662
```

### 9.3. FineoutFilter.C

```

1 //trend of enviromental parameter, current, corrected current, peak position
2 // plot for only ONE single wire -> choose SW from DataPeak line! (or use other script: PeakTrend_v?.C)
3 // output of enviromental parameter in AmbParCorr.dat
4
5 #include <string>
6 #include <stdlib>
7 #include <fstream>
8 #include <iostream>
9
10
11 void FineoutFilter()
12 {
13
14     //open file
15
16     fstream filtered("C:/root/macros/SingleWire/OutputandPlots/FilterdFineout.dat",ios::out);
17     fstream DataCurrent("C:/root/macros/SingleWire/fineout45.out",ios::in); //merged file of current and
... environmental parameters
18     int i=0;
19     int j=0;
20     int r=0;
21     int f=0;
22     int pass=0;
23     int time[100000];
24     float a6[100000], a7[100000], a9[100000], a12[100000], a15[100000], a17[100000], a18[100000];
25     int time1[100000];
26     float a61[100000], a71[100000], a91[100000], a121[100000], a151[100000], a171[100000], a181[100000];
27     while(!DataCurrent.eof())
28     {
29         DataCurrent >> time[i] >> a6[i] >> a7[i] >> a9[i] >> a12[i] >> a15[i] >> a17[i] >> a18[i];
30         //filtered << time[i] << "\t" << a6[i] << "\t" << a7[i] << "\t" << a12[i] << "\t" << a15[i] << "\t" << a17[i] << "\t" << a18[i] << endl;
... a12[i] << "\t" << a15[i] << "\t" << a17[i] << "\t" << a18[i] << endl;
31         i++;
32     }
33     j=i;
34     while(j>=0)
35     {
36         //cout << j << endl;
37         for(r=1; r<=j; r++)
38         {
39             //cout << r << endl;
40             if(time[j] != time[j-r]) pass=1;
41             else
42             {
43                 pass=0;
44                 break;
45             }
46         }
47         if(pass == 1)
48         {
49             time1[f] = time[j];
50             a61[f] = a6[j];
51             a71[f] = a7[j];
52             a91[f] = a9[j];
53             a121[f] = a12[j];
54             a151[f] = a15[j];
55             a171[f] = a17[j];
56             a181[f] = a18[j];
57             f++;
58         }
59         j--;
60     }
61     int k = f;
62     while(k>=0)
63     {
64         filtered << time1[k] << "\t" << a61[k] << "\t" << a71[k] << "\t" << a91[k] << "\t" << a121[k] << "\t"
... << a151[k] << "\t" << a171[k] << "\t" << a181[k] << endl;
65         k--;
66     }
67
68
69 }
70
71

```

#### 9.4. IntegratedCurrent.C

```

1 //trend of enviromental parameter, current, corrected current, peak position
2 // plot for only ONE single wire -> choose SW from DataPeak line! (or use other script: PeakTrend_v?.C)
3 // output of enviromental parameter in AmbParCorr.dat
4
5 #include <string>
6 #include <stdlib>
7 #include <fstream>
8 #include <iostream>
9
10
11 void IntegratedCurrent(){
12
13     //open file
14     //fstream check("C:/root/macros/SingleWire/OutputandPlots/check.out",ios::out);
15     fstream IntCurrentAll("C:/root/macros/SingleWire/OutputandPlots/IntCurrentAll.dat",ios::out);
16     fstream IntCurrentP("C:/root/macros/SingleWire/OutputandPlots/IntCurrentSW2P4_1.dat",ios::out);
17     fstream DataCurrent("C:/root/macros/SingleWire/OutputandPlots/FilterdFineout.dat",ios::in); //merged file
18     //fstream DataPeak("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW1.txt",ios::in);
19     //remember to choose SW1 or SW2!!!
20
21     //file variables
22     float a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19;
23     float alnew;
24     float mean1, mean2, maximumX, FirstValue, integral, integralnoise;
25     float convYearPeak, convMonthPeak, convDayPeak, convHourPeak, convMinutePeak, convSecondPeak, a6fPeak,
26     a12fPeak, a15fPeak; //Bea
27     float
28     alf[100000],a2f[100000],a3f[100000],a4f[100000],a5f[100000],a6f[100000],a7f[100000],a8f[100000],a9f[100000],
29     a10f[100000];
30     float
31     a11f[100000],a12f[100000],a13f[100000],a14f[100000],a15f[100000],a15f_test[100000],a16f[100000],a17f[100000],
32     a18f[100000],a19f[100000];
33     float IntCurr[100000], IntCurrA[100000];
34     float alnewf[100000];
35     float P0 = 970.; //mbar
36     float T0 = 293.; //K
37
38     //time variables
39     char cday[30],cmonth[30],cyear[30],chour[30],cmin[30],csec[30];
40     int day,month,year,hour,min,sec;
41     TDateTime t; //boh
42     UInt t ctime; //boh
43     int convYear[100000], convMonth[100000], convDay[100000]; //boh
44     int convHour[100000], convMinute[100000], convSecond[100000]; //boh
45     int pTime;
46     float Time, Timef[100000], Timeg[100000];
47
48     //plot variables
49     TCanvas *c[100];
50     TGraph *gr[100];
51     TLegend *Legend;
52
53     char ctitle[30];
54     char titlegif[60];
55     char title[60],title2[60],titleday[60];
56
57     //?
58     int iplt;
59     int i=0;
60     int oldHour = 0; //boh
61
62     while(!DataCurrent.eof())
63     {
64         DataCurrent >> alnew >> a6 >> a7 >> a9 >> a12 >> a15 >> a17 >> a18;
65
66         Timef[i] = alnew;
67         t.Set(alnew); //boh
68         convYear[i] = t.GetYear(); convMonth[i] = t.GetMonth(); convDay[i] = t.GetDay(); //boh
69         convHour[i] = t.GetHour(); convMinute[i] = t.GetMinute(); convSecond[i] = t.GetSecond(); //boh
70         /*         if (i<10) { */
71         /*             cout << alnew << "\t" << convYear[i] << "\t" << convMonth[i] << "\t" << convDay[i] << endl;
72         */
73     }
74     /*
75     cout << convHour[i] << "\t" << convMinute[i] << "\t" << convSecond[i] << endl; */
76     //T1
77     a6f[i] = a6;
78     //T2
79     a7f[i] = a7;
80     //I (nA)
81     a9f[i] = -1*a9 * 1000. - 0.2; //(0.2 nA dark current)
82     //dewp
83     a12f[i] = 7.5*(a12*1000)-90.;
84     //Pabs
85     a15f[i] = ((0.075*(1000*a17))-0.3)*1000.; //giusto che sia a17: invertito in file creato da labview

```



```

78 //Prel
79 a17f[i] = 3.125*(a15*1000)-37.5;
80 //O2
81 a18f[i] = 6.25*(a18*1000)-25;
82 i++;
83 }
84 int k=0;
85 int k1=0;
86 float periodstart, periodend, periodday;
87 int yearP, monthP, dayP, hourP, minP, secP;
88
89 //*****Period
... selector*****
90
91 yearP = 2013;
92
93 monthP = 10;
94
95 dayP = 1;
96
97 hourP = 0; minP = 0; secP = 0;
98
99 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
100
101 UInt_t ttt;
102
103 ttt = dateP->Convert();
104
105 pTimeP = (int)ttt;
106
107 periodstart = (float)pTimeP;
108
109
110
111 yearP = 2013;
112
113 monthP = 11;
114
115 dayP = 1;
116
117 hourP = 0; minP = 0; secP = 0;
118
119 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
120
121 UInt_t ttt;
122
123 ttt = dateP->Convert();
124
125 pTimeP = (int)ttt;
126
127 periodend = (float)pTimeP;
128
... //*****
... *****
129 while(k<i)
130 {
131
132 yearP = convYear[k];
133
134 monthP = convMonth[k];
135
136 dayP = convDay[k];
137
138 hourP = convHour[k]; minP = 0; secP = 0;
139
140 TDateTime * dateP = new TDateTime(yearP, monthP, dayP, hourP, minP, secP);
141
142 UInt_t ttt;
143
144 ttt = dateP->Convert();
145
146 pTimeP = (int)ttt;
147
148 periodday = (float)pTimeP;
149
150
151
152 if(periodday>periodstart && periodday<periodend)
153 {
154 if(k1==0) IntCurr[k1] = a9f[k];
155 else IntCurr[k1] = a9f[k] + IntCurr[k1-1];
156 if (convHour[k] != oldHour && convHour[k]%6==0)
157 {
158 IntCurrentP << convYear[k] << "\t" << convMonth[k] << "\t" << convDay[k] << "\t" << convHour[k]
... << "\t"<< IntCurr[k1] << endl;

```

```
159         oldHour = convHour[k];
160     }
161     k1++;
162 }
163
164     if(k==0) IntCurrA[k] = a9f[k];
165     else IntCurrA[k] = a9f[k] + IntCurrA[k-1];
166     IntCurrentAll << convYear[k] << "\t" << convMonth[k] << "\t" << convDay[k] << "\t" << convHour[k] <<
... "\t" << IntCurrA[k] << "\t" << a9f[k] << endl;
167
168     k++;
169 }
170 }
171
```

## 9.5. PeakTend\_v3.C

