Università degli Studi di Siena

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Fisica e Tecnologie Avanzate

Tesi di Laurea

18 Aprile 2011

A Large GEM detector prototype

Test-beam results and analysis

Candidato:
Elena Graverini

Relatore:
Prof. Nicola Turini

Anno Accademico 2009–2010

Contents

Abstract

My thesis presents the tests performed on a new Gas Electron Multiplier detector, capable of revealing electromagnetic-interacting particles. It is a large area prototype, developed using innovative techniques that allow for the splicing of several small GEM foils together to obtain a single large foil. A single-mask etching technique was used to overcome alignment problems.

The tests were performed at the CERN laboratories in Genève, in order to verify the prototype efficiency. It was also necessary to test the compatibility of a high-capacitance readout plane with the front-end electronics now used by the TOTEM experiment at LHC. The prototype was indeed conceived as a possible future replacement for the CSCs (Cathode Strip Chambers) TOTEM has been employing so far.

In this thesis, I will report on the manufacture of the detector in detail, as well as on the preliminary gain and noise tests. The readout electronics and the test beam setup will also be described. I will then give a thorough account of the test beam results in order to provide a complete characterization of the prototype. Finally, I will go through some of the algorithms exploited in the data analysis process. The pieces of code I wrote myself are collected in an Appendix.

The results we achieved will be then compared with those expected by previous generation GEM detectors. I will suggest some opportunities for further improvements, especially with regard to the better time resolution which will be needed, in the near future, in order to meet the expected LHC higher working rate.

Sommario

La mia tesi presenta la serie di test eseguiti su un nuovo rivelatore di particelle di tipo Gas Electron Multiplier, in grado di rivelare fotoni e particelle cariche. Si tratta di un prototipo di grande area, realizzato con tecniche innovative che permettono di unire più fogli GEM di dimensioni minori in un solo piano e di evitare problemi di allineamento in fase di foratura.

I test sono stati eseguiti nei laboratori CERN di Ginevra allo scopo di verificare le prestazioni del prototipo. Inoltre, era necessario testare la compatibilità di un piano di readout ad alta capacità con l'elettronica attualmente in uso nell'esperimento TOTEM presso LHC: il prototipo è stato infatti concepito nell'ottica di una futura sostituzione delle attuali camere CSC (Cathode Strip Chamber) di TOTEM con rivelatori a GEM di grande area.

Esporrò in dettaglio il processo di costruzione del prototipo, i test preliminari di guadagno e l'analisi del rumore; verrà descritta l'elettronica di readout, per poi passare all'allestimento dei test su fascio di particelle. Esaminerò quindi i risultati ottenuti in questa fase, nell'ottica di un'integrale caratterizzazione del rivelatore. Spiegherò inoltre il funzionamento di alcuni degli algoritmi utilizzati per l'analisi dei dati raccolti durante il test beam. In appendice ho raccolto, in particolare, quelli che io stessa ho elaborato.

I risultati ottenuti verranno infine confrontati con le prestazioni attese dai rivelatori GEM di precedente generazione, e suggerirò le ulteriori migliorie che possono essere apportate in futuro, soprattutto per aumentarne la risoluzione temporale in vista del previsto incremento della frequenza di lavoro di LHC.

Chapter 1

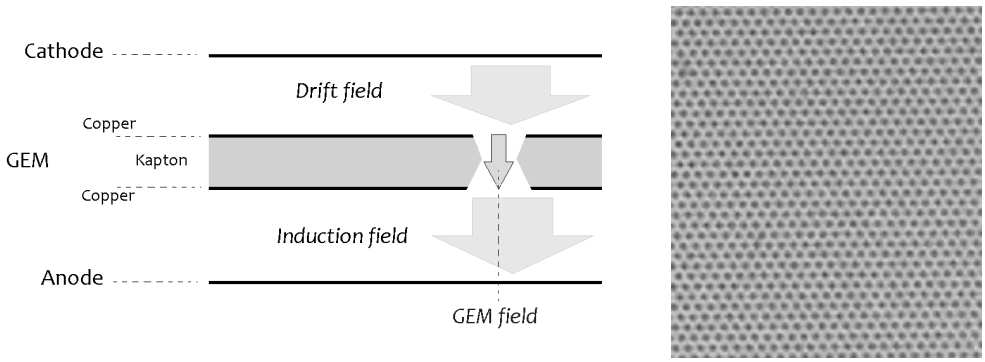
The prototype: a large GEM

1.1 GEM detectors

High-energy physics studies the kinetic features of particles and their interactions by using detectors which reveal, for example, the product of collisions in accelerators, or detect cosmic rays. Different kinds of detectors are used to reveal different properties of the particles themselves: kinetic energy, momentum, charge, mass and even their internal structure.

Gas ionization detectors can reveal high-rate electromagnetic-interacting particles. If the incident photon or charged massive particle has enough energy to ionize one atom of gas inside the detector, the flux of ion-electron pairs will be amplified; this will generate a current pulse for each event. Each signal can be revealed by a copper anode and can then be sent to electronic *data acquisition* (DAQ) systems.

Gaseous detectors can produce an avalanche multiplication of the signal by accelerating (through an electric field) the first electrons produced by the incident particle, so that they release more pairs. Gas electron multipliers (*GEMs*) work according to such principle. *GEM foils* consist of two thin copper layers, with a kapton foil between them. Small holes are made in a regular pattern through these foils, via chemical processes such as photoli-



(a) Draft of a single-GEM detector **(b)** Hole pattern

Figure 1.1: Schematic outline of a single-GEM detector and, on the right, view of the hole pattern in a GEM foil

tography and acid etching (Figure 1.1(b)).

A potential is then applied between the two copper layers (*anode* and *cathode*) in order to put the GEM foil into operation; this will create a high electric field through the holes:

$$E = \frac{V}{d}$$

where E is the electric field, V is the voltage and d is the distance between the anode and the cathode. Since the kapton foil is very thin (around $25\mu m$), but it can resist high electric fields[1], the potential between GEM electrodes can be as high as $500 \div 600V$.

A GEM-based detector consists of one or more GEM foils laid, as shown in Figure 1.1(a), between a copper foil (*cathode*) and a readout plane (*anode*).

A *drift field* is applied to drive electrons from the cathode drift foil to the GEM. An *amplification voltage* between the copper layers of every GEM foil accelerates the electrons, producing avalanche multiplication. Finally, an

[1]Kapton is a plastic polyimide created by DuPont, whose molecular structure remains stable in a wide range of temperatures. It was dicovered to be very radiation-hard, and is commonly used for its good insulating power [9].

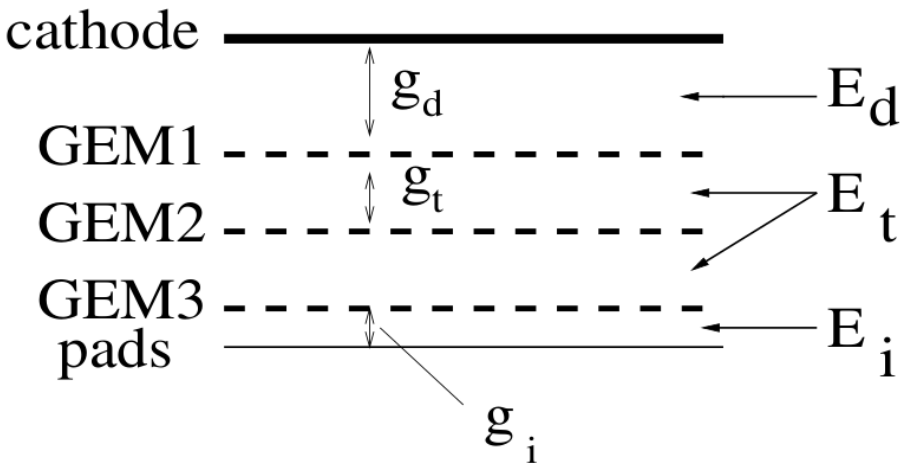


Figure 1.2: Draft cross-section of a triple GEM detector [7]

induction field leads them from the GEM bottom layer to the readout plane. In the case of a multiple GEM detector, *transfer* fields drive electrons from a GEM foil to the next.

More GEM foils can be inserted as a cascade in a single detector. This allows to reduce the discharge probability, as we can set quite a low multiplication potential between the electrodes of each foil. The discharge probability is indeed a function of charge density inside the GEM holes. In this configuration, the typical gains of each GEM are around $G \simeq 15 \div 30$; for a triple-GEM detector this leads to a total gain of the order of magnitude of several thousands (for example, for TOTEM **T2** telescope chambers $G \simeq 8,000$).

Figure 1.2 shows the usual cross section of a triple GEM chamber. E_d, E_t and E_i stand for the drift, transfer and induction fields, respectively. Gap depths g_d, g_t and g_i are customizable. In particular, stretching the drift gap allows to increase the number of primary ionizations, but on the other hand it decreases the time performance. Specific application requirements determine the most fitting chamber geometry.

The detector being analyzed in this thesis is a triple GEM chamber, whose gaps were set as in Table 1.1 on the next page; Table 4.1 on page 40

Drift gap	**Transfer gap 1**	**Transfer gap 2**	**Induction gap**
$g_d = 3mm$	$g_{t_1} = 2mm$	$g_{t_2} = 2mm$	$g_i = 2mm$
$E_d^{MIN} = 2.7\frac{kV}{cm}$	$E_{t_1}^{MIN} = 4.0\frac{kV}{cm}$	$E_{t_2}^{MIN} = 4.0\frac{kV}{cm}$	$E_i^{MIN} = 4.0\frac{kV}{cm}$
$E_d^{MAX} = 3.0\frac{kV}{cm}$	$E_{t_1}^{MAX} = 4.4\frac{kV}{cm}$	$E_{t_2}^{MAX} = 4.4\frac{kV}{cm}$	$E_i^{MAX} = 4.4\frac{kV}{cm}$

Table 1.1: Large GEM gap depths and electric fields

shows instead the gain values we reached during the tests conducted on this chamber. Table 1.1 also shows the range of electric fields we applied to each gap during the test-beam.

1.2 Choice of the gas

Interaction of incident charged particles or photons with matter is the basis on which Gas Electron Multiplier detectors work. A gas mixture flows inside the detector, or is sealed in it; its components are chosen according to their ionization potential, ion transport characteristics and ageing properties.

In most cases, only the electromagnetic (Coulomb) interaction is used for detection purposes. Strong and weak interactions are by far less probable, and in any case they cannot be revealed in a GEM. Coulomb interactions result in excitation and ionization of the gas molecules, and in negligible part in bremsstrahlung, Čerenkov and transition emission of radiation. In particular, noble gases do not present many energy dissipation modes: they can only be excited through photon absorption (and consequent emission). The absence of the many non-ionizing dissipation modes, which are typical of polyatomic molecules, is then the reason for which noble gases are chosen as main component in any gaseous detector. Ionization is indeed the main mode of interaction occurring in noble gases, so electron avalanche multiplication is eased. On the other hand, specific experimental requirements often oblige the use of compounds containing polyatomic molecules.

As already said, excited noble gases can only return to the ground state through emission of a photon. Argon, for example, releases photons with energy $E_\gamma \geq 11.6eV$, that are themselves able to ionize copper atoms of the GEM anodes (whose ionization potential equals $W_i = 7.7eV$). Argon positive ions drift instead towards the cathode, where they recombine extracting an electron and causing secondary photons or electrons to be emitted as energy balancement. Both these processes result in spurious avalanches; even with a moderate gain, detectors may therefore experience frequent discharges. In addition, spurious signals such those would affect the space and time resolution.

We thus need to add a percentage of so-called **quencher**s to the gas mixture. Hydrocarbons, alcohols and freons are examples of the most used quenchers. Their complex molecules feature a large number of non-radiative excitation modes and allow photon absorption in the range of energy of those emitted by argon. Elastic collisions and quencher dissociation into simpler molecules dissipate the energy in excess, ensuring stability of operation of the detector. Despite this, attention must be paid to the process through which polyatomic ionized molecules neutralize. In fact, if radicals use to combine together to form larger polymers, precipitate and accumulate inside the chamber, that might speed up its ageing process.

The expression obtained by Bethe and Bloch for the average differential energy loss due to Coulomb interactions is:

$$\frac{dE}{dx} = -\frac{2\pi N z^2 e^4}{mc^2} \frac{Z}{A} \frac{\rho}{\beta^2} \left(\ln \frac{2mc^2 \beta^2 E_M}{I^2 (1-\beta^2)} - 2\beta^2 \right)$$

where N is the Avogadro number, Z, A, ρ and I are the atomic number, atomic mass, density and average ionization potential of the medium, z and β are the atomic number and speed in units of c of the projectile [14]. E_M represents the maximum transferable energy. Figure 1.3(a) on page 15 shows, as an example, the contributions of various processes to the total energy loss

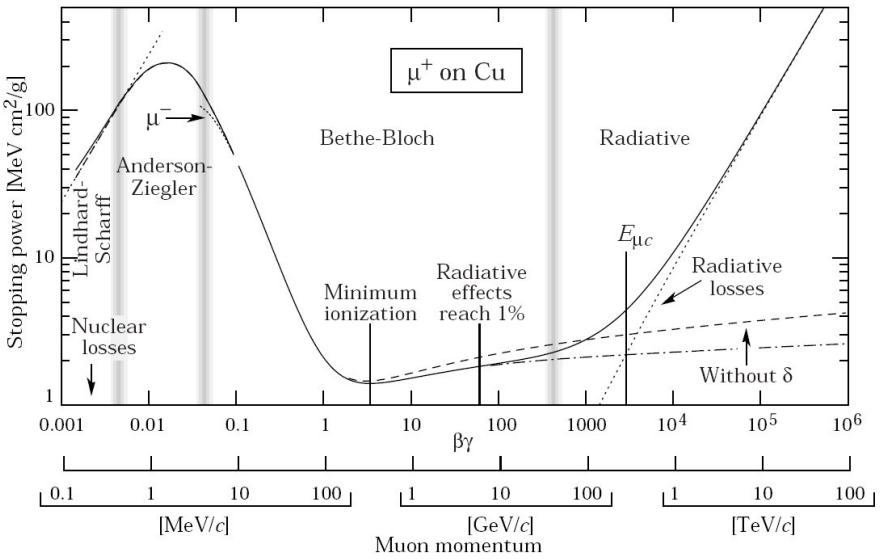
of muons μ^+ in copper.

Above a kinetic energy of the order of magnitude of a few hundred MeV every particle lies in the *minimum ionization* region of the curve (Figure 1.3(b) on the next page) and release on average $2\frac{MeV \cdot cm^2}{g}$. At those energy levels, which is the most common situation in high-energy physics, the particles are called *MIP*s.