```

1 //trend of the peak position
2
3 #include <string>
4 #include <stdlib>
5 #include <fstream>
6 #include <iostream>
7 #include <TGaxis.h>
8 #include "TCanvas.h"
9
10 void PeakTrend_v3(){
11
12 //-----
13
14 gROOT->SetStyle("Plain");
15 // background is no longer mouse-dropping white
16 gStyle->SetCanvasColor(kWhite);
17 // blue to red false color palette. Use 9 for b/w
18 gStyle->SetPalette(1,0);
19 // turn off canvas borders
20 gStyle->SetCanvasBorderMode(0);
21 gStyle->SetPadBorderMode(0);
22 // What precision to put numbers if plotted with "TEXT"
23 gStyle->SetPaintTextFormat("5.2f");
24
25 // For publishing:
26 gStyle->SetLineWidth(1.5);
27 gStyle->SetTextSize(1.1);
28 gStyle->SetLabelSize(0.03,"xy");
29 gStyle->SetTitleSize(0.04,"xy");
30 gStyle->SetTitleOffset(1.1,"x");
31 gStyle->SetTitleOffset(0.9,"y");
32 gStyle->SetPadTopMargin(0.1);
33 gStyle->SetPadRightMargin(0.05);
34 gStyle->SetPadBottomMargin(0.1);
35 gStyle->SetPadLeftMargin(0.1);
36
37 //-----
38
39 //Open File
40 fstream DataPeakSW1("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW1clean.txt",ios::in);
41 fstream DataPeakSW2("C:/root/macros/SingleWire/OutputandPlots/peakpositionfinal_SW2clean.txt",ios::in);
42 fstream gaincorrSW1("C:/root/macros/SingleWire/OutputandPlots/gaincorrSW1.dat",ios::in);
43 fstream gaincorrSW2("C:/root/macros/SingleWire/OutputandPlots/gaincorrSW2.dat",ios::in);
44
45 //fstream DataPeakSW2("/Applications/root/macros/SW/OutputAndPlots/peakpositionRMS_SW2.txt",ios::in);
46 //fstream AmbParCorr2("/Applications/root/macros/SW/OutputAndPlots/AmbParCorr2.dat",ios::in); //boh
47
48 //Variables
49 float mean1, mean2, maximumX, maximumXcorr, FirstValue, FirstValue1, FirstValue2, FirstValuec, integral,
...
integralnoise, RMS, maximumY, fit1RMS, fit2RMS, FWHM;
50 float convYearPeak, convMonthPeak, convDayPeak, convHourPeak, convMinutePeak, convSecondPeak, a6fPeak,
...
a12fPeak, a15fPeak;
51 float CorrFact[100000];
52 float PeakADC1[100000], PeakADC1corr[100000], PeakADCtmp, maximumXSW1[100000], mean1SW1[100000],
mean2SW1[100000], FWHMSW1[100000], fit1RMSSW1[100000], fit2RMSSW1[100000], resolutionSW1[100000],
...
integralSW1[100000], RMSSW1[100000];
53 float PeakADC2[100000], PeakADC2corr[100000], PeakADCtmp2, maximumXSW2[100000], mean1SW2[100000],
mean2SW2[100000], FWHMSW2[100000], fit1RMSSW2[100000], fit2RMSSW2[100000], resolutionSW2[100000],
...
integralSW2[100000], RMSSW2[100000];
54 float P0 = 970.; //mbar
55 float T0 = 293.; //K
56 int day,month,year,hour,min,sec;
57 int yearC, monthC, dayC, hourC;
58 float Timeg1[100000], Timeg2[100000], TimegC[100000], TimegC2[100000];
59 float trash;
60
61 /***** SW1 *****/
62
63 int i=0;
64 int iC=0;
65 int j=0;
66
67 //Peak SW1
68 while(!gaincorrSW1.eof()) //while(!DataPeakSW1.eof() && !AmbParCorr2.eof())
69 {
70     gaincorrSW1 >> trash >> yearC >> monthC >> dayC >> hourC >> trash >> maximumXcorr >> trash >> trash;
71
72     //Time conversion
73     min=0;
74     sec=0;
75     TDate * date = new TDate(yearC, monthC, dayC, hourC, min, sec);
76     UInt_t ttt;
77     ttt = date->Convert();
78     pTime = (int)ttt;
79     TimegC[iC] = (float)pTime;

```

```

80
81
82 //Normalization for peak position
83 if (iC==0)
84 {
85     FirstValuec = maximumXcorr; //define the first value for the normalization
86     PeakADC1corr[iC]=maximumXcorr/FirstValuec;
87 }
88 if (iC>0)
89 {
90     PeakADC1corr[iC] = maximumXcorr/FirstValuec;
91 }
92 iC++;
93 }
94 int nptPC = iC-1;
95
96 iC=0;
97 //SW2 corrected peak position
98
99 while(!gaincorrSW2.eof()) //while(!DataPeakSW1.eof() && !AmbParCorr2.eof())
100 {
101     gaincorrSW2 >> trash >> yearC >> monthC >> dayC >> hourC >> trash >> maximumXcorr >> trash >> trash;
102
103     //Time conversion
104     min=0;
105     sec=0;
106     TDate * date = new TDate(yearC, monthC, dayC, hourC, min, sec);
107     UInt_t ttt;
108     ttt = date->Convert();
109     pTime = (int)ttt;
110     TimegC2[iC] = (float)pTime;
111
112
113     //Normalization for peak position
114     if (iC==0)
115     {
116         FirstValuec = maximumXcorr; //define the first value for the normalization
117         PeakADC2corr[iC]=maximumXcorr/FirstValuec;
118     }
119     if (iC>0)
120     {
121         PeakADC2corr[iC] = maximumXcorr/FirstValuec;
122     }
123     iC++;
124 }
125 int nptPC2 = iC-1;
126
127 while(!DataPeakSW1.eof()) //while(!DataPeakSW1.eof() && !AmbParCorr2.eof())
128 { //Bea
129
130     DataPeakSW1 >> mean1 >> mean2 >> maximumX >> maximumY >> RMS >> FWHM >> fit1RMS >> fit2RMS >> integral >>
... integralnoise >> hour >> day >> month >> year;
131     year= 2000 + year;
132
133     //Time conversion
134     min=0;
135     sec=0;
136     TDate * date = new TDate(year, month, day, hour, min, sec);
137     UInt_t ttt;
138     ttt = date->Convert();
139     pTime = (int)ttt;
140     Timegl[i] = (float)pTime;
141
142     //Variables for plots
143
144
145     //mean1SW1[i] = mean1;
146     //mean2SW1[i] = mean2;
147     FWHMSW1[i] = FWHM;
148     if (fit1RMS>=0) fit1RMSSW1[i] = fit1RMS;
149     if (fit1RMS<0) fit1RMSSW1[i] = -1*fit1RMS;
150     if (fit2RMS>=0) fit2RMSSW1[i] = fit2RMS;
151     if (fit2RMS<0) fit2RMSSW1[i] = -1*fit2RMS;
152     resolutionSW1[i] =FWHM/maximumX;
153     integralsW1[i] = integral;
154     RMSSW1[i] = RMS;
155
156     //Normalization for peak position, mean1, mean 2
157     if (i==0)
158     {
159         FirstValue = maximumX; //define the first value for the normalization
160         PeakADC1[i]=maximumX/FirstValue;
161         FirstValue1 = mean1; //define the first value for the normalization
162         mean1SW1[i]=mean1/FirstValue1;
163         FirstValue2 = mean2; //define the first value for the normalization

```

```

164     mean2SW1[i]=mean2/FirstValue2;
165 }
166 if (i>0)
167 {
168     PeakADC1[i] = maximumX/FirstValue;
169     mean1SW1[i] = mean1/FirstValue1;
170     mean2SW1[i] = mean2/FirstValue2;
171 }
172 //PeakADC1[i] = maximumX;
173 i++;
174 }
175
176 int nptP1 = i-1;
177
178 //Graphs SW1:
179 //peak position
180
181 Canvas1 = new TCanvas("SW1 Peak Position Trend","SW1 peak trend",10,10,1300,800);
182
183 GraphSW1 = new TGraph(nptP1, Timeg1, PeakADC1);
184 //Peak position
185 GraphSW1->SetMarkerSize(0.6);
186 GraphSW1->SetMarkerStyle(21);
187 GraphSW1->SetMarkerColor(1);
188 GraphSW1->SetTitle("SW1 peak trend");
189 GraphSW1->GetXaxis()->SetTitle("Time");
190 GraphSW1->GetYaxis()->SetTitle("Peak [adc]");
191 GraphSW1->GetXaxis()->SetNdivisions(605);
192 GraphSW1->GetXaxis()->SetTimeDisplay(1);
193 GraphSW1->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
194 GraphSW1->Draw("AP");
195
196
197 //*****Time division lines for SW1*****
198 float Time1[7];
199 year = 2013;
200 month = 10;
201 day = 1;
202 hour = 0; min = 0; sec = 0;
203 TDate * date = new TDate(year, month, day, hour, min, sec);
204 UInt_t ttt;
205 ttt = date->Convert();
206 pTime = (int)ttt;
207 Time1[0] = (float)pTime;
208
209 year = 2013;
210 month = 11;
211 day = 1;
212 hour = 0; min = 0; sec = 0;
213 TDate * date = new TDate(year, month, day, hour, min, sec);
214 UInt_t ttt;
215 ttt = date->Convert();
216 pTime = (int)ttt;
217 Time1[1] = (float)pTime;
218
219 year = 2013;
220 month = 11;
221 day = 21;
222 hour = 0; min = 0; sec = 0;
223 TDate * date = new TDate(year, month, day, hour, min, sec);
224 UInt_t ttt;
225 ttt = date->Convert();
226 pTime = (int)ttt;
227 Time1[2] = (float)pTime;
228
229 year = 2014;
230 month = 1;
231 day = 7;
232 hour = 0; min = 0; sec = 0;
233 TDate * date = new TDate(year, month, day, hour, min, sec);
234 UInt_t ttt;
235 ttt = date->Convert();
236 pTime = (int)ttt;
237 Time1[3] = (float)pTime;
238
239 year = 2014;
240 month = 2;
241 day = 27;
242 hour = 0; min = 0; sec = 0;
243 TDate * date = new TDate(year, month, day, hour, min, sec);
244 UInt_t ttt;
245 ttt = date->Convert();
246 pTime = (int)ttt;
247 Time1[4] = (float)pTime;
248

```

```

249 year = 2014;
250 month = 4;
251 day = 28;
252 hour = 0; min = 0; sec = 0;
253 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
254 UInt_t ttt;
255 ttt = date->Convert();
256 pTime = (int)ttt;
257 Time1[5] = (float)pTime;
258
259 year = 2014;
260 month = 5;
261 day = 28;
262 hour = 0; min = 0; sec = 0;
263 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);
264 UInt_t ttt;
265 ttt = date->Convert();
266 pTime = (int)ttt;
267 Time1[6] = (float)pTime;
268
269 //*****Time division lines for SW1 end*****
270 //Canvas1 lines
271 Canvas1->Update();
272 for(int lines=0; lines<7; lines++)
273 {
274     TLine *templine = new TLine(Time1[lines],Canvas1->GetYmin(),Time1[lines],Canvas1->GetYmax());
275     templine->SetLineColor(32);
276     templine->SetLineWidth(2);
277     templine->SetLineStyle(5);
278     templine->Draw();
279 }
280
281 Canvas1corr = new TCanvas("Corrected SW1 Peak Position Trend","Corrected SW1 peak trend",10,10,1300,800);
282 GraphSW1corr = new TGraph(nptPC, TimegC, PeakADC1corr);
283 //Peak position
284 GraphSW1corr->SetMarkerSize(0.6);
285 GraphSW1corr->SetMarkerStyle(21);
286 GraphSW1corr->SetMarkerColor(2);
287 GraphSW1corr->SetTitle("Corrected SW1 peak trend ");
288 GraphSW1corr->GetXaxis()->SetTitle("Time");
289 GraphSW1corr->GetYaxis()->SetTitle("Peak [adc]");
290 GraphSW1corr->GetXaxis()->SetNdivisions(605);
291 GraphSW1corr->GetXaxis()->SetTimeDisplay(1);
292 GraphSW1corr->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
293 GraphSW1corr->Draw("AP");
294
295 //Canvascorr1 lines
296 Canvas1corr->Update();
297 for(int lines=0; lines<7; lines++)
298 {
299     TLine *templine = new TLine(Time1[lines],Canvas1corr->GetYmin(),Time1[lines],Canvas1corr->GetYmax());
300     templine->SetLineColor(32);
301     templine->SetLineWidth(2);
302     templine->SetLineStyle(5);
303     templine->Draw();
304 }
305
306 CanvasSW1both= new TCanvas("Corrected SW1 Peak Position Trend vs Uncorrected","Corrected SW1 peak trend vs
... Uncorrected",10,10,1300,800);
307 Corr1Graph = new TMultiGraph("SW1 peak position correction", "SW1 peak position correction");
308 Corr1Graph->Add(GraphSW1);
309 Corr1Graph->Add(GraphSW1corr);
310 Corr1Graph->Draw("AP");
311 Corr1Graph->GetXaxis()->SetTitle("Time");
312 Corr1Graph->GetYaxis()->SetTitle("Peak [adc]");
313 Corr1Graph->GetXaxis()->SetNdivisions(605);
314 Corr1Graph->GetXaxis()->SetTimeDisplay(1);
315 Corr1Graph->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
316 //MeansGraph->Draw("AP");
317
318 CanvasSW1both->Update();
319 for(int lines=0; lines<7; lines++)
320 {
321     TLine *templine = new TLine(Time1[lines], CanvasSW1both->GetYmin(),Time1[lines],
... CanvasSW1both->GetYmax());
322     templine->SetLineColor(32);
323     templine->SetLineWidth(2);
324     templine->SetLineStyle(5);
325     templine->Draw();
326 }
327
328 Canvas2 = new TCanvas("SW1 Mean1 & Mean2 Trend","SW1 Mean1 & Mean2 Trend",10,10,1300,800);
329 Canvas2->Divide(1,2);
330
331 GraphMean1SW1 = new TGraph(nptP1, Timeg1, mean1SW1);

```

```

332 GraphMean2SW1 = new TGraph(nptP1, Timeg1, mean2SW1);
333
334 //mean1
335 Canvas2->Update();
336 GraphMean1SW1->SetMarkerSize(0.75);
337 GraphMean1SW1->SetMarkerStyle(21);
338 GraphMean1SW1->SetMarkerColor(2);
339 GraphMean1SW1->SetTitle("SW1 Mean1 Trend");
340 GraphMean1SW1->GetXaxis()->SetTitle("Time");
341 GraphMean1SW1->GetYaxis()->SetTitle("mean [adc]");
342 GraphMean1SW1->GetXaxis()->SetNdivisions(605);
343 GraphMean1SW1->GetXaxis()->SetTimeDisplay(1);
344 GraphMean1SW1->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
345
346 //mean2
347 GraphMean2SW1->SetMarkerSize(0.6);
348 GraphMean2SW1->SetMarkerStyle(21);
349 GraphMean2SW1->SetMarkerColor(3);
350 GraphMean2SW1->SetTitle("SW1 Mean2 Trend");
351 GraphMean2SW1->GetXaxis()->SetTitle("Time");
352 GraphMean2SW1->GetYaxis()->SetTitle("mean [adc]");
353 GraphMean2SW1->GetXaxis()->SetNdivisions(605);
354 GraphMean2SW1->GetXaxis()->SetTimeDisplay(1);
355 GraphMean2SW1->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
356
357 Canvas2->cd(1);
358 GraphMean1SW1->Draw("AP");
359
360 Canvas2->cd(2);
361 GraphMean2SW1->Draw("AP");
362
363 //Canvas2 lines
364 Canvas2->Update();
365 for(int lines=0; lines<7; lines++)
366 {
367     TLine *templine = new
... TLine(Time1[lines],Canvas2->cd(1)->GetYmin(),Time1[lines],Canvas2->cd(1)->GetYmax());
368     templine->SetLineColor(32);
369     templine->SetLineWidth(2);
370     templine->SetLineStyle(5);
371     templine->Draw();
372 }
373 for(int lines=0; lines<7; lines++)
374 {
375     TLine *templine = new
... TLine(Time1[lines],Canvas2->cd(2)->GetYmin(),Time1[lines],Canvas2->cd(2)->GetYmax());
376     templine->SetLineColor(32);
377     templine->SetLineWidth(2);
378     templine->SetLineStyle(5);
379     templine->Draw();
380 }
381
382 /*
383 Canvas7 = new TCanvas("SW1 Means","SW1 Means",10,10,1300,800);
384 MeansGraph = new TMultiGraph("SW1 Means 1 2", "SW1 Means 1 2");
385 MeansGraph->Add(GraphMean1SW1);
386 MeansGraph->Add(GraphMean2SW1);
387 MeansGraph->Draw("AP");
388 MeansGraph->GetXaxis()->SetTitle("Time");
389 MeansGraph->GetYaxis()->SetTitle("Mean [adc]");
390 MeansGraph->GetXaxis()->SetNdivisions(605);
391 MeansGraph->GetXaxis()->SetTimeDisplay(1);
392 MeansGraph->GetXaxis()->SetTimeFormat("%d\%m\%Y %F1970-01-01 00:00:00");
393 //MeansGraph->Draw("AP");
394
395 //Canvas7 lines
396 Canvas7->Update();
397 for(int lines=0; lines<7; lines++)
398 {
399     TLine *templine = new TLine(Time1[lines],Canvas7->GetYmin(),Time1[lines],Canvas7->GetYmax());
400     templine->SetLineColor(32);
401     templine->SetLineWidth(2);
402     templine->SetLineStyle(5);
403     templine->Draw();
404 }
405
406
407 //FWHM
408 Canvas3 = new TCanvas("SW1 FWHM Trend","SW1 FWHM trend",10,10,1300,800);
409
410 GraphFWHMSW1 = new TGraph(nptP1, Timeg1, FWHMSW1);
411 //Peak position
412 GraphFWHMSW1->SetMarkerSize(0.6);
413 GraphFWHMSW1->SetMarkerStyle(21);
414 GraphFWHMSW1->SetMarkerColor(1);

```



```

415 GraphFWHMSW1->SetTitle("SW1 FWHM trend");
416 GraphFWHMSW1->GetXaxis()->SetTitle("Time");
417 GraphFWHMSW1->GetYaxis()->SetTitle("FWHM [adc]");
418 GraphFWHMSW1->GetXaxis()->SetNdivisions(605);
419 GraphFWHMSW1->GetXaxis()->SetTimeDisplay(1);
420 GraphFWHMSW1->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
421 GraphFWHMSW1->SetMinimum(0.1);
422 GraphFWHMSW1->Draw("AP");
423
424 //Canvas3 lines
425 Canvas3->Update();
426 for(int lines=0; lines<7; lines++)
427 {
428     TLine *templine = new TLine(Time1[lines],Canvas3->GetYmin(),Time1[lines],Canvas3->GetYmax());
429     templine->SetLineColor(32);
430     templine->SetLineWidth(2);
431     templine->SetLineStyle(5);
432     templine->Draw();
433 }
434 /*
435 //Resolution
436 Canvas4 = new TCanvas("SW1 Resolution Trend","SW1 Resolution trend",10,10,1300,800);
437
438 GraphResolutionSW1 = new TGraph(nptP1, Timeg1, resolutionSW1);
439 //Peak position
440 GraphResolutionSW1->SetMarkerSize(0.6);
441 GraphResolutionSW1->SetMarkerStyle(21);
442 GraphResolutionSW1->SetMarkerColor(1);
443 GraphResolutionSW1->SetTitle("SW1 Resolution trend");
444 GraphResolutionSW1->GetXaxis()->SetTitle("Time");
445 GraphResolutionSW1->GetYaxis()->SetTitle("SW1 Resolution");
446 GraphResolutionSW1->GetXaxis()->SetNdivisions(605);
447 GraphResolutionSW1->GetXaxis()->SetTimeDisplay(1);
448 GraphResolutionSW1->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
449 GraphResolutionSW1->SetMinimum(0.1);
450 GraphResolutionSW1->Draw("AP");
451
452 //Canvas4 lines
453 Canvas4->Update();
454 for(int lines=0; lines<7; lines++)
455 {
456     TLine *templine = new TLine(Time1[lines],Canvas4->GetYmin(),Time1[lines],Canvas4->GetYmax());
457     templine->SetLineColor(32);
458     templine->SetLineWidth(2);
459     templine->SetLineStyle(5);
460     templine->Draw();
461 }
462 /*
463 //RMS
464 Canvas5 = new TCanvas("SW1 fit1RMS & fit2RMS Trend","SW1 fit1RMS & fit2RMS Trend",10,10,1300,800);
465 Canvas5->Divide(1,2);
466
467 Graphfit1RMSSW1 = new TGraph(nptP1, Timeg1, fit1RMSSW1);
468 Graphfit2RMSSW1 = new TGraph(nptP1, Timeg1, fit2RMSSW1);
469
470 //fit1RMS
471 Canvas5->Update();
472 Graphfit1RMSSW1->SetMarkerSize(0.75);
473 Graphfit1RMSSW1->SetMarkerStyle(21);
474 Graphfit1RMSSW1->SetMarkerColor(2);
475 Graphfit1RMSSW1->SetTitle("SW1 fit1RMS Trend");
476 Graphfit1RMSSW1->GetXaxis()->SetTitle("Time");
477 Graphfit1RMSSW1->GetYaxis()->SetTitle("fit1RMS [adc]");
478 Graphfit1RMSSW1->GetXaxis()->SetNdivisions(605);
479 Graphfit1RMSSW1->GetXaxis()->SetTimeDisplay(1);
480 Graphfit1RMSSW1->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
481 Graphfit1RMSSW1->SetMaximum(300);
482
483 //fit2RMS
484 Graphfit2RMSSW1->SetMarkerSize(0.6);
485 Graphfit2RMSSW1->SetMarkerStyle(21);
486 Graphfit2RMSSW1->SetMarkerColor(3);
487 Graphfit2RMSSW1->SetTitle("SW1 fit2RMS Trend");
488 Graphfit2RMSSW1->GetXaxis()->SetTitle("Time");
489 Graphfit2RMSSW1->GetYaxis()->SetTitle("fit2RMS [adc]");
490 Graphfit2RMSSW1->GetXaxis()->SetNdivisions(605);
491 Graphfit2RMSSW1->GetXaxis()->SetTimeDisplay(1);
492 Graphfit2RMSSW1->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
493 Graphfit2RMSSW1->SetMaximum(300);
494
495 Canvas5->cd(1);
496 Graphfit1RMSSW1->Draw("AP");
497
498 Canvas5->cd(2);
499 Graphfit2RMSSW1->Draw("AP");

```