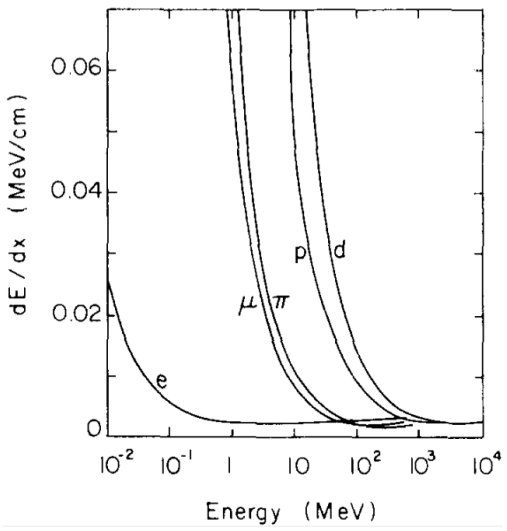
A number of ionization clusters are created on the passage of a MIP through the drift gap of the detector. Drift and transfer fields drive the ejected electrons through the GEM foils, where they are accelerated and cause more ionizations. The induction field attracts the electron avalanche on the anode, where each drifting electron turns into a current signal of intensity $I = \frac{e}{\Delta t_{drift}}$ and duration $\Delta t_{drift} = \frac{l_i}{v_{drift}}$, where l_i is the thickness of the induction gap and v_{drift} is the drift velocity of electrons (v_{drift} depends on the voltage applied to the gaps).

The total signal induced by an incident particle is given by the superposition of the signals produced by the primary ionization electrons (the ones extracted directly by the MIP crossing the drift region) multiplied by the total gain factor. The signals coming from different primary electrons will be detected at different times, because the spacial distribution of the ionization clusters can be as wide as the drift gap. The time resolution of the detector is determined by the space distribution of each ionization cluster, which follow the Poisson law, with a sigma $\sigma(t) \propto \frac{1}{n \cdot v_{drift}}$ (where n is the average number of primary ionization clusters) [7].

In order to optimise the time resolution of a detector it is therefore necessary to maximize the drift velocity, and the number of primary clusters, using gas mixtures with a large average atomic number. The gas inside the detector should allow a compromise between these two features and the minimum possible loss of efficiency as compared to a noble gas. A higher drift velocity



(a) An example of $-\frac{dE}{dx}$ of projectiles in a medium according to the Bethe-Bloch formula, from [1]. Energy loss is plotted as a function of $\beta\gamma = \frac{p}{Mc}$



(b) Energy loss in air for different particles as a function of their energy, from [14]

Figure 1.3: Energy loss due to Coulomb interaction of particles in media

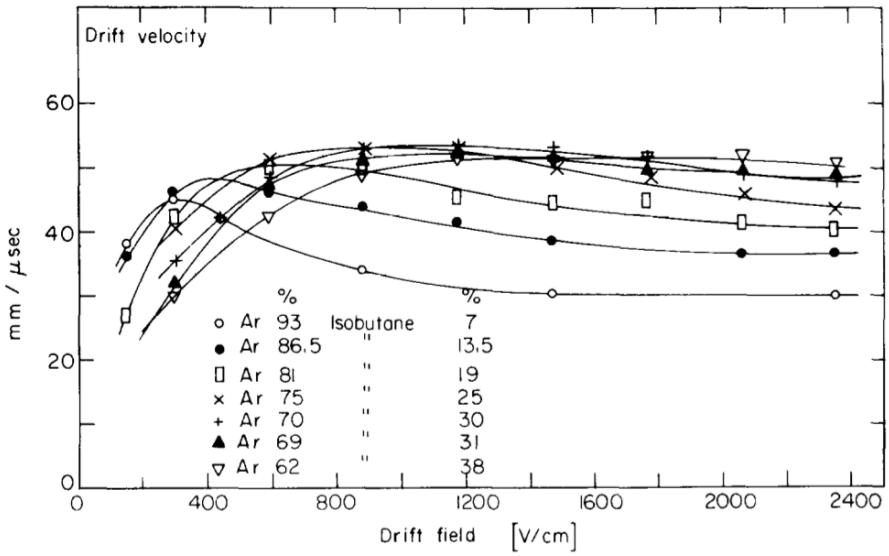
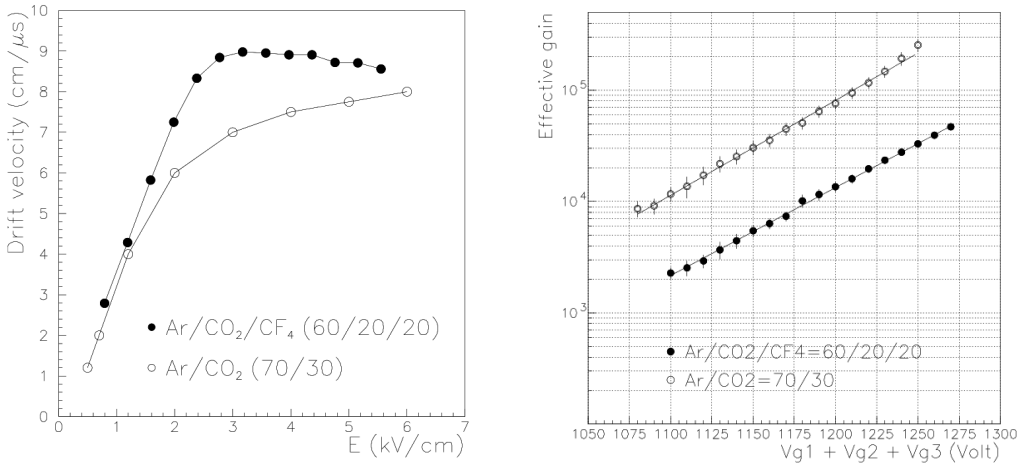


Figure 1.4: Drift velocity of electrons in several argon-isobutane mixtures

may be obtained by either increasing the drift, transfer and induction fields or adding a percentage of polyatomic molecules (a full explanation of this topic can be found in [14]).

The measurements presented in this text were performed using two gas mixtures: Ar/CO_2 70/30 and $Ar/CO_2/CF_4$ 60/20/20, with CO_2 and CF_4 as quenchers. The choice of CF_4 for the last set of tests was driven by its good timing characteristics, already verified by the authors of [7]; moreover, with respect to methane and other hydrocarbons, it is safer. CF_4 is indeed non-flammable, non-toxic and non-corrosive either for metal or plastic materials. It might only be harmful if it comes into contact with hydrogen, as this would allow for the formation of hydrofluoric acid HF. Ar/CO_2 70/30 is instead the "standard gas mixture" in use at TOTEM and other experiments.

Figure 1.5 on the facing page shows the differences between Ar/CO_2 70/30 and $Ar/CO_2/CF_4$ 60/20/20 as found in literature; our measurements will be presented in Chapter 4.2 on page 40 for what concerns the relationship between GEM fields and gain, and in Chapter 4.4 on page 49 I will show and discuss the timing tests we performed.



(a) Electron drift velocity

(b) Triple GEM gain

Figure 1.5: Properties of different gas mixtures. The measurements were done for a triple GEM detector with fields $E_d = E_t = 3\frac{kV}{cm}$ [7]

1.3 A large triple GEM detector

At CERN, a prototype triple GEM detector (Large GEM, abbreviated LG) was built[2] with an active area of about $2000cm^2$ using $66 \cdot 66cm^2$ kapton copper-clad foils [13]. Two innovative techniques were used to manufacture this detector: *single-mask* etching and *GEM foils splicing*.

GEM foils are manufactured through the same photolitographic processes used to produce common printed circuit boards. Tipically, small GEMs are etched on both sides to obtain symmetric holes. However, it can very uncomfortable to align masks when the GEM foil dimensions grow. Due to the very large area of this prototype, it was necessary to renew a single-mask etching technique. As shown in Figure 1.6 on the next page, after the top copper layer is etched, a basic mixture containing potassium hydroxide (KOH, which etches isotropically) and ethylene diamine ($C_2H_4(NH_2)_2$, which etches anisotropically) was used to make the holes in the polymide material (Kapton). An anisotropic etcher was needed to keep the holes aspect ratio large, defined as $\frac{depth}{width}$.

[2]Design by S. D. Pinto, CERN, on behalf of RD51 group.

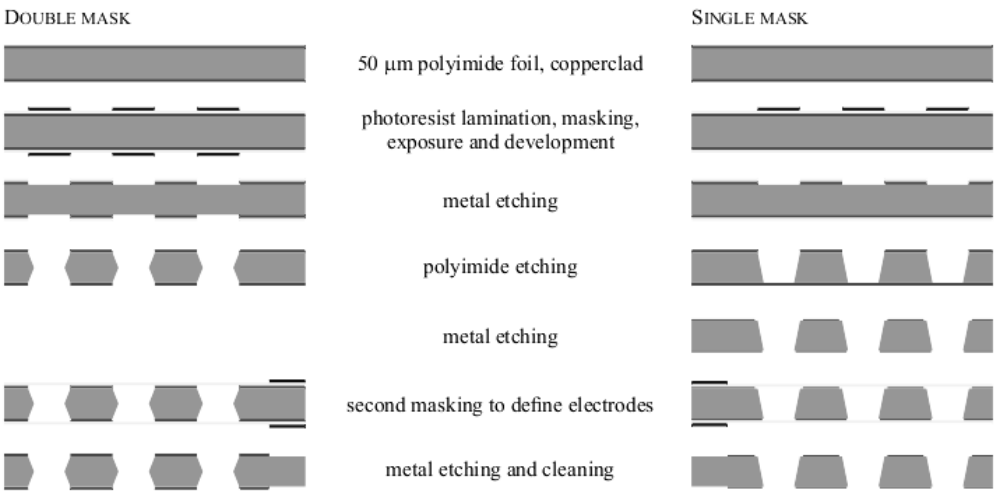


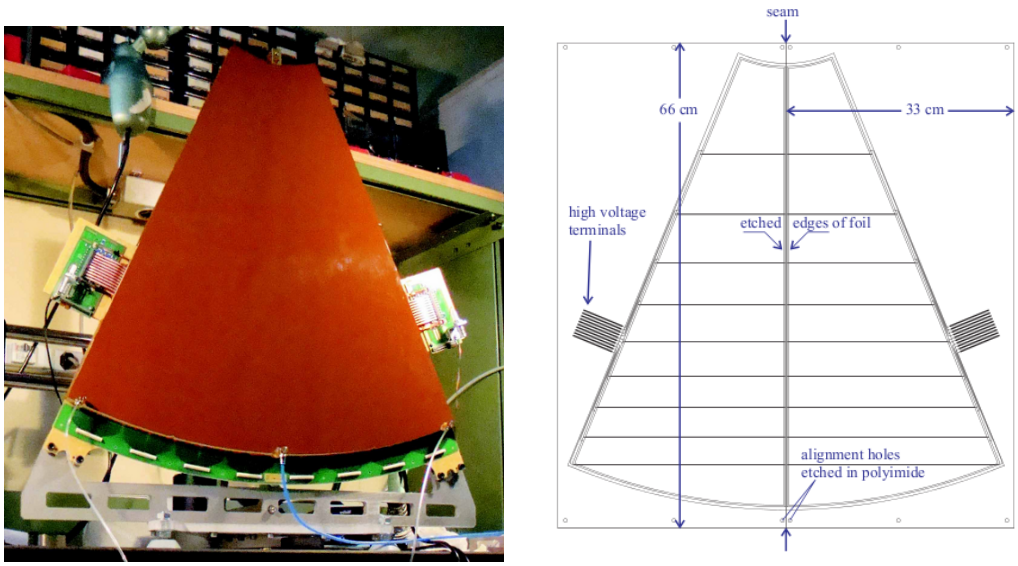
Figure 1.6: Comparison of the standard *double-mask* and the new *single-mask* GEM etching procedures [13]

The bottom copper layer was then etched on both sides, using the holes in the kapton as masks, dipping the whole foil in an acidic etchant mixture in order to finalize the holes and to reduce the thickness of the electrodes.

When the detector was built, suppliers of the GEM foils base material could only offer $50 \div 70cm$ wide rolls. A $66x66cm^2$ foil was cut crosswise, and the two halves were spliced together by covering the junction with a $25\mu m$ Kapton adhesive substrate. The glue polymerized by baking.

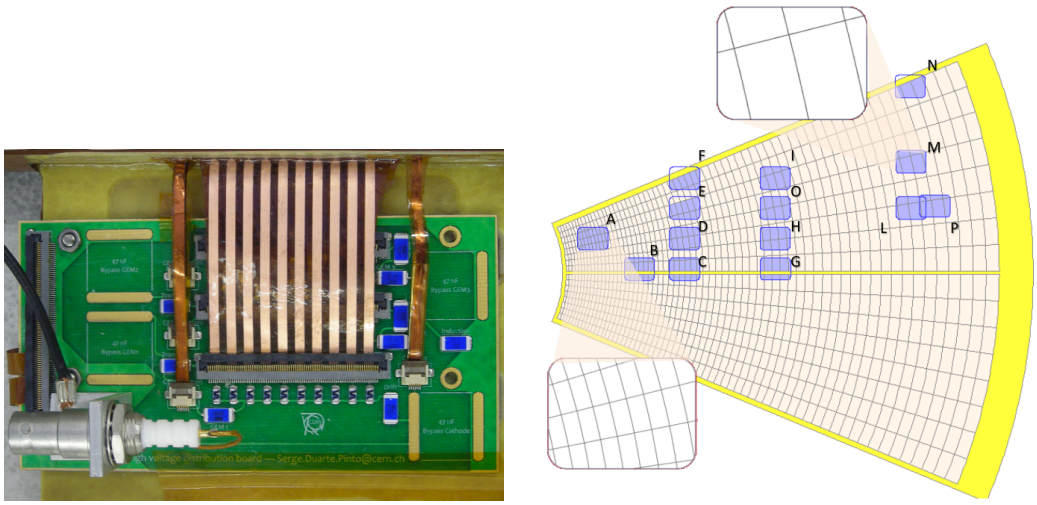
Rate capability tests demonstrated that the chamber performance remains unaffected except for the $2mm$ wide seam zone [12]. In addition, a $3.5mm$ wide plastic spacer was placed, in the triple GEM detector we tested, exactly over the foils junction areas, covering them totally. This made it impossible to recognize the loss of efficiency due to the seam.

Foreseeing the possibility of discharges, the cathode electrodes are segmented and connected to the power supply via $10M\Omega$ resistors in order to reduce their energy storage. A compact divider board supplies high voltage,



(a) View of the prototype **(b)** Layout of the GEM foils sectors

Figure 1.7: A photo and a scheme of the large area triple GEM prototype



(a) Divider board **(b)** Readout plane, made of pads

Figure 1.8: A view of the compact divider board, and a sketch of the pad-based readout showing the beam-tested regions

in order to make it easier to debug the circuit if needed. Since the board was not designed to handle such high voltages, groups of pins were connected together and alternated with groups of floating strips (see Figure 1.8(a) on the preceding page).

The readout configuration consists of 1024 pads, with a surface of $0.25cm^2$ (in the narrowest part of the detector) to $6cm^2$. Figure 1.8(b) on the previous page shows the different pad dimensions; the image also evidences the zones of the chamber where we directed the beam during the tests.

Small hybrid printed circuit boards bound to the chips themselves connect the readout VFAT2s to the detector, on its outer round edge.

1.3.1 Gain curve

Before the scheduled test beam period, Serge Duarte Pinto[3] had performed some tests to study the gain of the prototype, using Ar/CO_2 70/30 gas mixture and Cu X-Rays.

The gain of the detector represents the ratio between its output and input currents, that is:

$$G = \frac{I_{OUT}}{q * f}$$

where q is the charge gathered by the first layer of the detector. In this setup q can be computed as:

$$q = n * e$$

where e is the electron charge and n is the average number of electrons produced in the drift region by the incident photon or charged particle; f is the interaction rate of the incident particles in the gas. Since we know the energies of the characteristic X-ray lines of the source, and the ionization potential of the gas mixture in use, n is well determined. In this setup (Ar/CO_2 70/30) we have $n \simeq 290$.

[3] CERN researcher, on behalf of RD51 group

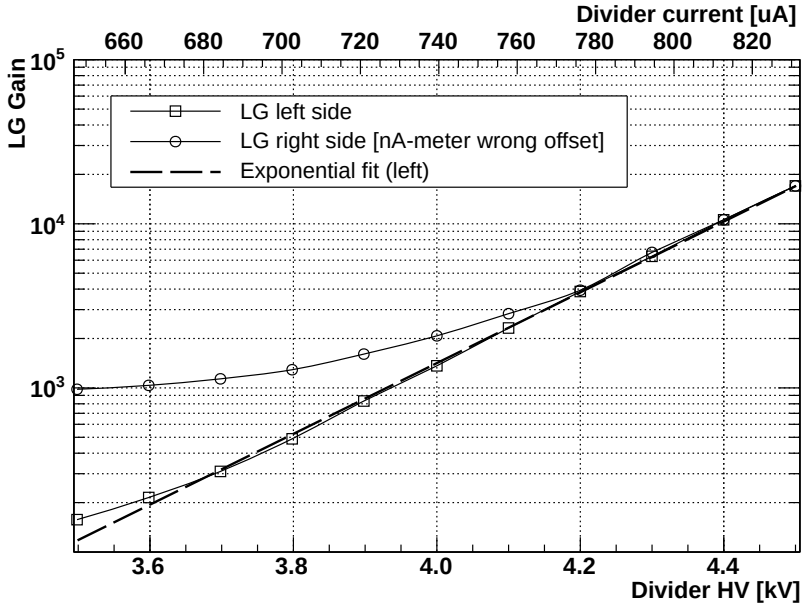


Figure 1.9: Cu X-Ray gain curve for the large prototype GEM detector (S. D. Pinto)

After q is computed, we need to measure I_{OUT} and f in order to obtain G. The interaction rate f is measured by sending the LG output signals first to an amplifier, then to a comparator, and finally to a counter. Connecting all the 128 pads of each readout line together, I_{OUT} can be measured with an amperometer.

Figure 1.9 shows the results of S. D. Pinto's measurements. There are two different plots for the two sides of the chamber; the graphs are not symmetric due to a wrong offset of one of the amperometers, which becomes relevant at low current levels. However, the high-gain regions of the two graphs coincide, and they can be fitted fairly well with an exponential function, as expected for a GEM detector.

At the test beam we raised the current over $830\mu A$, where Pinto's data end, losing the possibility to check the exact gain our detector had. We got to such high voltage values in order to test the prototype with a different gas mixture (see Chapter 4.2 on page 40 and following).

For this and other reasons, a new gain curve is required, which will touch high divider current values and will check the uniformity of the detector within its left and right sides.

Chapter 2

Readout electronics

2.1 VFAT2 readout chip

VFAT2 is the front-end electronics developed for the TOTEM experiment at the LHC. This ASIC (*Application Specific Integrated Circuit*) converts the signals from the detectors into digital data through an amplifier-shaper-comparator chain.

The readout chips used during the test-beam were VFAT2 chips as well, since one of the aims of the test was a compatibility check between large capacitance pad-based readout systems and the TOTEM front-end electronics.

VFAT2 features a *transimpedance* preamplification step ($V_{OUT} \propto I_{IN}$), whose output is sent to a shaper, and then compared to a customizable threshold potential. The latter is programmable in terms of *VFAT2 DAC step bins*[1]. A feature named *TrimDAC* allows us to set different thresholds for each channels, optimizing therefore the process of data acquisition even in noisy environments; however, we did not use this feature during the tests.

The monostable block then synchronizes the output of the comparator,

[1]Each *VFAT2 DAC step bin* corresponds to $3.3mV$, that is an input charge of about $0,045fC \approx 281e^{-}$.

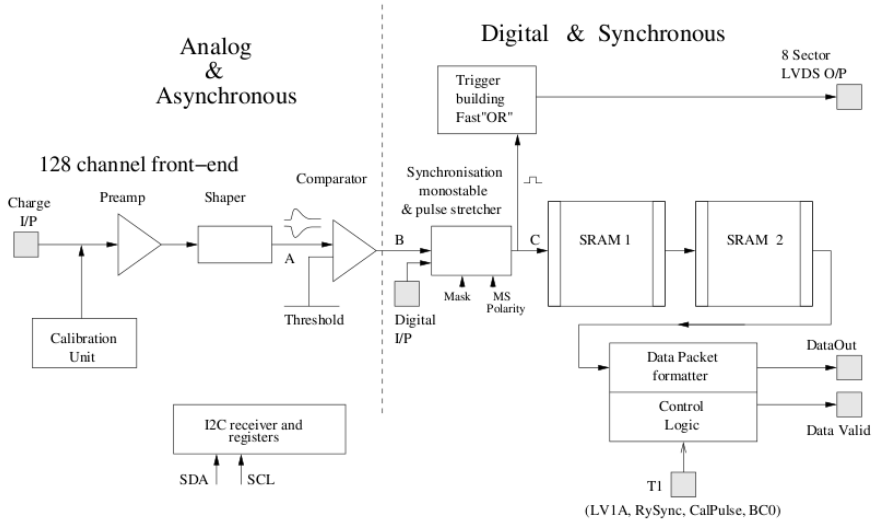


Figure 2.1: Block diagram of the VFAT2 chip [3]

providing by default a $1clk$ pulse for each threshold-crossing signal.

A *Fast-OR* logic provides an `OR` of the monostable outputs of a programmable number of channels in just one clock cycle. This so-called *S-Bit* can then be used as a trigger, making the VFAT2 an autotriggering front-end chip.

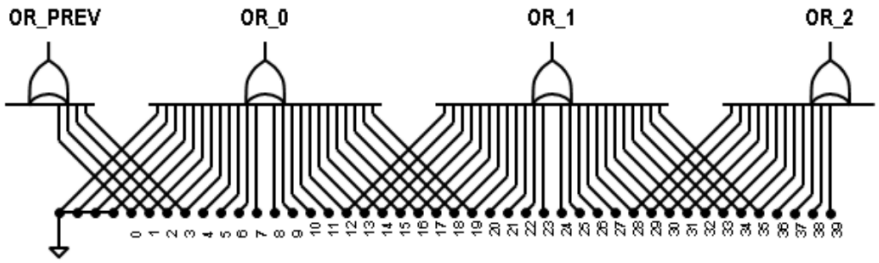
Threshold

It is necessary to set a threshold in order to prevent undesired hits[2] (i.e. noise) that would affect our data. VFAT2 allows to set both positive and negative thresholds, by changing the values of two registers V_{T1} and V_{T2}:

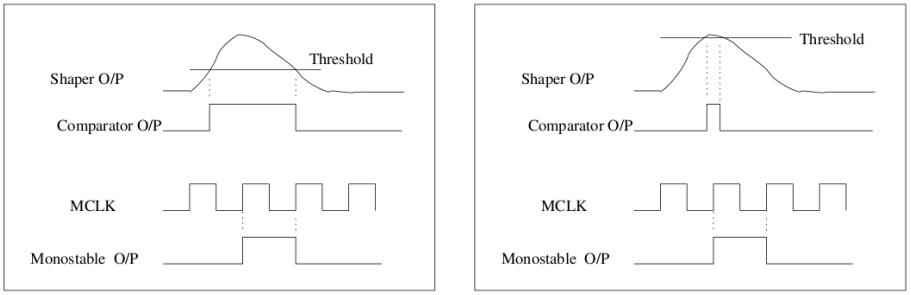
Positive threshold	$V_{T1} = 0$	$V_{T2} = variable$
Negative threshold	$V_{T1} = variable$	$V_{T1} = 0$

Table 2.1: Threshold settings in VFAT2

[2] A *hit* will be later defined as a channel that overcomes the threshold.



(a) Fast-OR logic combining the monostable outputs to provide a trigger signal within one clock cycle [2]



(b) Signal shaping for $MSPL = 1clk$ [5]

Figure 2.2: Fast trigger and shaping features of VFAT2 chips

The threshold is then computed as $V_{TH} = V_{T2} - V_{T1}$.

Most of our scans were run at $th = -60ds$ (where ds stands for *VFAT2 DAC step bin*). Threshold scans were performed as well, to see at what level the noise starts to become significant: see Chapter 4.3 on page 44.

Latency

Given a trigger signal, we define **latency** the number of SRAM locations the chip has to go back in order to read the digital output of the event corresponding to that trigger. It is measured in clock periods, as it represents the elapsed time between the arrival of the trigger and the preceding storage of the corresponding event in the VFAT2 SRAM.

The latency of our setup was found to be $lat = 17clk$ when working with Ar/CO_2 70/30, and $lat = 18clk$ when working with $Ar/CO_2/CF_4$ 60/20/20.

Monostable settings

A signal can have either a thin or a large time distribution of charge (*jitter* effect); the preamplifier stage preserves its $\frac{amplitude}{width}$ ratio. The monostable block allows to stretch its output pulse, which can be programmed to be as long as 1 to 8 clock periods. Stretching the monostable pulse generally decreases the time performance, as a subsequent signal may cross the threshold while the monostable output is still high. This would result in a single pulse instead of two, which would stay high for n more clock cycles after the last threshold crossing, $1 \leq n \leq 8$.

However, stretching the monostable pulse allows us to find out whether two or more pulses are to be recognized as originated by the same event. Two undesired effects may happen indeed, which are countered by stretching the monostable output:

- **Timewalk**: adjacent channels may be hit with different intensity (e.g. an electron avalanche falls over two adjacent pads): in this case, the channels which gather less charge may cross the threshold later than the other ones;
- **Jitter**: primary ionization clusters may be produced anywhere in the drift region, which has a significant thickness. Electrons released by the same incident particle at different heights in the drift region will thus reach the anode at different times; for a $3mm$ thick gap, and fields like those of Table 1.1 on page 12, the arrival time spread of drifting electrons can reach $60ns$.

By stretching the monostable we can then synchronize several detectors being crossed by the same particle. In addition, a single latency value would allow us to retrieve the signals produced by all the tracks, despite possible timewalk-jitter delays.

2.2 Turbo readout card

We used two **turbo** cards to control the VFAT2 chips during the tests.

Developed on the basis of the TOTEM Test Platform *TTP*[3], *turbo* is a stand-alone portable control and DAQ platform for front-end VFAT2 chips. It was designed by Dr. Roberto Cecchi and Dr. Maria Grazia Bagliesi[4]; an outline of the card is shown in Figure 2.3 on the following page.

Each *turbo* hosts an ALTERA Stratix II FPGA, which can interface with up to 8 VFAT2 chips via I^2C lines; configurations allowing to control more chips are currently under development. The card also features a Bitwise Systems *Quick-USB* interface, so that it can be controlled via PC.

[3] *TTP*s are platforms aimed at checking the functioning of VFAT chips and hybrids, in use at TOTEM.

[4] University of Siena

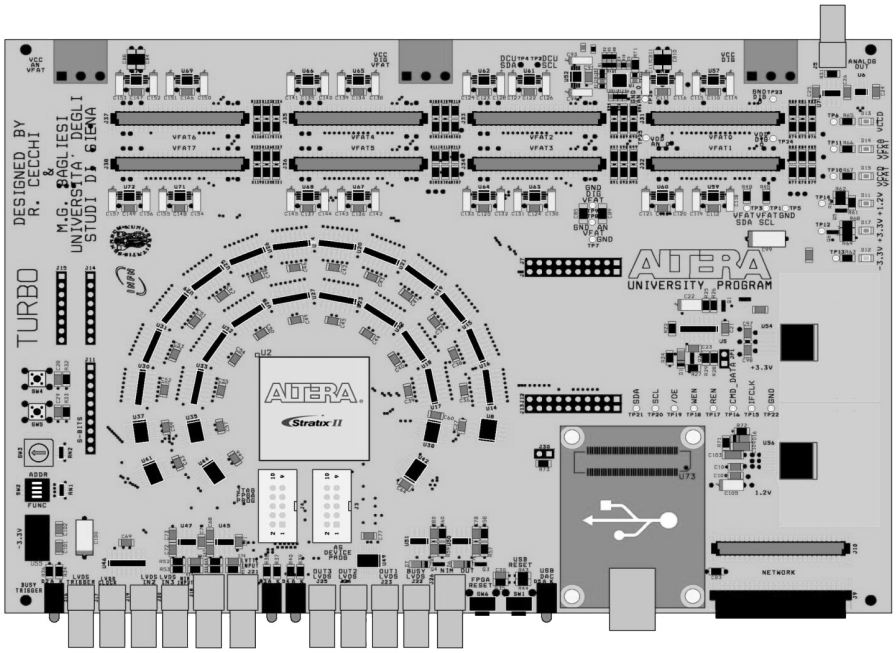


Figure 2.3: Draft of the *turbo* card

LabVIEW softwares were programmed to automate standard scans such as threshold, latency and calibration pulse scans; data acquisition and quality monitor programs can also be run through a PC connected to the card.

2.3 Off-beam threshold scan

Preliminary noise measurements were performed in the laboratory through a threshold scan, sending commands to a *turbo* card to control VFAT2 chips. This means that the number of hits was measured as a function of VFAT2 threshold, collecting $10,000$ triggers for each threshold bin. A LabVIEW tool was programmed for this purpose.

The detector was totally shielded by a kapton copper-clad foil. A ground plane was spread close to the hybrids in order to achieve a noise level compatible with the expected signals.