```

500 //Canvas5 lines
501 Canvas5->Update();
502 for(int lines=0; lines<7; lines++)
503 {
504     TLine *templine = new
505     ... TLine(Time1[lines],Canvas5->cd(1)->GetYmin(),Time1[lines],Canvas5->cd(1)->GetYmax());
506     templine->SetLineColor(32);
507     templine->SetLineWidth(2);
508     templine->SetLineStyle(5);
509     templine->Draw();
510 }
511 for(int lines=0; lines<7; lines++)
512 {
513     TLine *templine = new
514     ... TLine(Time1[lines],Canvas5->cd(2)->GetYmin(),Time1[lines],Canvas5->cd(2)->GetYmax());
515     templine->SetLineColor(32);
516     templine->SetLineWidth(2);
517     templine->SetLineStyle(5);
518     templine->Draw();
519 }
520 //Integral
521 Canvas6 = new TCanvas("SW1 Integral Trend","SW1 Integral trend",10,10,1300,800);
522
523 GraphIntSW1 = new TGraph(nptP1, Timeg1, integralSW1);
524 GraphIntSW1->SetMarkerSize(0.6);
525 GraphIntSW1->SetMarkerStyle(21);
526 GraphIntSW1->SetMarkerColor(1);
527 GraphIntSW1->SetTitle("SW1 Integral trend");
528 GraphIntSW1->GetXaxis()->SetTitle("Time");
529 GraphIntSW1->GetYaxis()->SetTitle("Integral");
530 GraphIntSW1->GetXaxis()->SetNdivisions(605);
531 GraphIntSW1->GetXaxis()->SetTimeDisplay(1);
532 GraphIntSW1->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
533 GraphIntSW1->SetMaximum(1000000);
534 GraphIntSW1->SetMinimum(100000);
535 GraphIntSW1->Draw("AP");
536
537 //Canvas6 lines
538 Canvas6->Update();
539 for(int lines=0; lines<7; lines++)
540 {
541     TLine *templine = new TLine(Time1[lines],Canvas6->GetYmin(),Time1[lines],Canvas6->GetYmax());
542     templine->SetLineColor(32);
543     templine->SetLineWidth(2);
544     templine->SetLineStyle(5);
545     templine->Draw();
546 }
547
548 */
549 /*
550 //Scale GraphSWint to the pad coordinates
551 Float_t rightmax = GraphSW1RMS->GetHistogram()->GetMaximum();
552 Float_t scale = gPad->GetYmax()/rightmax;
553 GraphSW1RMS->SetLineColor(kRed);
554 for (int i=0;i<GraphSW1RMS->GetN();i++) {
555     GraphSW1RMS->GetY()[i] *= scale; //equivalent to scale
556 }
557 GraphSW1RMS->Draw("P");
558
559 //draw an axis on the right side
560 TGaxis *axis = new TGaxis(gPad->GetXmax(),gPad->GetYmin(),
561     gPad->GetXmax(), gPad->GetYmax(),0,rightmax,510,"+L");
562 axis->SetLineColor(kRed);
563 axis->SetLabelColor(kRed);
564 axis->Draw();
565 */
566
567
568
569 /***** SW2 *****/
570
571 int k = 0;
572
573 //Peak SW2
574 while(!DataPeakSW2.eof()){
575
576     DataPeakSW2 >> mean1 >> mean2 >> maximumX >> maximumY >> RMS >> FWHM >> fit1RMS >> fit2RMS >> integral >>
577     ... integralnoise >> hour >> day >> month >> year;
578     year= 2000 + year; //spostata qui da sotto
579
580 //Time conversion
581 TDateTime * date = new TDateTime(year, month, day, hour, min, sec);

```

```

582     UInt_t ttt;
583     ttt = date->Convert();
584     pTime = (int)ttt;
585     Timeg2[k] = (float)pTime;
586
587     FWHMSW2[k] = FWHM;
588     if (fit1RMS>=0) fit1RMSSW2[k] = fit1RMS;
589     if (fit1RMS<0) fit1RMSSW2[k] = -1*fit1RMS;
590     if (fit2RMS>=0) fit2RMSSW2[k] = fit2RMS;
591     if (fit2RMS<0) fit2RMSSW2[k] = -1*fit2RMS;
592     resolutionSW2[k] =FWHM/maximumX;
593     integralSW2[k] = integral;
594     RMSSW2[k] = RMS;
595
596     //Normalization for peak position, mean1, mean 2
597     if (k==0)
598     {
599         FirstValue = maximumX; //define the first value for the normalization
600         PeakADC2[k]=maximumX/FirstValue;
601         FirstValue1 = mean1; //define the first value for the normalization
602         mean1SW2[k]=mean1/FirstValue1;
603         FirstValue2 = mean2; //define the first value for the normalization
604         mean2SW2[k]=mean2/FirstValue2;
605     }
606     if (k>0)
607     {
608         PeakADC2[k] = maximumX/FirstValue;
609         mean1SW2[k] = mean1/FirstValue1;
610         mean2SW2[k] = mean2/FirstValue2;
611     }
612     k++;
613 }
614
615 int nptP2 = k-1;
616 //for(int t=0; t<957; t++)
617     //cout <<
618
619 //Graphs for SW
620
621 //peak position
622
623 Canvas8 = new TCanvas("SW2 Peak Position Trend","SW2 peak trend",10,10,1300,800);
624
625 GraphSW2 = new TGraph(nptP2, Timeg2, PeakADC2);
626 //Peak position
627 GraphSW2->SetMarkerSize(0.6);
628 GraphSW2->SetMarkerStyle(21);
629 GraphSW2->SetMarkerColor(1);
630 GraphSW2->SetTitle("SW2 peak trend");
631 GraphSW2->GetXaxis()->SetTitle("Time");
632 GraphSW2->GetYaxis()->SetTitle("Peak [adc]");
633 GraphSW2->GetXaxis()->SetNdivisions(605);
634 GraphSW2->GetXaxis()->SetTimeDisplay(1);
635 GraphSW2->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
636 GraphSW2->Draw("AP");
637 //Canvas 8 lines
638 Canvas8->Update();
639 for(int lines=0; lines<7; lines++)
640 {
641     TLine *templine = new TLine(Time1[lines],Canvas8->GetUymin(),Time1[lines],Canvas8->GetUymax());
642     templine->SetLineColor(32);
643     templine->SetLineWidth(2);
644     templine->SetLineStyle(5);
645     templine->Draw();
646 }
647
648 Canvas8corr = new TCanvas("Corrected SW2 Peak Position Trend","Corrected SW2 peak trend",10,10,1300,800);
649 GraphSW2corr = new TGraph(nptPC2, TimegC2, PeakADC2corr);
650 //Peak position
651 GraphSW2corr->SetMarkerSize(0.6);
652 GraphSW2corr->SetMarkerStyle(21);
653 GraphSW2corr->SetMarkerColor(2);
654 GraphSW2corr->SetTitle("Corrected SW2 peak trend ");
655 GraphSW2corr->GetXaxis()->SetTitle("Time");
656 GraphSW2corr->GetYaxis()->SetTitle("Peak [adc]");
657 GraphSW2corr->GetXaxis()->SetNdivisions(605);
658 GraphSW2corr->GetXaxis()->SetTimeDisplay(1);
659 GraphSW2corr->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
660 GraphSW2corr->Draw("AP");
661
662
663
664 //Canvascorr1 lines
665 Canvas8corr->Update();
666 for(int lines=0; lines<7; lines++)

```

```

667 {
668     TLine *templine = new TLine(Time1[lines],Canvas8corr->GetYmin(),Time1[lines],Canvas8corr->GetYmax());
669     templine->SetLineColor(32);
670     templine->SetLineWidth(2);
671     templine->SetLineStyle(5);
672     templine->Draw();
673 }
674
675 CanvasSW2both= new TCanvas("Corrected SW2 Peak Position Trend vs Uncorrected","Corrected SW2 peak trend vs
... Uncorrected",10,10,1300,800);
676 corr2Graph = new TMultiGraph("SW2 peak position correction", "SW2 peak position correction");
677 corr2Graph->Add(GraphSW2);
678 corr2Graph->Add(GraphSW2corr);
679 corr2Graph->Draw("AP");
680 corr2Graph->GetXaxis()->SetTitle("Time");
681 corr2Graph->GetYaxis()->SetTitle("Peak [adc]");
682 corr2Graph->GetXaxis()->SetNdivisions(605);
683 corr2Graph->GetXaxis()->SetTimeDisplay(1);
684 corr2Graph->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
685 //MeansGraph->Draw("AP");
686
687 CanvasSW2both->Update();
688 for(int lines=0; lines<7; lines++)
689 {
690     TLine *templine = new TLine(Time1[lines], CanvasSW2both->GetYmin(),Time1[lines],
... CanvasSW2both->GetYmax());
691     templine->SetLineColor(32);
692     templine->SetLineWidth(2);
693     templine->SetLineStyle(5);
694     templine->Draw();
695 }
696
697 Canvas9 = new TCanvas("SW2 Mean1 & Mean2 Trend","SW2 Mean1 & Mean2 Trend",10,10,1300,800);
698 Canvas9->Divide(1,2);
699
700 GraphMean1SW2 = new TGraph(nptP2, Timeg2, mean1SW2);
701 GraphMean2SW2 = new TGraph(nptP2, Timeg2, mean2SW2);
702
703 //mean1
704 Canvas9->Update();
705 GraphMean1SW2->SetMarkerSize(0.75);
706 GraphMean1SW2->SetMarkerStyle(21);
707 GraphMean1SW2->SetMarkerColor(2);
708 GraphMean1SW2->SetTitle("SW2 Mean1 Trend");
709 GraphMean1SW2->GetXaxis()->SetTitle("Time");
710 GraphMean1SW2->GetYaxis()->SetTitle("mean [adc]");
711 GraphMean1SW2->GetXaxis()->SetNdivisions(605);
712 GraphMean1SW2->GetXaxis()->SetTimeDisplay(1);
713 GraphMean1SW2->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
714 GraphMean1SW2->SetMaximum(1.3);
715 GraphMean1SW2->SetMinimum(0.5);
716
717 //mean2
718 GraphMean2SW2->SetMarkerSize(0.6);
719 GraphMean2SW2->SetMarkerStyle(21);
720 GraphMean2SW2->SetMarkerColor(3);
721 GraphMean2SW2->SetTitle("SW2 Mean2 Trend");
722 GraphMean2SW2->GetXaxis()->SetTitle("Time");
723 GraphMean2SW2->GetYaxis()->SetTitle("mean [adc]");
724 GraphMean2SW2->GetXaxis()->SetNdivisions(605);
725 GraphMean2SW2->GetXaxis()->SetTimeDisplay(1);
726 GraphMean2SW2->GetXaxis()->SetTimeFormat("%d\\%m\\%Y %F1970-01-01 00:00:00");
727 GraphMean2SW2->SetMaximum(1.3);
728 GraphMean2SW2->SetMinimum(0.5);
729
730 Canvas9->cd(1);
731 GraphMean1SW2->Draw("AP");
732
733 Canvas9->cd(2);
734 GraphMean2SW2->Draw("AP");
735
736 //Canvas9 lines
737 Canvas9->Update();
738 for(int lines=0; lines<7; lines++)
739 {
740     TLine *templine = new
... TLine(Time1[lines],Canvas9->cd(1)->GetYmin(),Time1[lines],Canvas9->cd(1)->GetYmax());
741     templine->SetLineColor(32);
742     templine->SetLineWidth(2);
743     templine->SetLineStyle(5);
744     templine->Draw();
745 }
746 for(int lines=0; lines<7; lines++)
747 {
748     TLine *templine = new

```