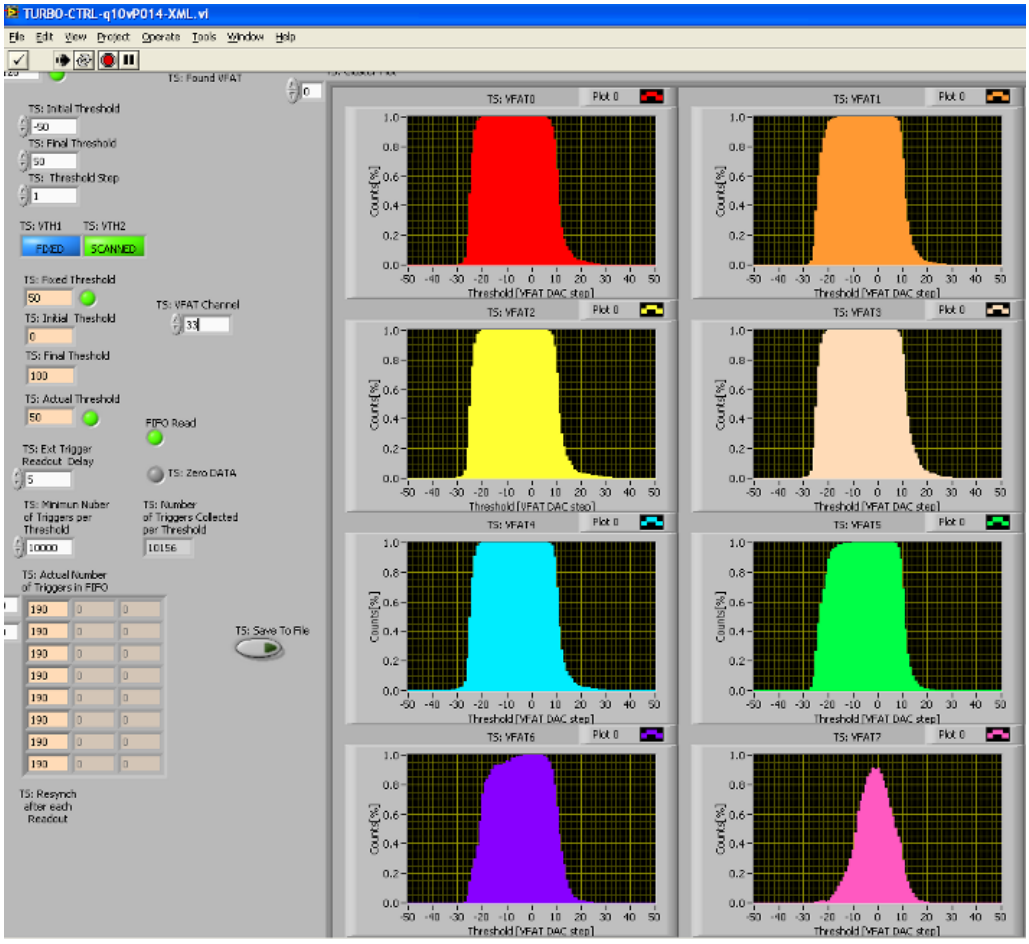


Figure 2.4: An off-beam threshold scan for VFAT2 channels

Threshold scans were performed for all channels of the VFAT2 chips connected to the large GEM detector. We found some disconnected channels and some noisy ones; however, the ground layer enclosing the chips generally reduced the noise fairly well. Indeed most of the channels showed noise levels similar to those visible in Figure 2.4 on the previous page. This means that a threshold $th \geq 35$ *VFAT2 DAC step bins* should cut off all of the noise. Yet, subsequent on-beam results still had low noise levels at threshold $40ds$. The reason for few noise hits still showing up at that threshold level during the test beam, but not in the lab tests, might be that, during the test beam, the high voltage was turned on.

A more thorough analysis of the noise level was done by acquiring **S-Curve**s, using the internal calibration pulse of the chips: see more on this in Chapter 5.1.2 on page 61.

Chapter 3

Experimental setup

3.1 The test-beam facility

CERN provides researchers with **test beam** facilities to test detectors before letting them down into the LHC cavern. Our trial period was scheduled between 12^{th} and 22^{nd} August 2010, and it took place at the **H4** beam line in the CERN site of Prévessin.

Bunches of protons are accelerated in the SPS, a circular particle accelerator, up to a momentum of about $450 GeV/c$. In order to feed more than one test beam facility at the same time, the beam extracted from the SPS is then branched into several channels, each of them terminating in a target where the incident protons create secondary particles. In our case, the target (**T2**) produced pions π^-, with a momentum of $150 GeV/c$.

Both hadron and lepton beams are provided, so that all the possible test requests made by users are covered. Pions have a mean lifetime $\tau = (2,6033 \pm 0,0005) * 10^{-8} s$; their primary decay branch (with *branching ratio* $\Gamma_i/\Gamma = (99,98770 \pm 0,00004)\%$ [1]) is leptonic:

$$\pi^- \longrightarrow \mu^- + \overline{\nu}_\mu$$

H4 beam consists of both pions and their decay products, muons. It is possible to switch to a pure lepton beam by closing the beam collimators: at this energy muons are minimum ionizing particles (**MIP**s) and pass through the collimators, while pions are stopped.

We tested the large prototype GEM detector on:

- a $0,8kHz$ μ^- beam;
- a $38kHz$ π^- beam;
- several π^- beams at intermediate intensities (in order to check if it is possible to experience charging up effects).

3.2 The telescope

The test-beam experimental setup was conceived so to make use of the RD51-GDD[1] tracker. The tracker is made of:

- 3 scintillators, used as a trigger;
- 3 $10 \cdot 10cm^2$ GEM detectors, used as a track detecting system;
- a metallic frame to support the tracker together with the detector prototypes that were to be tested.

The GEM chambers of the tracker feature a strip-based readout plane, made of two layers (one for x and one for y direction). The strip *pitch* is $391\mu m$, which represents the distance between the axes of two adjacent strips. The distribution of gaps between the hit channels of different tracker chambers is Gaussian; its sigma defines the spacial resolution of the tracker:

$$\Delta x = \frac{\sigma_{(G1-G2)}}{\sqrt{2}} = 0.0915mm$$

where $G1$ and $G2$ stand for the position of hit clusters in the first and in the second GEM chamber, respectively.

[1] Abbreviations stand for the CERN *Research and Development* group no. *51*, and the *Gas Detector Development* group.

Our measurements gave a resolution of $0.0915mm$, compatible within the error with that expected by such strip pitch ($\frac{pitch}{\sqrt{12}} = 0.113mm$).

All the cables, the dividers and the cards were also fixed to the metallic frame, which was aligned so that the detectors were perpendicular to the beam line. Figure 3.1 on the next page shows the final setup. The signals from the scintillators were sent to three comparators connected to an `AND` port, the output signal of which was used as **trigger**. The trigger signal was sent to the $turbo_0$ card, which acted as master and forwarded that signal to itself (into another input pin) and to the slave card $turbo_1$. The two *turbo* cards controlled and received inputs from the *VFAT2* chips (both those on the tracking GEM detectors and those on the LG prototype).

3.3 Data analysis system

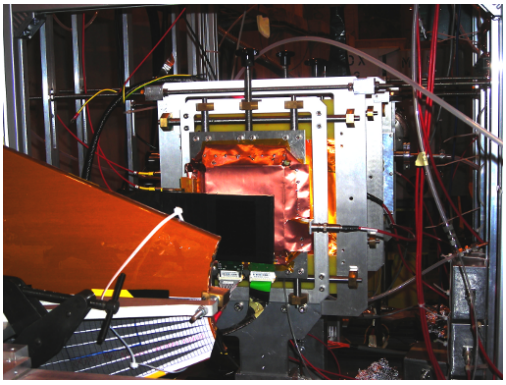
3.3.1 Hits, clusters and tracks

A **hit** on a detector occurs when one of its readout channels collects enough charge to exceed the threshold set in the readout chip. This happens when an electron cloud crosses the last GEM foil. This occurs usually after a particle has started an avalanche in the multiplication volume, but noise and *cross-talk* between adjacent readout channels can also produce a moderate number of hits.

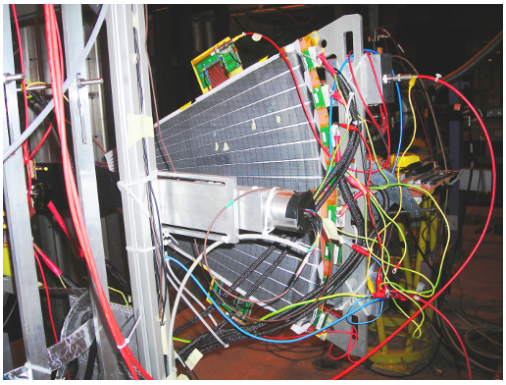
We define **cluster** a set of adjacent hit channels along the x or y axis. A one-channel wide gap is allowed within the set.

During the test-beam, we needed to select a sbset of the data we collected, in order to speed up the analysis process. Therefore we reconstructed the tracks of the incident particles only if the following conditions were both satisfied:

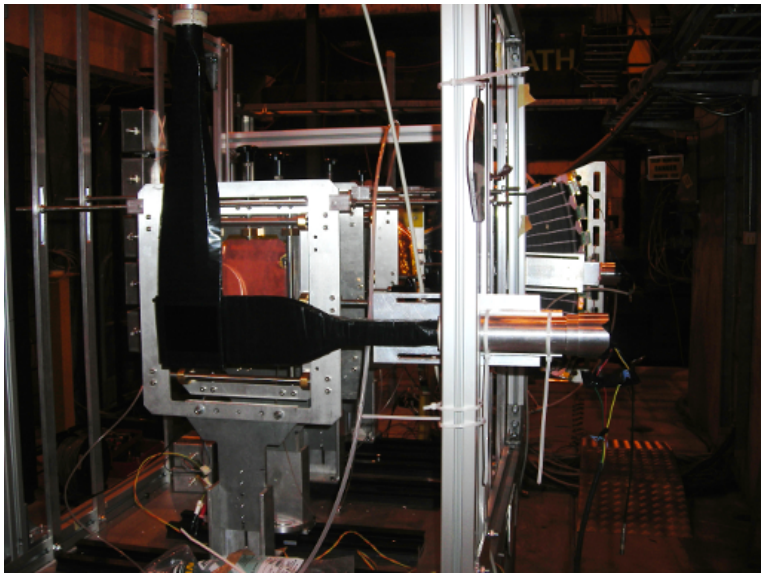
- all the chambers of the tracker presented exactly one hit cluster along



(a) Tracker system setup



(b) Large GEM prototype setup



(c) View of the telescope

Figure 3.1: The test-beam experimental setup: a telescope made of scintillators and of small GEM tracking chambers, and the Large GEM prototype

the x axis, and one along the y axis;

- the hit number in each chamber of the tracker (that is the *cluster size*, since we requested a single cluster) was ≤ 120.

The track reconstructing algorithm exploits ROOT's class `TGraphErrors`. The algorithm structure is:

1. the previously measured distance along the z axis between the tracker chambers is used as a constant;
2. the x position of clusters is plotted versus z in a `TGraphErrors` object;
3. the plot is fitted with a first order polynomial via the ROOT `Fit()` function.

This way, the fit function itself and its parameters (χ^2, residuals, q and m) become available i the $x-$track object itself. The same procedure is used to define $y-$track objects.

If we define $\mathrm{n_{act}}$ as the actual number of hits collected by the LG, and $\mathrm{n_{exp}}$ as the number of expected hits, then

$$\frac{n_{act}}{n_{exp}}$$

represents the Large GEM's **efficiency**. Every time the distance between the position $\mathbf{r}_{HIT}$ of the hit on the LG and the *projection* $\mathbf{r}_{TRK}$ on the LG of the corresponding track is minor than a given efficiency radius $effrad$, we say that the chamber has been **efficient** (and thus we increment n_{act} by 1):

$$|\mathbf{r}_{HIT} - \mathbf{r}_{TRK}| \leq effrad \quad \Longrightarrow \quad n_{act} = n_{act} + 1$$

3.3.2 Reconstructing the beam profile

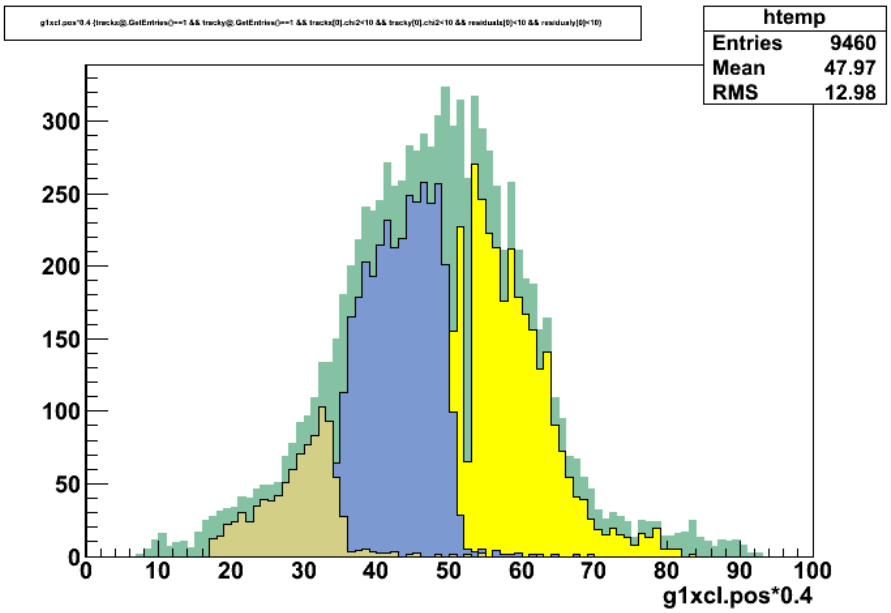
First of all, we checked the prototype mapping and the track reconstruction algorithm used for data analisys. To achieve this, we reconstructed the beam profile, which was known, using the data acquired by tracker and LG. Figure 3.2(a) on page 37 shows our first attempt, which proceeded as follows:

1. we defined a subset of the data, selecting the entries with $\chi^2 < 10$ for every track, in both x and y directions, and low residuals for the hits positions on the traker chambers ($\Delta x < 10 channels$ and $\Delta y < 10 channels$);
2. we plotted the position of the x clusters detected by the first tracker chamber (with respect to a system where axes originate at the lower left corner of the tracker GEM detectors). Since the strip pitch is $\sim 0.4mm$, we compute the position as $0.4 \cdot pos_{CL}$, where pos_{CL} is the position of the cluster expressed as channel number;
3. we plotted the hits on the LG channels, to find the most irradiated ones;
4. we plotted again the distribution of the x clusters detected by the first tracker chamber;
5. finally, we selected only the entries containing a hit on one of the most irradiated LG channels, and we plotted them over the graph produced at the previous step.

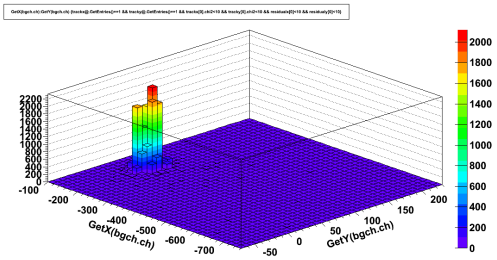
This way we made a sort of puzzle whose pieces represent the beam portion seen by each LG channel. It is like projecting the shadow of the LG pad corresponding to that channel on the reconstruction of the beam profile by the tracker.

Figure 3.2(a) on the next page shows that the adopted LG mapping is correct. Indeed the superposition of signals seen by its pads traces fairly well out the beam profile drawn by the tracker. The different height of the two profiles (the one from the tracker and the one from the LG) is due to the fact that we did not include some of the adjacent LG channels in the plot, which indeed caught a little part of the beam.

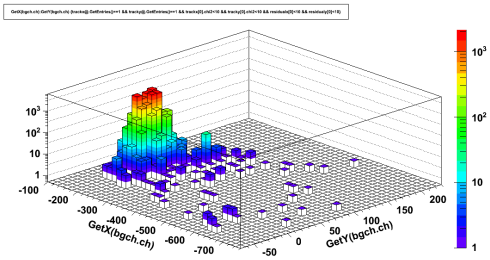
A two-dimensional plot can be done as well. A `GetX()` and a `GetY()` functions were defined to extract x and y pad position information given the number of the corresponding channel. Figure 3.2 on the facing page shows the two-dimensional distribution of the hits detected by the prototype, in



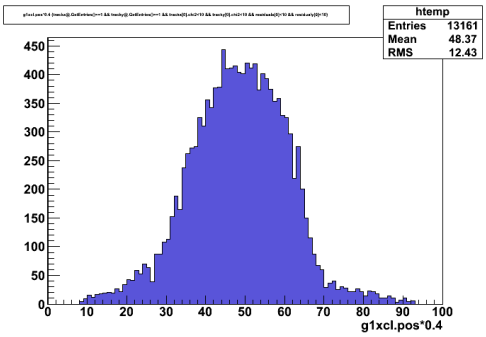
(a) x beam profile by LG channels 699, 700 and 701. The green backround represents the same profile as seen by the tracker.



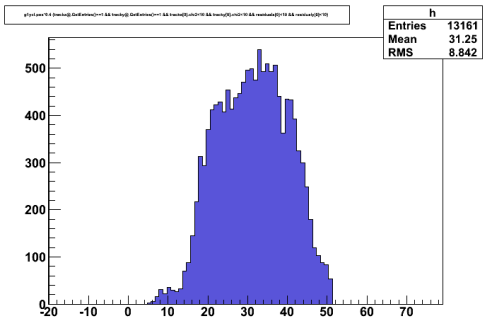
(b) LG beam profile (linear)



(c) LG beam profile (logarithmic)



(d) Tracker beam profile (x)



(e) Tracker beam profile (y)

Figure 3.2: One- and two-dimensional beam profile reconstruction

linear (3.2(b)) and logarithmic (3.2(c)) scales, as well as the tracker profiles for comparison (3.2(d) and 3.2(e)).

The beam shown in Figure 3.2 on the previous page is a muon beam, and thus it is quite large in both directions. Pion beams were more tightly focused, and almost all the charge was concentrated on no more than four adjacent pads.

The beam profile reconstruction algorithm proved to work well, and so did the track reconstruction one.

Figure 3.2(c) on the preceding page gives an idea of the noise affecting the prototype: some channels are hit even though the beam is focused on another area. However, the scale of the plot is logarithmic, and in this case the noise intensity is really negligible with respect of the intensity of the beam.

Chapter 4

On-beam tests

During the test beam we worked at gain values higher than those S. D. Pinto found, presented in Chapter 1.3.1 on page 20. Table 4.1 on the following page shows the extrapolated gain values for our working points (with Ar/CO_2 70/30), using Pinto's exponential fit of his X-Rays gain curve:

$$G = 2 \cdot 10^{-6}\, e^{0.026 \cdot I}$$

The expected current values refer to both the detector sides; actually, the two sides presented slightly different amperage due to small differences between the two divider circuits, in the range of $I_{left} = (I_{right} + \xi)\, \mu A$, where $0 \leq \xi \leq 4.3$.

The gas mixture inside the detector was Ar/CO_2 in 70/30 proportions, where not specified otherwise.

4.1 Efficiency of the tracker

When we started using the test beam, we performed an high voltage scan in order to find the tracker chambers working point. At first we computed its efficiency without checking whether the clusters corresponded to actual

Divider HV (V)	**Expected I** (μA)	**Expected Gain**
$-4,600$	-764.8	$1,930$
$-4,700$	-781.5	$3,021$
$-4,800$	-798.2	$4,767$
$-4,900$	-815.1	$7,403$
$-5,000$	-831.3	$11,684$
$-5,050$	-840.1	$15,100$
$-5,100$	-847.9	$18,439$
$-5,150$	-856.7	$22,947$
$-5,200$	-865.1	$29,737$
$-5,250$	-873.4	$37,206$
$-5,300$	-881.4	$45,599$
$-5,350$	-890.2	$57,104$

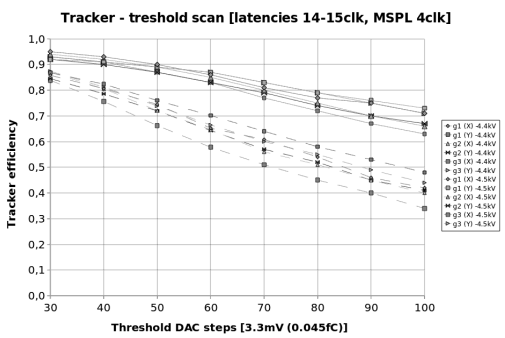
Table 4.1: Extrapolated gain values for $I > 750\mu A$, using Ar/CO_2 70/30

tracks, since we already knew from previous tests that the tracker was reliable.

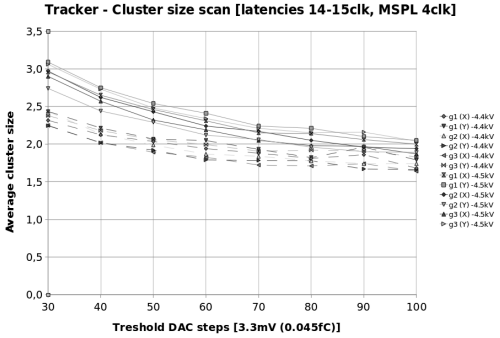
We verified the relationships between efficiency and threshold, and between the average size of hit clusters and the threshold; these measurements were carried out after finding the right latency. Figure 4.1 on the next page shows the results.

4.2 High voltage scan

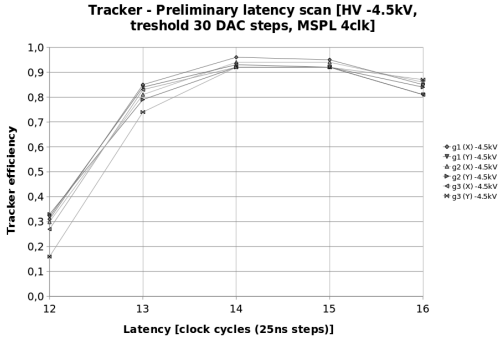
The prototype efficiency was first computed as a function of the High-Voltage (subsequently referred to as **HV**) applied to the divider poles. These scans were performed focussing the muon beam on two regions of the chamber (P and A, whose readout plane areas are covered respectively with large and small pads). We did it in order to check the behaviour of our detector with VFAT2 front-ends in the two extreme pad sizes. Slightly different results were obtained.



(a) Efficiency VS threshold

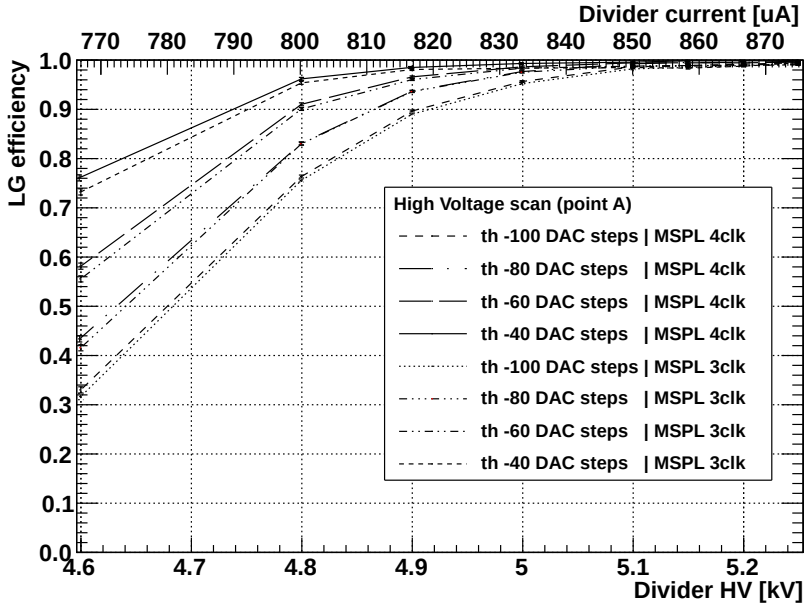


(b) Cluster size VS threshold

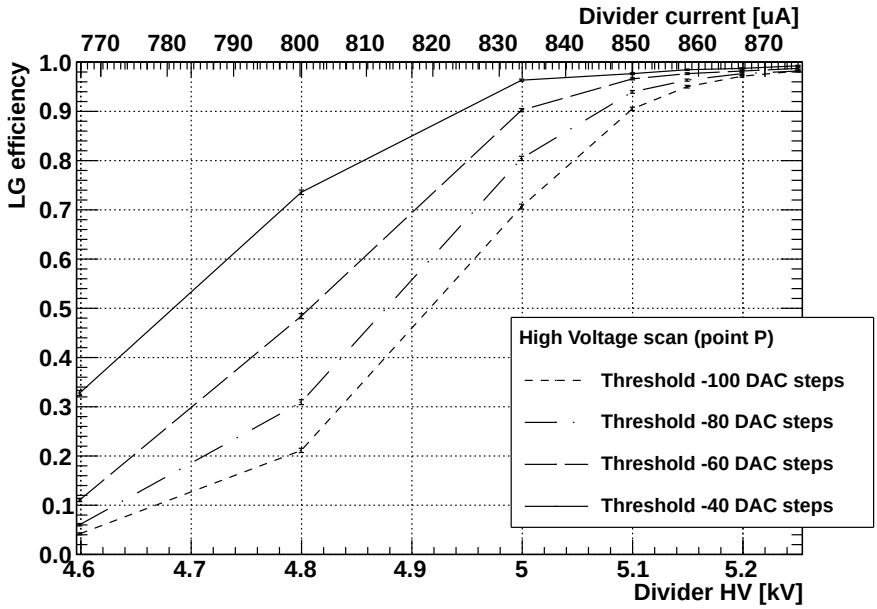


(c) Efficiency VS latency

Figure 4.1: Preliminary tracker tests



(a) *Zone A*: beam on small pads



(b) *Zone P*: beam on large pads ($MSPL = 4clk$)

Figure 4.2: High voltage scans performed with a muon beam focussed on zones A and P

Four different thresholds were applied, in order to have a complete set of data to be used as reference for future comparison: -40 *DAC steps* (subsequently called ds), $-60ds$, $-80ds$ and $-100ds$. The lenght of the monostable pulse (subsequently referred to as **MSPL**) was set at 4 *clock cycles* (clk); in *point A* we also repeated the scan with $MSPL = 3clk$.

Figure 4.2(a) on the facing page shows the results of the HV scan performed when the beam was centered on A, while Figure 4.2(b) shows the results for P.

The two figures show a higher efficiency at lower HV values on the region made of smaller pads. At first we supposed that this effect might be due to the capacitance of the pads themselves, which affects the signal where the pads are large. We studied this phenomenon, but the results (presented in Chapter 5.1.2 on page 61) demonstrated that there should be another reason for this efficiency gap.

In zone A we reached about 95% or higher efficiency (ε) at thresholds $-40ds$ and $-60ds$, already with a divider current $I_D = 817\mu A$. At $I_D = 850\mu A$ we got $\varepsilon \simeq 98\%$ for all the four threshold values. $MSPL = 3clk$ graphs do not diverge significantly from $MSPL = 4clk$ ones. Instead, in zone P the LG prototype approached full efficiency for all thresholds only at $I_D \geq 866\mu A$, with $\varepsilon > 95\%$ at $I_D = 850\mu A$ only for $th = -40ds$ and $th = -60ds$ data sets.

We also ran an HV scan after changing the gas mixture inside the detector, as described in Chapter 4.4 on page 49. We added tetrafluoromethane (CF_4), obtaining a lower gain. Figure 4.3 on the next page shows a comparison of the prototype efficiency with its standard gas mixture (Ar/CO_2 70/30) and with CF_4 ($Ar/CO_2/CF_4$ 60/20/20). I will come back on the purpose of this scan in Chapter 4.4 on page 49. The loss of efficiency at lower current levels may be reduced by optimizing the divider in order to make the detector work with CF_4 with appropriate internal electric fields.