```

748... TLine(Time1[lines],Canvas9->cd(2)->GetYmin(),Time1[lines],Canvas9->cd(2)->GetYmax());
749     templine->SetLineColor(32);
750     templine->SetLineWidth(2);
751     templine->SetLineStyle(5);
752     templine->Draw();
753 }
754
755 /*
756 Canvas10 = new TCanvas("SW2 Means","SW2 Means",10,10,1300,800);
757 MeansGraph2 = new TMultiGraph("SW2 Means 1 2", "SW2 Means 1 2");
758 MeansGraph2->Add(GraphMean1SW2);
759 MeansGraph2->Add(GraphMean2SW2);
760 MeansGraph2->Draw("AP");
761 MeansGraph2->GetXaxis()->SetTitle("Time");
762 MeansGraph2->GetYaxis()->SetTitle("Mean [adc]");
763 MeansGraph2->GetXaxis()->SetNdivisions(605);
764 MeansGraph2->GetXaxis()->SetTimeDisplay(1);
765 MeansGraph2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
766 //Canvas 10 lines
767 Canvas10->Update();
768 for(int lines=0; lines<7; lines++)
769 {
770     TLine *templine = new TLine(Time1[lines],Canvas10->GetYmin(),Time1[lines],Canvas10->GetYmax());
771     templine->SetLineColor(32);
772     templine->SetLineWidth(2);
773     templine->SetLineStyle(5);
774     templine->Draw();
775 }
776
777
778 //FWHM
779 Canvas11 = new TCanvas("SW2 FWHM Trend","SW2 FWHM trend",10,10,1300,800);
780
781 GraphFWHMSW2 = new TGraph(nptP2, Timeg2, FWHMSW2);
782 //Peak position
783 GraphFWHMSW2->SetMarkerSize(0.6);
784 GraphFWHMSW2->SetMarkerStyle(21);
785 GraphFWHMSW2->SetMarkerColor(1);
786 GraphFWHMSW2->SetTitle("SW2 FWHM trend");
787 GraphFWHMSW2->GetXaxis()->SetTitle("Time");
788 GraphFWHMSW2->GetYaxis()->SetTitle("FWHM [adc]");
789 GraphFWHMSW2->GetXaxis()->SetNdivisions(605);
790 GraphFWHMSW2->GetXaxis()->SetTimeDisplay(1);
791 GraphFWHMSW2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
792 GraphFWHMSW2->SetMinimum(100);
793 GraphFWHMSW2->Draw("AP");
794 //Canvas 11 lines
795 Canvas11->Update();
796 for(int lines=0; lines<7; lines++)
797 {
798     TLine *templine = new TLine(Time1[lines],Canvas11->GetYmin(),Time1[lines],Canvas11->GetYmax());
799     templine->SetLineColor(32);
800     templine->SetLineWidth(2);
801     templine->SetLineStyle(5);
802     templine->Draw();
803 }
804 */
805 //Resolution
806 Canvas12 = new TCanvas("SW2 Resolution Trend","SW2 Resolution trend",10,10,1300,800);
807
808 GraphResolutionSW2 = new TGraph(nptP2, Timeg2, resolutionSW2);
809 //Peak position
810 GraphResolutionSW2->SetMarkerSize(0.6);
811 GraphResolutionSW2->SetMarkerStyle(21);
812 GraphResolutionSW2->SetMarkerColor(1);
813 GraphResolutionSW2->SetTitle("SW2 Resolution trend");
814 GraphResolutionSW2->GetXaxis()->SetTitle("Time");
815 GraphResolutionSW2->GetYaxis()->SetTitle("SW2 Resolution");
816 GraphResolutionSW2->GetXaxis()->SetNdivisions(605);
817 GraphResolutionSW2->GetXaxis()->SetTimeDisplay(1);
818 GraphResolutionSW2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
819 GraphResolutionSW2->SetMinimum(0.1);
820 GraphResolutionSW2->Draw("AP");
821 //Canvas 12 lines
822 Canvas12->Update();
823 for(int lines=0; lines<7; lines++)
824 {
825     TLine *templine = new TLine(Time1[lines],Canvas12->GetYmin(),Time1[lines],Canvas12->GetYmax());
826     templine->SetLineColor(32);
827     templine->SetLineWidth(2);
828     templine->SetLineStyle(5);
829     templine->Draw();
830 }
831
832 /*

```

```

833 //RMS
834 Canvas13 = new TCanvas("SW2 fit1RMS & fit2RMS Trend","SW2 fit1RMS & fit2RMS Trend",10,10,1300,800);
835 Canvas13->Divide(1,2);
836
837 Graphfit1RMSSW2 = new TGraph(nptP2, Timeg2, fit1RMSSW2);
838 Graphfit2RMSSW2 = new TGraph(nptP2, Timeg2, fit2RMSSW2);
839
840 //fit1RMS
841 Canvas13->Update();
842 Graphfit1RMSSW2->SetMarkerSize(0.75);
843 Graphfit1RMSSW2->SetMarkerStyle(21);
844 Graphfit1RMSSW2->SetMarkerColor(2);
845 Graphfit1RMSSW2->SetTitle("SW2 fit1RMS Trend");
846 Graphfit1RMSSW2->GetXaxis()->SetTitle("Time");
847 Graphfit1RMSSW2->GetYaxis()->SetTitle("fit1RMS [adc]");
848 Graphfit1RMSSW2->GetXaxis()->SetNdivisions(605);
849 Graphfit1RMSSW2->GetXaxis()->SetTimeDisplay(1);
850 Graphfit1RMSSW2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
851 Graphfit1RMSSW2->SetMaximum(300);
852
853 //fit2RMS
854 Graphfit2RMSSW2->SetMarkerSize(0.6);
855 Graphfit2RMSSW2->SetMarkerStyle(21);
856 Graphfit2RMSSW2->SetMarkerColor(3);
857 Graphfit2RMSSW2->SetTitle("SW2 fit2RMS Trend");
858 Graphfit2RMSSW2->GetXaxis()->SetTitle("Time");
859 Graphfit2RMSSW2->GetYaxis()->SetTitle("fit2RMS [adc]");
860 Graphfit2RMSSW2->GetXaxis()->SetNdivisions(605);
861 Graphfit2RMSSW2->GetXaxis()->SetTimeDisplay(1);
862 Graphfit2RMSSW2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
863 Graphfit2RMSSW2->SetMaximum(300);
864
865 Canvas13->cd(1);
866 Graphfit1RMSSW2->Draw("AP");
867
868 Canvas13->cd(2);
869 Graphfit2RMSSW2->Draw("AP");
870
871 //Canvas13 lines
872 Canvas13->Update();
873 for(int lines=0; lines<7; lines++)
874 {
875     TLine *templine = new
... TLine(Time1[lines],Canvas13->cd(1)->GetYmin(),Time1[lines],Canvas13->cd(1)->GetYmax());
876     templine->SetLineColor(32);
877     templine->SetLineWidth(2);
878     templine->SetLineStyle(5);
879     templine->Draw();
880 }
881 for(int lines=0; lines<7; lines++)
882 {
883     TLine *templine = new
... TLine(Time1[lines],Canvas13->cd(2)->GetYmin(),Time1[lines],Canvas13->cd(2)->GetYmax());
884     templine->SetLineColor(32);
885     templine->SetLineWidth(2);
886     templine->SetLineStyle(5);
887     templine->Draw();
888 }
889
890 //Integral
891 Canvas14 = new TCanvas("SW2 Integral Trend","SW2 Integral trend",10,10,1300,800);
892
893 GraphIntSW2 = new TGraph(nptP2, Timeg2, integralSW2);
894 GraphIntSW2->SetMarkerSize(0.6);
895 GraphIntSW2->SetMarkerStyle(21);
896 GraphIntSW2->SetMarkerColor(1);
897 GraphIntSW2->SetTitle("SW2 Integral trend");
898 GraphIntSW2->GetXaxis()->SetTitle("Time");
899 GraphIntSW2->GetYaxis()->SetTitle("Integral");
900 GraphIntSW2->GetXaxis()->SetNdivisions(605);
901 GraphIntSW2->GetXaxis()->SetTimeDisplay(1);
902 GraphIntSW2->GetXaxis()->SetTimeFormat("%d/%m/%Y %F1970-01-01 00:00:00");
903 GraphIntSW2->SetMaximum(400000);
904 GraphIntSW2->Draw("AP");
905 //Canvas 14 lines
906 Canvas14->Update();
907 for(int lines=0; lines<7; lines++)
908 {
909     TLine *templine = new TLine(Time1[lines],Canvas14->GetYmin(),Time1[lines],Canvas14->GetYmax());
910     templine->SetLineColor(32);
911     templine->SetLineWidth(2);
912     templine->SetLineStyle(5);
913     templine->Draw();
914 }
915 */

```

```
916 /*
917 //Scale GraphSW2RMS to the pad coordinates
918 Float_t rightmax = GraphSW2RMS->GetHistogram()->GetMaximum();
919 Float_t scale = gPad->GetUymax()/rightmax;
920 GraphSW2RMS->SetLineColor(kRed);
921 for (int i=0;i<GraphSW2RMS->GetN();i++) {
922     GraphSW2RMS->GetY()[i] *= scale; //equivalent to scale
923 }
924 GraphSW2RMS->Draw("P");
925
926 //draw an axis on the right side
927 TGaxis *axis = new TGaxis(gPad->GetUxmax(),gPad->GetUymin(),
928     gPad->GetUxmax(), gPad->GetUymax(),0,rightmax,510,"+L");
929 axis->SetLineColor(kRed);
930 axis->SetLabelColor(kRed);
931 axis->Draw();
932 */
933 }
934
935
```

## 9.6. SWspectra\_v1.C (Lines 69-239 [20])



```

1 //Difference with provabea3.C: adding the integral of peak in peakposition_SW#.txt file
2 //Add also RMS
3
4 #include <string>
5 #include <cstring>
6 #include <stdlib>
7 #include <fstream>
8 #include <iostream>
9
10 using namespace std;
11
12 void SWSpectra_v1(int d, int m){
13
14     //-----
15
16     gROOT->SetStyle("Plain");
17     // background is no longer mouse-dropping white
18     gStyle->SetCanvasColor(kWhite);
19     // blue to red false color palette. Use 9 for b/w
20     gStyle->SetPalette(1,0);
21     // turn off canvas borders
22     gStyle->SetCanvasBorderMode(0);
23     gStyle->SetPadBorderMode(0);
24     // What precision to put numbers if plotted with "TEXT"
25     gStyle->SetPaintTextFormat("5.2f");
26
27     // For publishing:
28     gStyle->SetLineWidth(1.5);
29     gStyle->SetTextSize(1.1);
30     gStyle->SetLabelSize(0.05,"xy");
31     gStyle->SetTitleSize(0.05,"xy");
32     gStyle->SetTitleOffset(1.1,"x");
33     gStyle->SetTitleOffset(0.9,"y");
34     gStyle->SetPadTopMargin(0.1);
35     gStyle->SetPadRightMargin(0.1);
36     gStyle->SetPadBottomMargin(0.16);
37     gStyle->SetPadLeftMargin(0.12);
38
39     //-----
40
41
42
43     fstream peakposition1("C:/root/macros/SingleWire/OutputandPlots/peakpositiontest_SW1.txt",ios::out
... |ios::app); //|ios::app appende i dati al file
44     fstream peakposition2("C:/root/macros/SingleWire/OutputandPlots/peakpositiontest_SW2.txt",ios::out
... |ios::app); //|ios::app appende i dati al file
45     char title[600],title2[600],title3[600],title4[600];
46     char date[60];
47
48     //const char *dirname="/Users/beatricemandelli/Desktop/SWprova/";
49     const char *dirname="C:/DataSW/SW01-SW02c/";
50     const char *ext=".dat";
51     char namefile[100];
52     char cday[30],cmonth[30],cyear[30],chour[30],cmin[30],csec[30];
53     int year, month, day, hour, min, hour_0, hour_6, hour_12, hour_18;
54     int if0, if6, if12, if18;
55     float a1, a2, a3;
56     double mean1, mean2, mean1_6, mean2_6, test,test1;
57     TString fname, fname2, fnameDir;
58     int HnumBin=600; //only multiples of 150 seem to work well
59
60     //TH1F *h0[31][12], *h6[31][12], *h12[31][12], *h18[31][12];
61     TH1F *h0_1[32][12], *h6_1[32][12], *h12_1[32][12], *h18_1[32][12], *h0_2[32][12], *h6_2[32][12],
... *h12_2[32][12], *h18_2[32][12];
62     TCanvas *peakCanvas1 = new TCanvas ("peakCanvas1","Peaks SW1",1000,750);
63     peakCanvas1->Divide(2,2);
64     gStyle->SetOptStat(00000000);
65     TCanvas *peakCanvas2 = new TCanvas ("peakCanvas2","Peaks SW2",1000,750);
66     peakCanvas2->Divide(2,2);
67
68
69     TSystemDirectory dir(dirname, dirname); //TSystemDirectory(const char* dirname, const char* path)
70     TList *files = dir.GetListOfFiles();
71     //cout << "files" << endl;
72     //files -> Print();
73
74
75     if (files) {
76         TSystemFile *file;
77         //TString fname;
78         TIter next(files);
79         while ((file=(TSystemFile*)next()))
80         {
81             fname = file->GetName();
82

```

```

83     if (fname!="." && fname!=".." && fname!=".DS_Store") {
84         //cout << "fname after " << fname << endl;
85
86         // miei cambiamenti
87         fnameDir="C:/DataSW/SW01-SW02c/"+fname;
88         const char *dirname2=fnameDir;
89         //cout << "dirname2 " << dirname2 << endl;
90
91         TSystemDirectory dir2(dirname2, dirname2); //TSystemDirectory(const char* dirname, const char*
... path)
92         TList *files2 = dir2.GetListOfFiles();
93         //cout << "files2" << endl;
94         //files2 -> Print();
95
96
97         if (files2) {
98             TSystemFile *file2;
99             TIter next2(files2);
100
101             if0=0;
102             if6=0;
103             if12=0;
104             if18=0;
105             while ((file2=(TSystemFile*)next2())) {
106
107                 fname2 = file2->GetName(); //
108                 //cout << "fname2 " <<fname2 << endl;
109
110                 if (!file2->IsDirectory() && fname2.EndsWith(ext)) { //
111                     if (fname2[32] == 'r'){
112                         //prendo file raw
113                         //cout << "raw file " << fname2 << endl;
114                         sprintf(cyear, "%c%c",fname2[0],fname2[1]);
115                         year = atoi(cyear);
116                         sprintf(cmonth, "%c%c",fname2[3],fname2[4]);
117                         month = atoi(cmonth);
118                         sprintf(cday, "%c%c",fname2[6],fname2[7]);
119                         day = atoi(cday);
120                         sprintf(chour, "%c%c",fname2[9],fname2[10]);
121                         hour = atoi(chour);
122                         sprintf(cmin, "%c%c",fname2[12],fname2[13]);
123                         min = atoi(cmin);
124                         //cout << "hour " << hour << endl;
125
126                         // if (day == d && month == m) { //aggiunto
127
128                         //Edited by dhervas
129                         TString testfname2;
130                         testfname2=fnameDir+"/"+fname2;
131
132                         //hour 0
133                         if (hour == 0 && if0 <= 4){ //if0<=4
134
135                             hour_0 = hour;
136                             ifstream datafile;
137
138                             datafile.open(testfname2,ios::in);
139
140                             //cout << "testfname2 " << testfname2 << endl;
141
142                             if (min < 11) { //define title
143                                 sprintf(title,"SW1 %d-%d-2014-hour:%d", day, month, hour);
144                                 sprintf(title2,"SW2 %d-%d-2014-hour:%d", day, month, hour);
145                                 h0_1[d][m] = new TH1F(title,title,HnumBin,0,3000);
146                                 h0_2[d][m] = new TH1F(title2,title2,HnumBin,0,3000);
147
148                             }
149
150                             while (!datafile.eof()) {
151                                 datafile >> a1 >> a2 >> a3; //a1=SW1, a2=SW2, a3=shutter
152                                 //cout << "c " << a1 <<endl;
153                                 h0_1[d][m]->Fill(a1);
154                                 h0_2[d][m]->Fill(a2);
155                                 //cout << "fname2 " << fname2 << endl;
156                             }
157                             //cout << if0 << endl;
158                             if0++;
159                         }
160
161                     }
162
163                     //hour 6
164                     if (hour == 6 && if6 <= 4){
165
166                         hour_6 = hour;

```

```

167         ifstream datafile6;
168         datafile6.open(testfname2,ios::in);
169
170         if (min < 11) { //define title
171             sprintf(title,"SW1 %d-%d-2014-hour:%d", day, month, hour);
172             sprintf(title2,"SW2 %d-%d-2014-hour:%d", day, month, hour);
173             h6_1[d][m] = new TH1F(title,title,HnumBin,0,3000);
174             h6_2[d][m] = new TH1F(title2,title2,HnumBin,0,3000);
175         }
176
177         while (!datafile6.eof()) {
178             datafile6 >> a1 >> a2 >> a3;
179             //cout << "c2 " << a1 <<endl;
180             h6_1[d][m]->Fill(a1);
181             h6_2[d][m]->Fill(a2);
182         }
183         if6++;
184     }
185
186     //hour 12
187     if (hour == 12 && if12 <= 4){
188
189         hour_12 = hour;
190         ifstream datafile12;
191         datafile12.open(testfname2,ios::in);
192
193         if (min < 11) { //define title
194             sprintf(title,"SW1 %d-%d-2014-hour:%d", day, month, hour);
195             sprintf(title2,"SW2 %d-%d-2014-hour:%d", day, month, hour);
196             h12_1[d][m] = new TH1F(title,title,HnumBin,0,3000);
197             h12_2[d][m] = new TH1F(title2,title2,HnumBin,0,3000);
198         }
199
200         while (!datafile12.eof()) {
201             datafile12 >> a1 >> a2 >> a3;
202             h12_1[d][m]->Fill(a1);
203             h12_2[d][m]->Fill(a2);
204         }
205         if12++;
206     }
207
208     //hour 18
209     if (hour == 18 && if18 <= 4){
210
211         hour_18 = hour;
212         ifstream datafile18;
213         datafile18.open(testfname2,ios::in);
214
215         if (min < 11) { //define title
216             sprintf(title,"SW1 %d-%d-2014-hour:%d", day, month, hour);
217             sprintf(title2,"SW2 %d-%d-2014-hour:%d", day, month, hour);
218             h18_1[d][m] = new TH1F(title,title,HnumBin,0,3000);
219             h18_2[d][m] = new TH1F(title2,title2,HnumBin,0,3000);
220         }
221
222         while (!datafile18.eof()) {
223             datafile18 >> a1 >> a2 >> a3;
224             h18_1[d][m]->Fill(a1);
225             h18_2[d][m]->Fill(a2);
226         }
227         if18++;
228     }
229 }
230
231     }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240
241 //-----plot hour 0-----
242 //SW1
243
244 //testing histogram arrays
245 //TH1F *testhist;
246 //int testvar=3;
247 //testhist=&testvar;
248 //Th1f does not seem to work as 2d vector
249 //TH1F testarray[2][4]={*h0_1[d][m], *h6_1[d][m], *h12_1[d][m], *h18_1[d][m]}, {*h0_2[d][m], *h6_2[d][m],
250 *h12_2[d][m], *h18_2[d][m]}};

```

```

251 TH1F *testarray[8]={*h0_1[d][m], *h6_1[d][m], *h12_1[d][m], *h18_1[d][m], *h0_2[d][m], *h6_2[d][m],
... *h12_2[d][m], *h18_2[d][m]};
252 //TH1F *testarray[1];
253 //testarray[0]=*h0_1[d][m];
254
255 ///loop for canvas 1 SW1
256 double mean1[4];
257 double mean2[4];
258 double maximumbinY[4];
259 double maximumY[4];
260 double maximumX[4];
261 double RMS[4];
262 double integral[4];
263 double integralnoise[4];
264 double fit1RMS[4];
265 double fit2RMS[4];
266 double FWHM[4];
267 int rangethreshold=150;
268
269 for (int canvashour=0; canvashour<=3; canvashour++)
270 {
271
272 //***** Start histogram range (without pedestrian) calculation*****
273 int binrange=-1;
274 //int nbins = 1000;
275 int nbinsr = testarray[canvashour]->GetNbinsX();
276
277 int nabove = 1;
278
279 //Finding bin for first cross -> Skipping pedestrian
280
281 for (int binr=1; binr<=nbinsr; binr++)
282
283 {
284
285 if (testarray[canvashour]->GetBinContent(binr) > rangethreshold)
286
287 {
288
289 binrange=binr;
290
291 break;
292
293 }
294
295 }
296 //Finding bin for second cross -> Skipping pedestrian
297 for (int bin2r=binrange; bin2r<=nbinsr; bin2r++)
298
299 {
300
301 if (testarray[canvashour]->GetBinContent(bin2r) < rangethreshold)
302
303 {
304
305 binrange=bin2r;
306
307 break;
308
309 }
310
311 }
312 //Finding 3rd cross-> first point for range calculation
313 for (int bin3r=binrange; bin3r<=nbinsr; bin3r++)
314
315 {
316
317 if (testarray[canvashour]->GetBinContent(bin3r) > rangethreshold)
318
319 {
320
321 binrange=bin3r;
322
323 break;
324
325 }
326
327 }
328 //int binrange= testarray[canvashour]->FindFirstBinAbove(rangethreshold); // Activate whe there is no
... pedestrail in data
329 int bin2range= testarray[canvashour]->FindLastBinAbove(rangethreshold);
330 //binrange=binrange-3;
331 //bin2range=bin2range+3;
332 double point1range= testarray[canvashour]->GetXaxis()->GetBinCenter(binrange);
333 double point2range= testarray[canvashour]->GetXaxis()->GetBinCenter(bin2range);

```