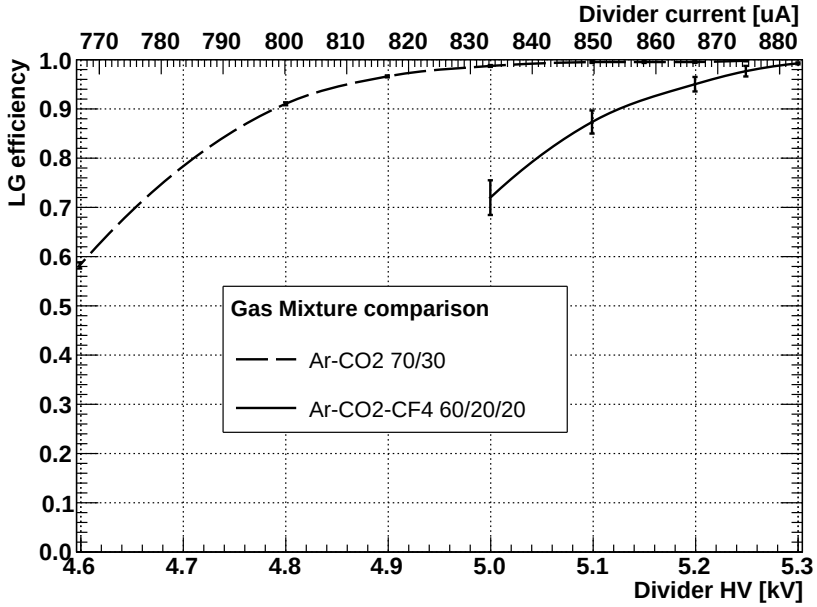
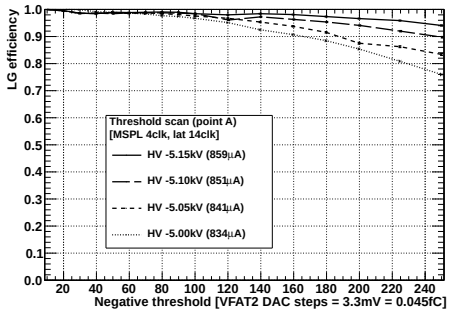


Figure 4.3: High voltage scan performed using an $Ar/CO_2/CF_4$ 60/20/20 gas mixture (same internal voltages and fields as for Ar/CO_2)

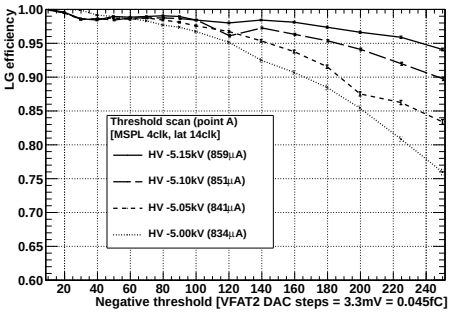
4.3 Threshold scan

We ran an efficiency scan as a function of the VFAT2 threshold level for two different regions of the chamber (A and P). This is particularly useful to understand how much the threshold can be raised (for example, in order to work in noisy environments) while avoiding a significant loss of efficiency. To make the test simpler, we set the same threshold for all the channels at a time, even if their noise levels were quite different. However, a channel by channel adjustment is available, if needed, through VFAT2 *TrimDAC* feature.

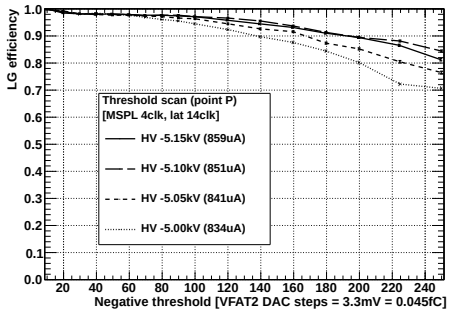
Figure 4.4 on the next page shows the results of the test. The response of the chamber is satisfying: when working at high gain (referring to the sets of data taken at divider current $I \geq 850\mu A$) we can arbitrarily raise the threshold to $th = 90ds$ without virtually experiencing any efficiency loss. However, *zone P* looks again less efficient than *zone A*.



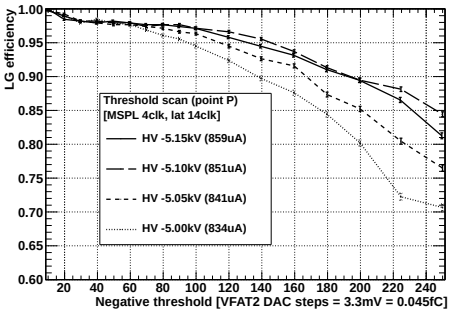
(a) Point A



(b) Point A (zoom)



(c) Point P



(d) Point P (zoom)

Figure 4.4: Threshold scan results for *zone A* and *zone P*

The plots in Figure 4.4 also display an ostensible full efficiency ($\varepsilon \simeq 100\%$) at $th < 30ds$. If we compare this result to those seen in Chapter 2.3 on page 28, it is clear that we are below the noise level: the efficiency seems high, but we are actually detecting more noise hits than particles.

The readout VFAT2 chips we used in our tests are digital: therefore they can only store boolean information about whether a channel is hit or not. The possibility to reconstruct the analog input signal from the digital output data would be extremely useful, because it could allow us to retrieve an estimation of the detector gain. For example, any previous gain curve may be inadequate and charging up effects may occur when a detector is put in front of the LHC beam line, where its working rate can be much higher than that experienced with X-Rays or during a test-beam. It would be needed a way to repeat the gain measurement once the detector is in its final configuration.

We can try to check the actual gain of the chamber using efficiency measurements as a function of threshold. If we had an analog readout electronics, the hit count distribution as a function of charge released in the drift volume would resemble the Landau energy loss distribution (multiplied by the gain of detector), superposed to a Gaussian noise distribution centered in zero.

We assume that:

1. incoming particles are *MIP*s, so that they lose energy almost only by ionization ($\Delta E \propto Q_{IN}$, where Q_{IN} is the charge released in the drift region);
2. the whole charge released by a MIP is collected by a single electrode[1].

Therefore, $\Delta E_{MIP} \propto Q_{IN} \propto Q_{DETECTED}$, where Q_{IN} is the charge released in the drift region. then, the loss of energy of a MIP can be simulated (e.g. with `Garfield`), together with the average ionization potential for the gas mixture in use. This way, we can trace back the initial charge Q_{IN}. In

[1] Electron avalanches diffuse according to a Gaussian law; for $9mm$ of Ar/CO_2 70/30, $\sigma \simeq 300 \div 400\mu m$.

$3mm$ of Ar/CO_2 70/30 we have $\overline{Q_{IN}} \simeq 28$ e^-.

The energy loss distribution, when few interactions cause the whole ΔE, is given by:

$$f(\lambda) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\lambda + e^{-\lambda})} \quad , \qquad \lambda = \frac{\Delta E - \Delta E_{MP}}{K \frac{Z}{A} \frac{\rho}{\beta^2} X}$$

where ΔE_{MP} is the most probable energy loss (the peak of the Landau distribution, see for example Figure 4.5 on the next page) and λ represents the normalized deviation from ΔE_{MP} [14].

We can now fit the efficiency versus threshold histogram we obtained, using the reverse integral of the simulated Landau distribution:

$$F(V_{TH}) = G \cdot \int_{V_{TH}}^{\infty} f(\lambda)\, d\lambda$$

We use G, the gain of the chamber, as a parameter to be found minimizing the χ^2.

By minimizing the χ^2, we can also fix another interesting parameter: the number n_{e^-} of electrons whose total charge equals the amplitude of a threshold bin. Given this value, the conversion from *VFAT2 DAC step bins* to charge, expressed as number of electrons, is allowed. We found that $550 \leq n_{e^-} \leq 1050$ produces a coherent fit, which is indeed reasonable: in principle, for fast signals such as those of GEM detectors, it should be $\Delta Q_{BIN} \simeq 0,045 fC \simeq 280$ e^-, which is of the same order of magnitude.

Figure 4.6 on the following page shows the results of this study. The Landau integral fit covers only a subset of threshold values: an upper bound is needed since the ADC may lose linearity above $n_{steps}^{MAX} - 10\% \simeq 230 ds$[2], while a lower bound allows us to cut the noise Gaussian out of the set of data

[2] $n_{steps}^{MAX} = 256$

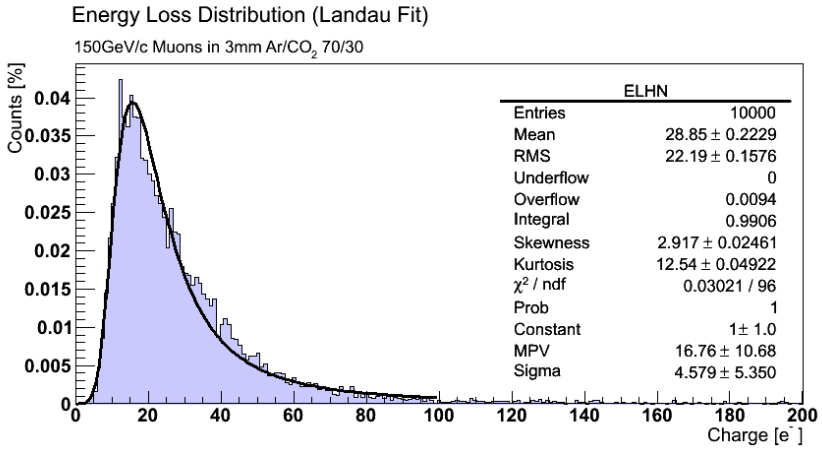


Figure 4.5: Simulation of MIPs energy loss distribution in the detector (*Garfield*)

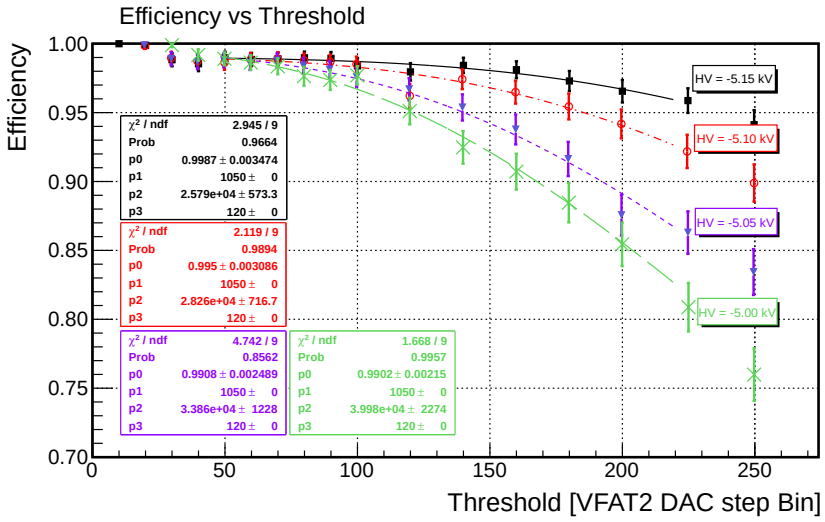


Figure 4.6: Efficiency versus threshold scan fitted by the integral of a Landau energy loss distribution

to be fitted. The lower threshold bound was set at $-50ds$, which was found to be enough to exclude the noise.

This was a very preliminary analysis, which should be refined in order to prove the validity of this method, aimed at checking the gain of proportional chambers in their final setup and at reconstructing analog inputs from digital data.

4.4 Timing scan

We worked on some time-performance scans as well, to check how fast the prototype response was, and what could be done to improve it. First we estimated the latency via the LabVIEW software that we used to interact with the VFAT2 chips, and we found it to be around 17 *clock cycles*. Then we investigated all the latencies in the $[10clk, 19clk]$ interval by means of a set of acquisitions at different thresholds and MSP lengths.

Figure 4.7 on the next page shows all the data sets. Efficiency is plotted versus latency; indeed, the signal starts at $17clk$, and the efficiency reaches 90% only at low thresholds or at $MSPL \geq 3clk$. This is due to the threshold-crossing time spread, that is longer than a single clock cycle when the detector is filled with Ar/CO_2 (it can reach $\sim 60ns$, as pointed out in Chapter 2.1 on page 27, while a clock cycle lasts $25ns$).

We can select two efficient working points:

1. $MSPL = 4clk$, with any threshold value such that $40ds < th < 100ds$;
2. $MSPL = 3clk$, with threshold $40ds \leq th \leq 60ds$.

During these tests, the rate of incoming particles was of the order of magnitude of kHz. In LHC, where the rate is much higher, a monostable pulse lenght $> 1clk$ may cause superposition of events and loss of time resolution. In that case, we would want to stretch the $MSPL$ as little as possible.

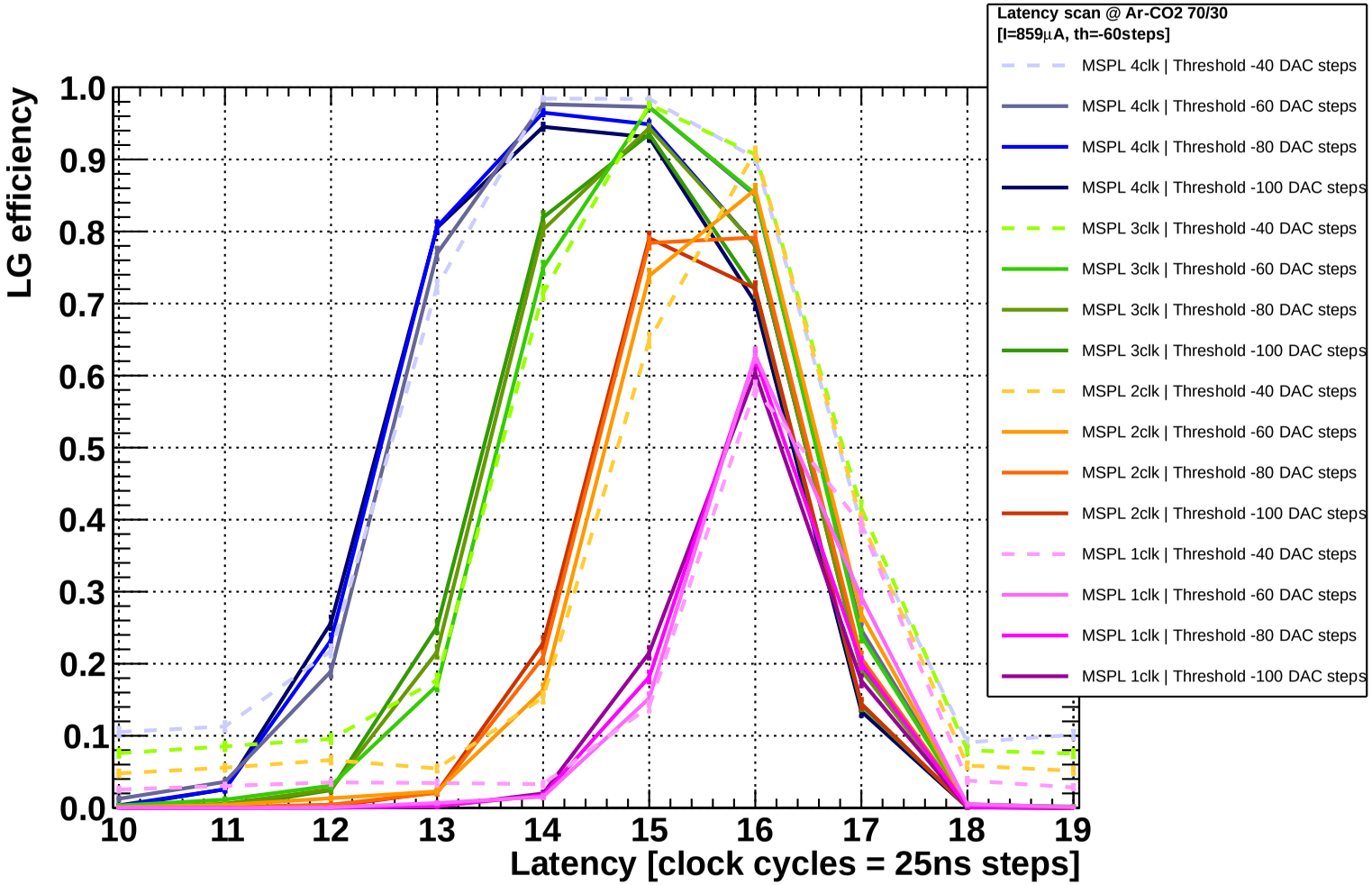


Figure 4.7: Latency scans performed at four different $MSPL$ values. Data sets were taken at various threshold values between $-100ds$ and $-40ds$, during a μ^- beam.

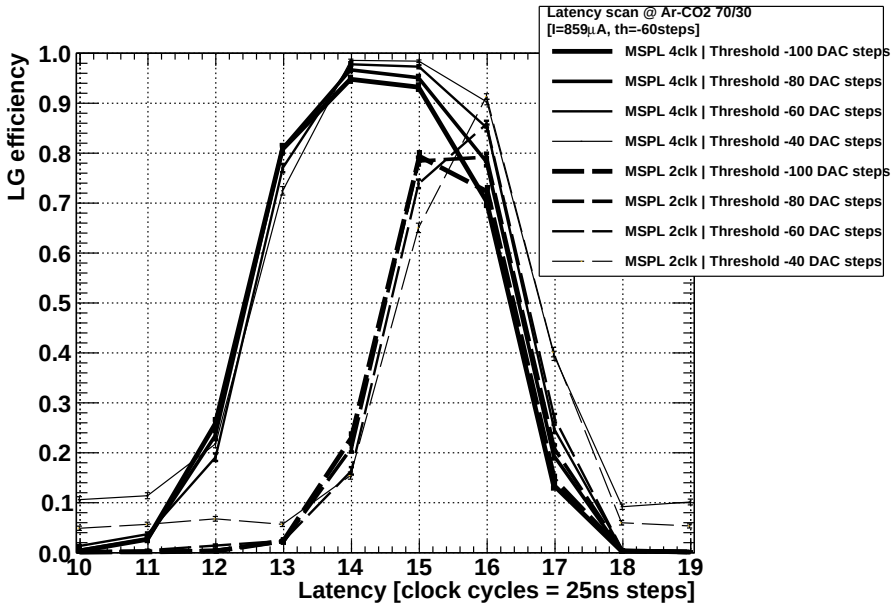


Figure 4.8: Noise counts become visible at $th \leq 40ds$

At the test-beam we made large use of the first working point listed above. We could have used $MSPL = 3clk$, setting $th = 40ds$ in order to reach the same detecting efficiency, which however might have not been enough to cut all of the noise off.

Indeed, we clearly got some noise at $th = -40ds$. The dotted lines in Figure 4.7 show that the chamber was detecting charge even out of the right latency range: that signal was not related to the beam and it was caused by noise. This is confirmed by the fact that the hit count increased as the MSPL lenghtened: by setting a longer MSPL we just integrated the signal (both the "real" and the noisy hits) over a longer time interval.

In Figure 4.8 I explain this behaviour, showing $MSPL = 2clk$ and $MSPL = 4clk$ data sets. For both of them, thicker lines represent high thresholds and thinner lines low thresholds. The number of hits does not fall to 0 outside the latency boundaries; indeed, where for thresholds $th > 40ds$

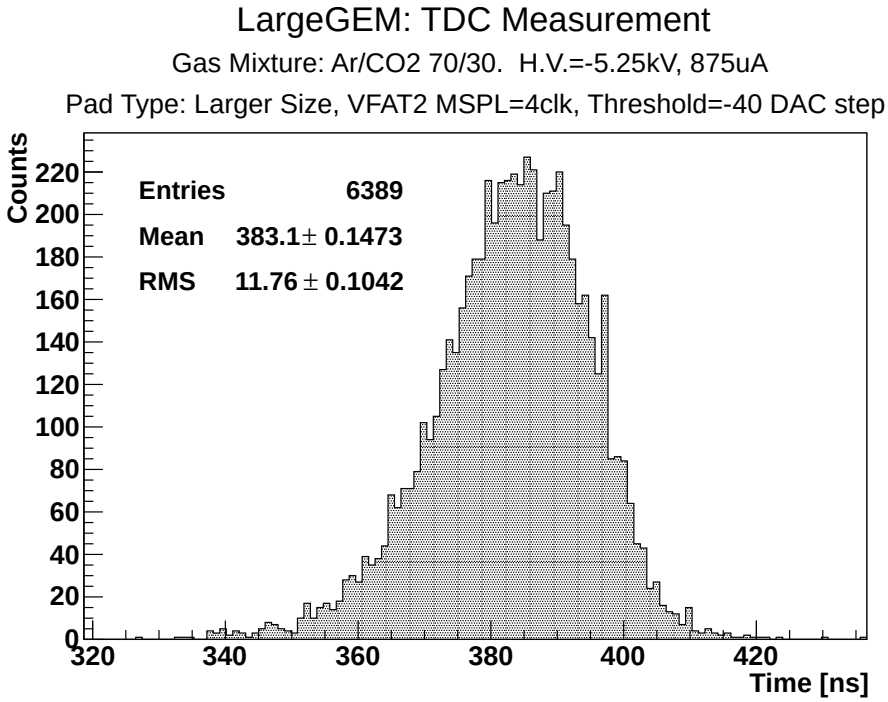


Figure 4.9: Distribution of elapsed time between scintillators and VFAT2 *S-Bit*s

the efficiencies are null, we see that:

$$\mathcal{E}^{th=40ds}_{MSPL=4clk} \simeq 2 * \mathcal{E}^{th=40ds}_{MSPL=2clk} \qquad (lat \leq 11clk \ \vee \ lat \geq 18clk)$$

With $MSPL = 4clk$, noise hits are indeed counted for twice as much time as with $MSPL = 2clk$.

A histogram[3] of the time intervals occurring between the scintillators trigger and the VFAT2 *Fast-OR* signal provides further information about the time resolution of the whole DAQ system[4]. The narrower this distribution is, the higher is the time resolution, which indeed is the *RMS* of the plot in Figure 4.9.

[3]Study performed during the previous test beam period (June 2010).

[4]DAQ stands for Data Acquisition. In this case the DAQ system consists of a large GEM detector with pad readout, and VFAT2 front-end chips.

The same Figure 4.9 also shows that:

$$\frac{< TDC >}{1clk\,(25ns)} = \Delta t_{(SC_{trigger} \rightarrow FastOR_{trigger})} \simeq 15clk$$

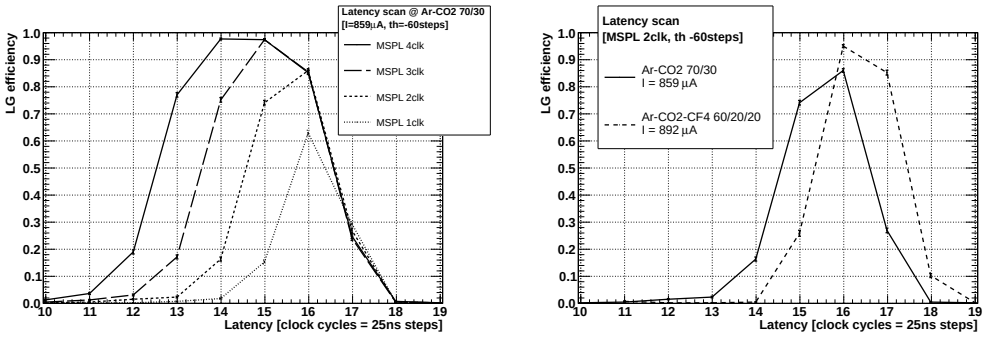
which is only relevant if we are unable to set the right DAQ latency, for which Δt can be used as an upper bound ($lat \leq \Delta t$).

Increasing detectors time performance is a priority. The LHC machine is expected to start working at very high frequency and intensity by the end of 2011, with a bunch crossing every $25ns$ and a luminosity $\mathcal{L} \geq 10^{31} \frac{1}{cm^2 \cdot s}$. TOTEM GEM-based detectors (**T2**) cover a region of very high pseudorapidity[5], and therefore they run through extreme radiation conditions. We ran a test trying to find a new gas mixture for those TOTEM detectors, which would allow for a faster detector response without modifying the divider.

A percentage of CF_4 was added to the gas mixture inside the LG, getting to an $Ar/CO_2/CF_4$ 60/20/20 configuration. After the High-Voltage scan described in Chapter 4.2 on page 40 we repeated some of the latency scans, whose results are compared to those we got with the previous gas mixture in Figure 4.10 on the following page.

We focussed on $MSPL = 2clk$ tests, since a lenght of $2clk$ would be suitable for the future TOTEM runs. Figure 4.10(b) on the next page clearly shows that CF_4 improves the response speed of this kind of detectors: the new gas mixture allows to approach full efficiency at $th = -60ds$ and $MSPL = 2clk$. On the other hand, we needed to supply a little more current on the divider. We did not observe any discharge, but we could run only a few tests with this gas mixture. It must be pointed out that we do not know what the exact gain value of the chamber was when we worked with CF_4. However, previous tests performed by the authors of [7] suggest that

[5]Being θ the angle between the momentum $\overrightarrow{p}$ of the incoming particle and the beam direction, we define **pseudorapidity** $\eta = -\ln\left(\tan\frac{\theta}{2}\right)$. Therefore, as the angle decreases, $\eta \longrightarrow \infty$.



(a) Standard gas mixture results

(b) Gas mixtures comparison at $MSPL = 2clk$

Figure 4.10: Data taken adding CF_4 to the standard Ar/CO_2 70/30 gas mixture (same internal voltages and fields as for Ar/CO_2)

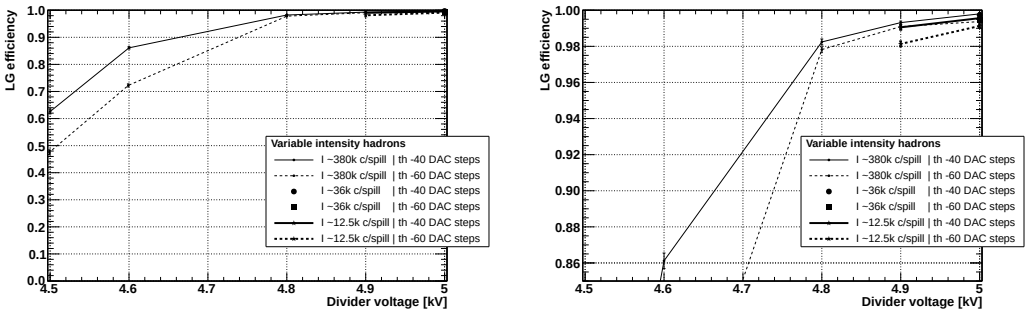
at our working point the gain of $Ar/CO_2/CF_4$ 60/20/20 mixture was even lower, by a factor of five, than that of Ar/CO_2 70/30.

The detector was designed to work with Ar/CO_2 70/30. It is now clear that redesigning its divider to work with this new gas mixture should improve the detector time resolution. In addition, a high efficiency should be reached with a lower gain, which would result in a lower probability of discharge.

4.5 Behaviour with hadron beam

We were given the possibility to test the behaviour of the prototype with hadron (π^-) beams. Particle rates went from $1.25kHz$ up to $38kHz$. Unlike muon beams, pion beams were sharp: the particle flux covered an area of approximately $10 \cdot 5mm^2$.

Figure 4.11(a) on the facing page shows an HV scan performed for different intensities of the π^- beam. However, for some intensity-threshold combinations we only got a few data. The plot shows beam intensities measured in counts per spill; each spill lasted $10s$. Figure 4.12 represents a comparison between pion beam and muon beam HV scans. It only shows data collected at threshold $-60ds$, so to compare two scans with the same settings.



(a) HV scan **(b)** Zoom

Figure 4.11: High-voltage scan under a beam of pions

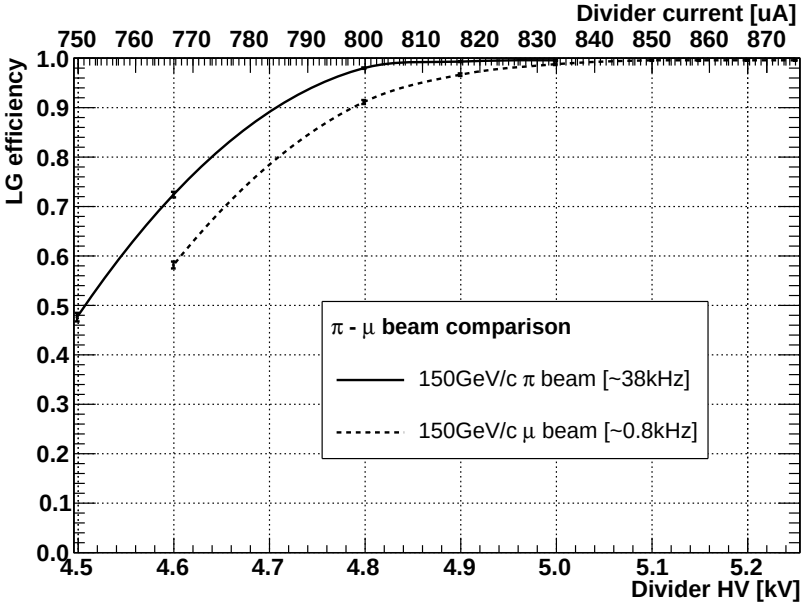


Figure 4.12: Detector behaviour: comparison between pions exposition and muons exposition

We may infer from Figure 4.12 that the prototype presents higher efficiency when detecting hadrons. However, due to the large rate difference between muon beam ($0.8kHz$) and hadron beam (Figure 4.12 plots the set of data collected when the beam intensity was $38kHz$), we can not state clearly whether the increase in efficiency is caused by the presence of hadrons. An effect of **charging up** may have indeed occured in this case: as a result of the higher interaction rate inside the detector, some of the electrons produced in the avalanche may have accumulated on the insulator surface by the GEM holes (which are not perfectly straight). As a consequnce, the electric field inside the holes would have been strenghtened, raising the GEM foil gain with no need of increasing the external voltage. The visible effect of this kind of *charging up* should be a raise of efficiency at constant divider current, as we may recognize in Figure 4.12 on the previous page. Figure 4.11(b) on the preceding page focusses on the slightly different behaviours of the detector when working at different rates. To verify this, we shall perform further laboratory tests, with different intensities X-Rays, or maybe organize future test-beam sessions aimed at solving this question.