```

334 //histRange = point2range-point1range;
335
336 cout<< point1range << endl << "HEY ITS A MEEEEEEEEEEEE 1" <<endl;
337 cout<< point2range << endl << "HEY ITS A MEEEEEEEEEEEE 2" <<endl;
338 //cout<< histRange << endl << "HEY ITS A MEEEEEEEEEEEE FWHM" <<endl;
339
340 //*****End range calculation*****
341
342
343 peakCanvas1->cd(canvashour+1);
344 testarray[canvashour]->GetXaxis()->SetRange(bin1range,bin2range); //40,100 for HnumBin 150
345 maximumbinY[canvashour] = testarray[canvashour]->GetMaximumBin();
346 maximumY[canvashour] = testarray[canvashour]->GetBinContent(testarray[canvashour]->GetMaximumBin());
347 //maximumX[canvashour] =testarray[canvashour]->GetXaxis()->GetBinCenter(maximumbinY[canvashour]);
348 double meanhist=testarray[canvashour]->GetMean(1);
349
350 //***** Start FWHM calculation*****
351 int bin1fwhm=-1;
352 //int nbins = 1000;
353 int nbins = testarray[canvashour]->GetNbinsX();
354
355 int nabove = 1;
356
357 //Finding bin for first cross -> Skipping pedestrian
358
359 for (int bin=1; bin<=nbins; bin++)
360
361 {
362
363     if (testarray[canvashour]->GetBinContent(bin) > maximumY[canvashour]/2)
364
365     {
366
367         bin1fwhm=bin;
368
369         break;
370
371     }
372
373 }
374 //Finding bin for second cross -> Skipping pedestrian
375 for (int bin2=bin1fwhm; bin2<=nbins; bin2++)
376
377 {
378
379     if (testarray[canvashour]->GetBinContent(bin2) < maximumY[canvashour]/2)
380
381     {
382
383         bin1fwhm=bin2;
384
385         break;
386
387     }
388
389 }
390 //Finding 3rd cross-> first point for fwhm calculation
391 for (int bin3=bin1fwhm; bin3<=nbins; bin3++)
392
393 {
394
395     if (testarray[canvashour]->GetBinContent(bin3) > maximumY[canvashour]/2)
396
397     {
398
399         bin1fwhm=bin3;
400
401         break;
402
403     }
404
405 }
406 //int bin1fwhm= testarray[canvashour]->FindFirstBinAbove(maximumY[canvashour]/2); // Activate whe there
... is no pedestrail in data
407 int bin2fwhm= testarray[canvashour]->FindLastBinAbove(maximumY[canvashour]/2);
408 double point1fwhm= testarray[canvashour]->GetXaxis()->GetBinCenter(bin1fwhm);
409 double point2fwhm= testarray[canvashour]->GetXaxis()->GetBinCenter(bin2fwhm);
410 FWHM[canvashour] = point2fwhm-point1fwhm;
411
412 //cout<< point1fwhm << endl << "HEY ITS A MEEEEEEEEEEEE 1" <<endl;
413 //cout<< point2fwhm << endl << "HEY ITS A MEEEEEEEEEEEE 2" <<endl;
414 //cout<< FWHM[canvashour] << endl << "HEY ITS A MEEEEEEEEEEEE FWHM" <<endl;
415 //*****End FWHM calculation*****
416
417

```

```

418
419
420 TF1 *f1 = new TF1("f1","gaus(0)+gaus(3)",point1range,point2range);
421 //testarray[canvashour]->GetXaxis()->SetRange(40,100);
422 f1->SetParameters(maximumY[canvashour],meanhist,50,maximumY[canvashour],meanhist,50);
423
424 //confining the mean of the fits to be on oposing sides of the mean of the histogram
425
426 //with max
427 //f1->SetParLimits(1,maximumX[canvashour]-testarray[canvashour]->GetRMS(),maximumX[canvashour]);
428 //f1->SetParLimits(4,maximumX[canvashour],maximumX[canvashour]+testarray[canvashour]->GetRMS());
429 //with mean
430
431 //double minlboundf1=meanhist-testarray[canvashour]->GetRMS();
432 //double max2boundf1=2*meanhist-minlboundf1;
433
434
435 f1->SetParLimits(1,point1range,meanhist);
436 f1->SetParLimits(0,0,maximumY[canvashour]);
437 f1->SetParLimits(4,meanhist,point2range);
438 f1->SetParLimits(3,0,maximumY[canvashour]);
439
440
... //f1->SetParLimits(1,testarray[canvashour]->GetMean(1)-testarray[canvashour]->GetRMS(),testarray[canvashour]->
GetMean(1));
...
441
... //f1->SetParLimits(4,testarray[canvashour]->GetMean(1),testarray[canvashour]->GetMean(1)+testarray[canvashour]-
>GetRMS());
...
442 //
443 //testarray[canvashour]->Fit("f1", "R+", " ", 700, 1600);
444 testarray[canvashour]->Fit("f1","RB");
445 //h0_1[d][m]->Fit("f1", "R+", " ", 700, 1600);//1000, 2000 without pedestrian
446 //h0_1[d][m]->SetMaximum(10000);
447 TF1 *g1 = new TF1("g1","[0]*exp(-0.5*((x-[1])/[2])**2)",point1range, point2range); //700, 1600 //1000,
... 2000
448 TF1 *g2 = new TF1("g2","[0]*exp(-0.5*((x-[1])/[2])**2)",point1range, point2range); //1000, 2000
449 g1->SetParameters(f1->GetParameter(0),f1->GetParameter(1), f1->GetParameter(2) );
450 g2->SetParameters(f1->GetParameter(3),f1->GetParameter(4), f1->GetParameter(5) );
451 g1->SetLineColor(2);
452 g1->SetLineWidth(2);
453 g1->Draw("SAME");
454 g2->SetLineColor(8);
455 g2->SetLineWidth(2);
456 g2->Draw("SAME");
457
458 float ciao = f1->GetMaximumX();
459 //cout << "ciao " << ciao << endl;
460 maximumX[canvashour] =ciao;
461
462 mean1[canvashour] = testarray[canvashour]->GetFunction("f1")->GetParameter(1);
463 fit1RMS[canvashour] = testarray[canvashour]->GetFunction("f1")->GetParameter(2);
464 //cout << rms1_0 << endl;
465 mean2[canvashour] = testarray[canvashour]->GetFunction("f1")->GetParameter(4);
466 fit2RMS[canvashour] = testarray[canvashour]->GetFunction("f1")->GetParameter(5);
467 //cout << rms2_0 << endl;
468
469 RMS[canvashour] = testarray[canvashour]->GetRMS(); //Get the RMS of the histogram
470 integral[canvashour] = testarray[canvashour]->Integral(bin1range,bin2range, "width"); //calculate the
... integral of the peak
471 integralnoise[canvashour] = testarray[canvashour]->Integral(0,bin1range, "width"); //calculate the
... integral of the noise
472 //to see pedestrian uncoment:
473 //testarray[canvashour]->GetXaxis()->SetRange();
474
475
476 peakposition1 << mean1[canvashour] << "\t" << mean2[canvashour] << "\t" << maximumX[canvashour] << "\t"
... << maximumY[canvashour]<< "\t" << RMS[canvashour] << "\t" << FWHM[canvashour] << "\t" << fit1RMS[canvashour] <<
... "\t" << fit2RMS[canvashour] << "\t" << integral[canvashour] << "\t" << integralnoise[canvashour] << "\t" <<
... canvashour*6 << "\t" << d << "\t" << m << "\t" << "13" << "\t" << endl; //year
477 cout << "maxY" << canvashour*6 << " " << maximumY[canvashour]<< endl;
478 cout << "RMS" << canvashour*6 << " " << RMS[canvashour]<< endl;
479 cout << "fit1RMS" << canvashour*6 << " " << fit1RMS[canvashour]<< endl;
480 cout << "fit2RMS" << canvashour*6 << " " << fit2RMS[canvashour]<< endl;
481 cout << "Xvalue" << canvashour*6 << " " << maximumX[canvashour]<< endl;
482 cout << "mean1_" << canvashour*6 << " " << mean1[canvashour]<< endl;
483 cout << "mean2_" << canvashour*6 << " " << mean2[canvashour]<< endl;
484 cout << "INTEGRAL" << canvashour*6 << " " << integral[canvashour]<< endl;
485 cout << "INTEGRAL NOISE" << canvashour*6 << " " << integralnoise[canvashour]<< endl;
486 }
487
488
489 double mean1_2[4];
490 double mean2_2[4];
491 double maximumbinY_2[4];
492 double maximumY_2[4];

```

```

493 double maximumX_2[4];
494 double RMS_2[4];
495 double integral_2[4];
496 double integralnoise_2[4];
497 double fit1RMS_2[4];
498 double fit2RMS_2[4];
499 double FWHM_2[4];
500
501 //loop for SW2
502
503 for (int canvashour2=0; canvashour2<=3; canvashour2++)
504 {
505
506     //***** Start histogram range (without pedestrian) calculation*****
507     int binlrange=-1;
508     //int nbins = 1000;
509     int nbinsr = testarray[canvashour2+4]->GetNbinsX();
510
511     int nabover = 1;
512
513     //Finding bin for first cross -> Skipping pedestrian
514
515     for (int binr=1; binr<=nbinsr; binr++)
516     {
517
518         if (testarray[canvashour2+4]->GetBinContent(binr) > rangethreshold)
519         {
520
521             {
522
523                 binlrange=binr;
524
525                 break;
526
527             }
528
529         }
530     //Finding bin for second cross -> Skipping pedestrian
531     for (int bin2r=binlrange; bin2r<=nbinsr; bin2r++)
532
533     {
534
535         if (testarray[canvashour2+4]->GetBinContent(bin2r) < rangethreshold)
536
537         {
538
539             binlrange=bin2r;
540
541             break;
542
543         }
544
545     }
546     //Finding 3rd cross-> first point for range calculation
547     for (int bin3r=binlrange; bin3r<=nbinsr; bin3r++)
548
549     {
550
551         if (testarray[canvashour2+4]->GetBinContent(bin3r) > rangethreshold)
552
553         {
554
555             binlrange=bin3r;
556
557             break;
558
559         }
560
561     }
562     int bin2range= testarray[canvashour2+4]->FindLastBinAbove(rangethreshold);
563     binlrange=binlrange-3;
564     bin2range=bin2range+3;
565     double pointlrange= testarray[canvashour2+4]->GetXaxis()->GetBinCenter(binlrange);
566     double point2range= testarray[canvashour2+4]->GetXaxis()->GetBinCenter(bin2range);
567     //histRange = point2range-pointlrange;
568
569     cout<< pointlrange << endl << "HEY ITS A MEEEEEEEEEE 1" <<endl;
570     cout<< point2range << endl << "HEY ITS A MEEEEEEEEEE 2" <<endl;
571     //cout<< histRange << endl << "HEY ITS A MEEEEEEEEEE FWHM" <<endl;
572
573     //*****End range calculation*****
574
575
576     peakCanvas2->cd(canvashour2+1);
577

```

```

578     testarray[canvashour2+4]->GetXaxis()->SetRange(bin1range,bin2range); //40,100 for bin 150
579     maximumbinY_2[canvashour2] = testarray[canvashour2+4]->GetMaximumBin();
580     maximumY_2[canvashour2] =
... testarray[canvashour2+4]->GetBinContent(testarray[canvashour2+4]->GetMaximumBin());
581     //maximumX_2[canvashour2]
... =testarray[canvashour2+4]->GetXaxis()->GetBinCenter(maximumbinY_2[canvashour2]);
582     double meanhist_2=testarray[canvashour2+4]->GetMean(1);
583
584     //***** Start FWHM calculation*****
585     int bin1fwhm=-1;
586     //int nbins = 1000;
587     int nbins = testarray[canvashour2+4]->GetNbinsX();
588
589     int nabove=1;
590
591     //Finding bin for first cross -> Skipping pedestrian
592
593     for (int bin=1; bin<=nbins; bin++)
594
595     {
596
597         if (testarray[canvashour2+4]->GetBinContent(bin) > maximumY[canvashour2]/2)
598
599         {
600
601             bin1fwhm = bin;
602
603             break;
604
605         }
606
607     }
608     //Finding bin for second cross -> Skipping pedestrian
609     for (int bin2=bin1fwhm; bin2<=nbins; bin2++)
610
611     {
612
613         if (testarray[canvashour2+4]->GetBinContent(bin2) < maximumY[canvashour2]/2)
614
615         {
616
617             bin1fwhm = bin2;
618
619             break;
620
621         }
622
623     }
624     //Finding 3rd cross-> first point for fwhm calculation
625     for (int bin3=bin1fwhm; bin3<=nbins; bin3++)
626
627     {
628
629         if (testarray[canvashour2+4]->GetBinContent(bin3) > maximumY[canvashour2]/2)
630
631         {
632
633             bin1fwhm = bin3;
634
635             break;
636
637         }
638
639     }
640     int bin2fwhm= testarray[canvashour2+4]->FindLastBinAbove(maximumY[canvashour2]/2);
641     double point1fwhm= testarray[canvashour2+4]->GetXaxis()->GetBinCenter(bin1fwhm);
642     double point2fwhm= testarray[canvashour2+4]->GetXaxis()->GetBinCenter(bin2fwhm);
643     FWHM_2[canvashour2] = point2fwhm - point1fwhm;
644
645     //cout<< point1fwhm << endl << "HEY ITS A MEEEEEEEEEE 1" <<endl;
646     //cout<< point2fwhm << endl << "HEY ITS A MEEEEEEEEEE 2" <<endl;
647     //cout<< FWHM_2[canvashour2] << endl << "HEY ITS A MEEEEEEEEEE FWHM" <<endl;
648     //*****End FWHM calculation*****
649
650
651     TF1 *f1 = new TF1("f1","gaus(0)+gaus(3)", point1range, point2range);
652
653     f1->SetParameters(maximumY_2[canvashour2],meanhist_2,50,maximumY_2[canvashour2],meanhist_2,50);
654
655     //confining the mean of the fits to be on oposing sides of the mean of the histogram
656
657     //with max
658
659     //f1->SetParLimits(1,maximumX_2[canvashour2]-testarray[canvashour2+4]->GetRMS(),maximumX_2[canvashour2]);
659     //f1->SetParLimits(4,maximumX_2[canvashour2],maximumX[canvashour2]+testarray[canvashour2+4]->GetRMS());

```



```

660 //with mean
661 //double min1boundf1_2 = meanhist_2-testarray[canvashour2+4]->GetRMS();
662 //double max2boundf1_2 = 2*meanhist_2-min1boundf1_2;
663
664 f1->SetParLimits(1,point1range,meanhist_2);
665 f1->SetParLimits(0,0,maximumY[canvashour2]);
666 f1->SetParLimits(4,meanhist_2,point2range);
667 f1->SetParLimits(3,0,maximumY[canvashour2]);
668 //
669 testarray[canvashour2+4]->Fit("f1","RB");
670 //h0_1[d][m]->Fit("f1", "R+", "", 700, 1600); //1000, 2000 without pedestrian
671 //testarray[canvashour2+4]->SetMaximum(40000);
672 TF1 *g1 = new TF1("g1","[0]*exp(-0.5*((x-[1])/[2])**2)", point1range, point2range); //700, 1600
... //1000, 2000
673 TF1 *g2 = new TF1("g2","[0]*exp(-0.5*((x-[1])/[2])**2)", point1range, point2range); //1000, 2000
674 g1->SetParameters(f1->GetParameter(0),f1->GetParameter(1), f1->GetParameter(2) );
675 g2->SetParameters(f1->GetParameter(3),f1->GetParameter(4), f1->GetParameter(5) );
676 g1->SetLineColor(2);
677 g1->SetLineWidth(2);
678 g1->Draw("SAME");
679 g2->SetLineColor(8);
680 g2->SetLineWidth(2);
681 g2->Draw("SAME");
682
683
684 float ciao2 = f1->GetMaximumX();
685 //cout << "cc " << ca << endl;
686 maximumX_2[canvashour2] =ciao2;
687
688 mean1_2[canvashour2] = testarray[canvashour2+4]->GetFunction("f1")->GetParameter(1);
689 fit1RMS_2[canvashour2] = testarray[canvashour2+4]->GetFunction("f1")->GetParameter(2);
690 //cout << rms1_0 << endl;
691 mean2_2[canvashour2] = testarray[canvashour2+4]->GetFunction("f1")->GetParameter(4);
692 fit2RMS_2[canvashour2] = testarray[canvashour2+4]->GetFunction("f1")->GetParameter(5);
693 //cout << rms2_0 << endl;
694
695 //testarray[canvashour2+4]->GetXaxis()->SetRange(40,100); //(40,100)range for peak with pedestrian
696 //maximumY_2[canvashour2] = testarray[canvashour2+4]->GetMaximumBin();
697 //maximumX_2[canvashour2]
... =testarray[canvashour2+4]->GetXaxis()->GetBinCenter(maximumbinY_2[canvashour2]);
698 RMS_2[canvashour2] = testarray[canvashour2+4]->GetRMS(); //Get the RMS of the histogram
699 integral_2[canvashour2] = testarray[canvashour2+4]->Integral(bin1range,bin2range, "width"); //calculate
... the integral of the peak
700 integralnoise_2[canvashour2] = testarray[canvashour2+4]->Integral(0,bin1range, "width"); //calculate
... the integral of the noise
701 //testarray[canvashour2+4]->GetXaxis()->SetRange();
702
703
704 peakposition2 << mean1_2[canvashour2] << "\t" << mean2_2[canvashour2] << "\t" <<
... maximumX_2[canvashour2] << "\t" << maximumY_2[canvashour2] << "\t" << RMS_2[canvashour2] << "\t" <<
... FWHM_2[canvashour2] << "\t" << fit1RMS_2[canvashour2] << "\t" << fit2RMS_2[canvashour2] << "\t" <<
... integral_2[canvashour2] << "\t" << integralnoise_2[canvashour2] << "\t" << canvashour2*6 << "\t" << d << "\t"
... << m << "\t" << "13" << "\t" << endl;
705 cout << "maxY" << canvashour2*6 << " " << maximumY_2[canvashour2]<< endl;
706 cout << "RMS" << canvashour2*6 << " " << RMS_2[canvashour2]<< endl;
707 cout << "fit1RMS" << canvashour2*6 << " " << fit1RMS_2[canvashour2]<< endl;
708 cout << "fit2RMS" << canvashour2*6 << " " << fit2RMS_2[canvashour2]<< endl;
709 cout << "Xvalue" << canvashour2*6 << " " << maximumX_2[canvashour2]<< endl;
710 cout << "mean1_" << canvashour2*6 << " " << mean1_2[canvashour2]<< endl;
711 cout << "mean2_" << canvashour2*6 << " " << mean2_2[canvashour2]<< endl;
712 cout << "INTEGRAL" << canvashour2*6 << " " << integral_2[canvashour2]<< endl;
713 cout << "INTEGRAL NOISE" << canvashour2*6 << " " << integralnoise_2[canvashour2]<< endl;
714 }
715
716
717 //save canvas
718
719 sprintf(title,"C:/root/macros/SingleWire/OutputandPlots/SW1PNG/Peaks_SW1_20%d-%d-%d.png", year, m, d);
720 peakCanvas1->SaveAs(title);
721 sprintf(title2,"C:/root/macros/SingleWire/OutputandPlots/SW1C/Peaks_SW1_20%d-%d-%d.C",year,m,d);
722 peakCanvas1->SaveAs(title2);
723
724
725 sprintf(title3,"C:/root/macros/SingleWire/OutputandPlots/SW2PNG/Peaks_SW2_20%d-%d-%d.png",year,m,d);
726 peakCanvas2->SaveAs(title3);
727 sprintf(title4,"C:/root/macros/SingleWire/OutputandPlots/SW2C/Peaks_SW2_20%d-%d-%d.C",year,m,d);
728 peakCanvas2->SaveAs(title4);
729
730 }
731

```

## References

- [1] The ROOT Team. (n.d.). ROOT: Data Analysis Framework. Retrieved April 13, 2015, from <https://root.cern.ch/drupal/>
- [2] The ATLAS Experiment. (2011). ATLAS Fact Sheet. Retrieved January 11, 2015, from [http://www.atlas.ch/pdf/ATLAS\\_fact\\_sheets.pdf](http://www.atlas.ch/pdf/ATLAS_fact_sheets.pdf)
- [3] Leo, W. (1987). *Techniques for Nuclear and Particle Physics Experiments*. Berlin: Springer-Verlag.
- [4] CERN Accelerating science. (n.d.). Retrieved January 11, 2015, from <http://home.web.cern.ch/about/how-detector-works>
- [5] Taking a closer look at LHC - LHC. (n.d.). Retrieved January 11, 2015, from <http://www.lhc-closer.es/1/3/13/0>
- [6] Particle detector. (n.d.). Retrieved April 21, 2015, from [http://en.wikipedia.org/wiki/Particle\\_detector](http://en.wikipedia.org/wiki/Particle_detector)
- [7] Haber, C. (Director) (2005, November 16). Tracking with Semiconductor Particle Detectors. Lecture conducted from University of California, Davis.
- [8] Particle Data Group. (2010). *Particle Physics Booklet*, Berkeley, Calif.: Lawrence Berkeley National Laboratory ;.
- [9] Raether, H. (1964). Electron Avalanches and Breakdown in Gases. *Butterworths Advanced Physics Series*.
- [10] Boggende, A., Brinkman, A., & Graaff, W. (1969). Comments on the ageing effect of gas-filled proportional counters. *Journal of Physics E: Scientific Instruments*, (2), 701-705.
- [11] Akesson, T. et al. (2003). Aging studies for the ATLAS Transition Radiation Tracker (TRT). *Nuclear Instruments and Methods in Physics Research A*, (515), 166–179.
- [12] Kowalski, T., & Mindur, B. (2003). Manifestation of aging effects in gas proportional counters. *Nuclear Instruments and Methods in Physics Research A*, (515), 180-184.
- [13] Spielberg, N., & Tsarnas, D. (1975). Counting rate dependent gain shifts in flow proportional counters. *Review of Scientific Instruments*, (46), 1086-1086.
- [14] Sputtering. (n.d.). Retrieved February 20, 2015, from <http://en.wikipedia.org/wiki/Sputtering>

- [15] Bouclier, R., Capeans, M., Garabatos, C., Sauli, F., & Silander, K. (1994). Effects of outgassing from some materials on gas chamber ageing. *Nuclear Instruments and Methods in Physics Research A*, (350), 464-469.
- [16] Capeans, M. (2003). Aging and materials: Lessons for detectors and gas systems. *Nuclear Instruments and Methods in Physics Research A*, (515), 73-88.
- [17] The ROOT Team. (n.d.). ROOT: Data Analysis Framework, About. Retrieved March 23, 2015, from <https://root.cern.ch/drupal/content/about>
- [18] B. Mandelli. (2014). *loop.bat* [UNIX shell].
- [19] B. Mandelli. (2014). *chisqtest.C*. [ROOT].
- [20] B. Mandelli. (2014). *SWspectra.C*. [ROOT].