Chapter 5

Remarks

5.1 (In)homogeneity of the prototype

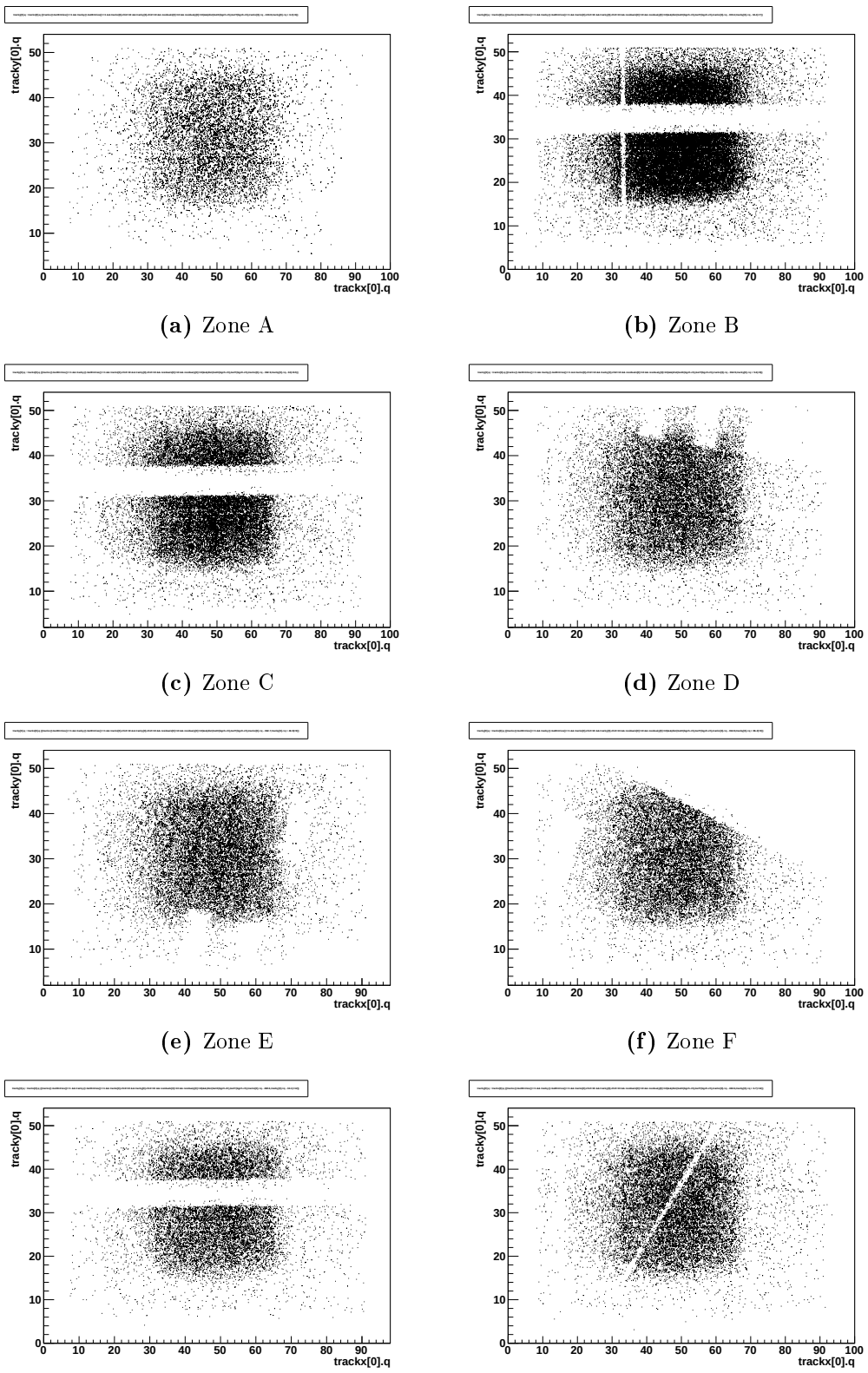
5.1.1 Critical chamber zones

The availability of such a good tracking system allowed us to highlight some spatial defects of the prototype. When we direct a flux of charged particles over a detector, for some reasons the response may vary according to which region of the chamber is irradiated. For instance, it happened that a pad was disconnected from the corresponding pin on its VFAT2 chip; or that we focused the beam over a region of the detector containing a piece of the spacer frame.

Figures 5.1 on the next page and 5.2 on page 59 display a "radiography" of the LG prototype, made by moving the detector around in order to check the response of several areas (see Figure 1.8(b) on page 19). The graphs plot the two-dimensional beam profile as it was detected by the tracker, with the condition that for every track there were corresponding hits on the LG channels.

We spotted:

- *dead* (disconnected) pads: see Figures 5.1(d), 5.1(e) and 5.2(c);



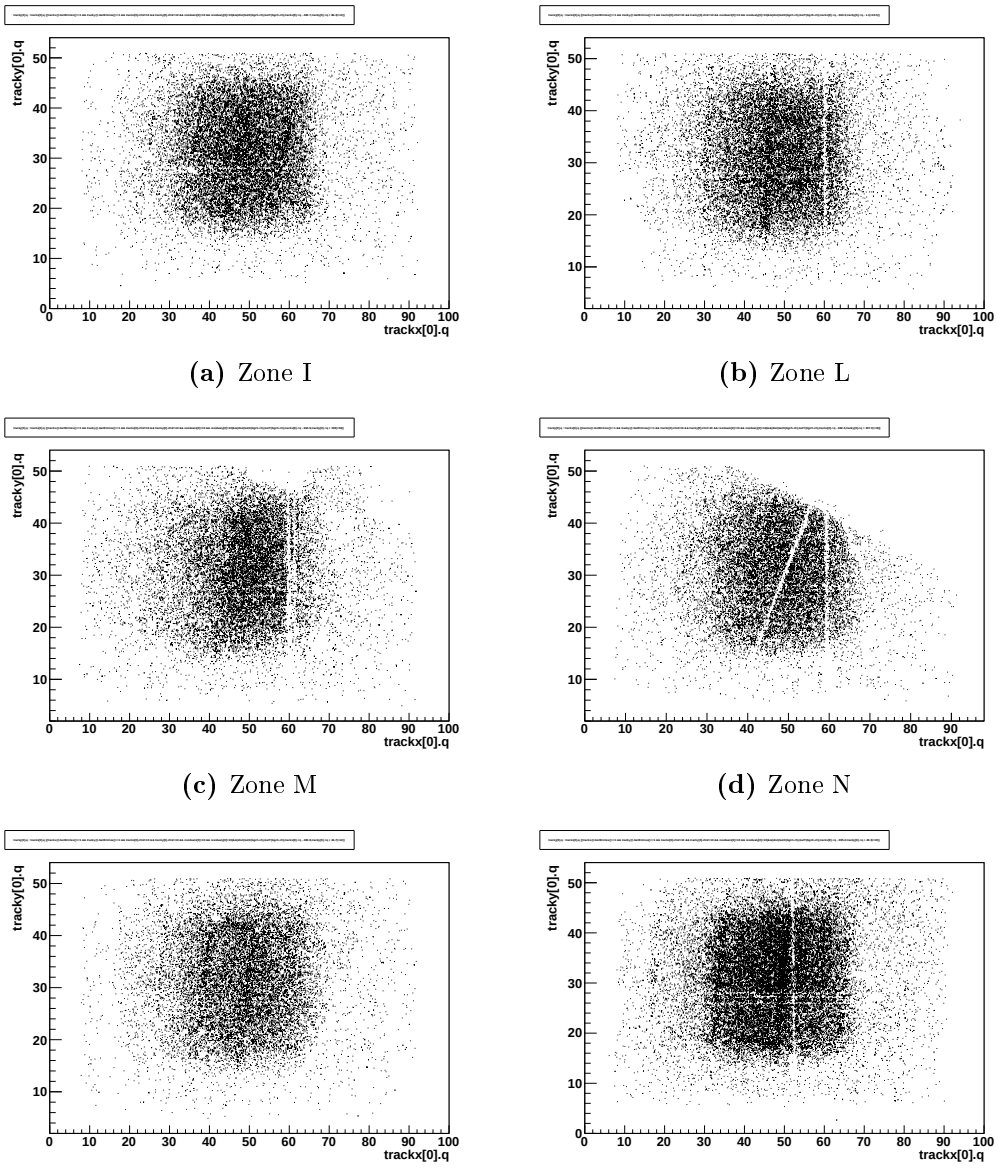
(a) Zone A **(b)** Zone B

(c) Zone C **(d)** Zone D

(e) Zone E **(f)** Zone F

(g) Zone G **(h)** Zone H

Figure 5.1: A radiography of the prototype triple GEM detector



(a) Zone I

(b) Zone L

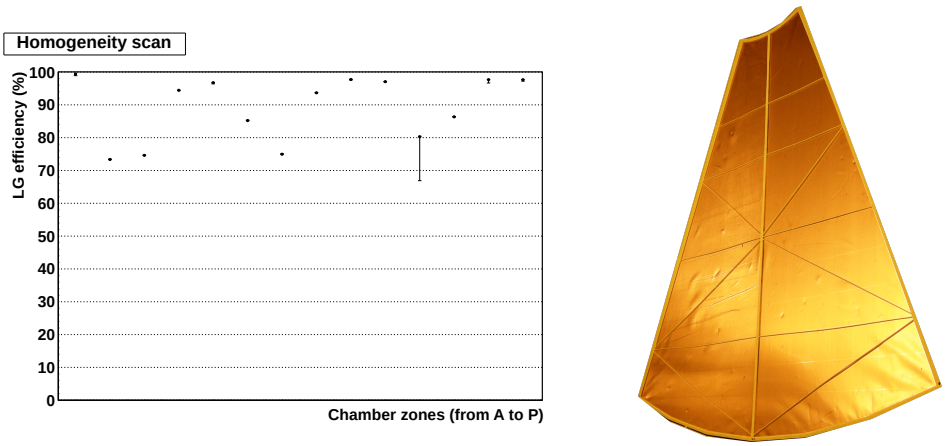
(c) Zone M

(d) Zone N

(e) Zone O

(f) Zone P

Figure 5.2: A radiography of the prototype triple GEM detector



(a) LG efficiency

(b) Spacer frame view

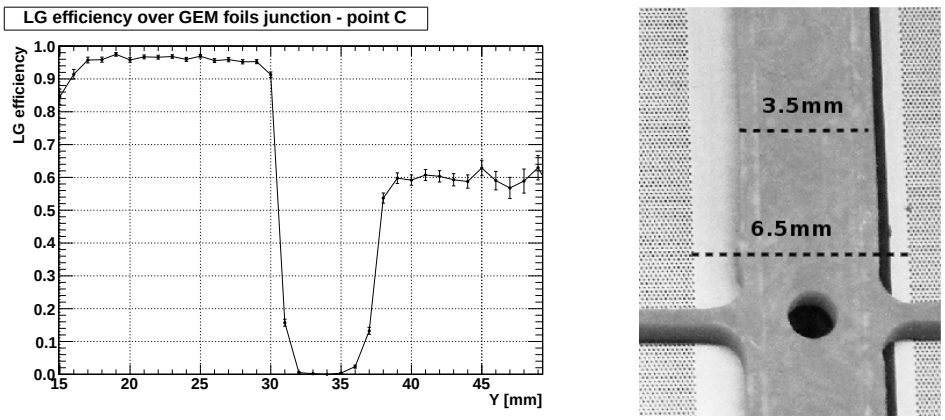
Figure 5.3: Efficiency scan over various critical chamber regions. On the right, a broken GEM layer and its spacer frame (same model as those inserted in the prototype under test)

- thin spacers: see Figures 5.1(b), 5.1(h), 5.2(b), 5.2(c), 5.2(d) and 5.2(f). In particular, Figure 5.2(c) shows a misalignment between a spacer and the edge of a cathode sector;
- segments of the thick central spacer covering the seam between the GEM foils: see Figures 5.1(b), 5.1(c) and 5.1(g);
- the edge of the chamber: see Figures 5.1(f) and 5.2(d).

Figure 5.3(a) and Table 5.1 on page 68 show the computed average efficiency for all these regions, and some more information that will be discussed below.

A deeper study of the response of the detector in the junction area (Figure 5.4(a) on the facing page) shows the steepness of the slope in the curve of efficiency as a function of y, which in that plot varies perpendicularly to the spacer that covers the seam. We can see that the low-efficiency area around the spacer is narrow, as required. No collateral malfunctionings were observed in spacer, border and junction areas.

However, Figure 5.4(a) on the next page also shows an unexpected prob-



(a) Efficiency versus y

(b) Spacer and inactive area width

Figure 5.4: Loss of efficiency at the GEM foils junction

lem, which we became aware of only long after the end of the test beam: the side of the detector which was not directly involved in the tests appears to be less efficient. This effect may be too high to be explained in terms of lack of *charging-up*, and it is visible in all the three junction zones we analized. It is not due to visible asymmetries between the two voltage distribution boards either, whose input impedances were both measured to be equal to $5.40 M\Omega$. The set of data we gathered so far is not enough to allow a proper explanation of this phenomenon.

As already said, we came across this problem after the test beam period finished, and thus we were not able to set up another irradiation test to investigate it. A new gain curve is needed and will be worked out as soon as possible with Cu X-Rays, to compare it with the absolute gain calibration made by S. D. Pinto [13].

5.1.2 Effects of the different pad dimensions

It is expected that larger copper pads have bigger capacitance. Comparing the two Figures 4.2(a) and 4.2(b) on page 42, we may conjecture that the loss of efficiency in *zone P* is due to the bigger pads capacitance, which could

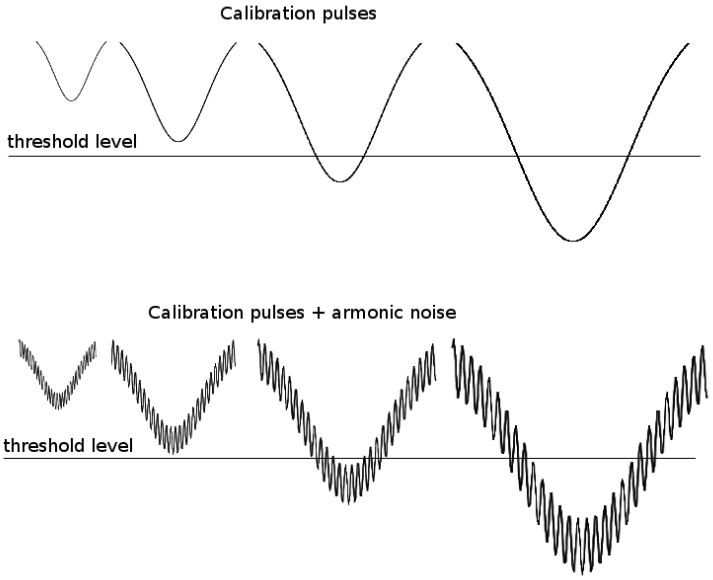


Figure 5.5: How a calibration pulse scan works

cause noise and channel coupling.

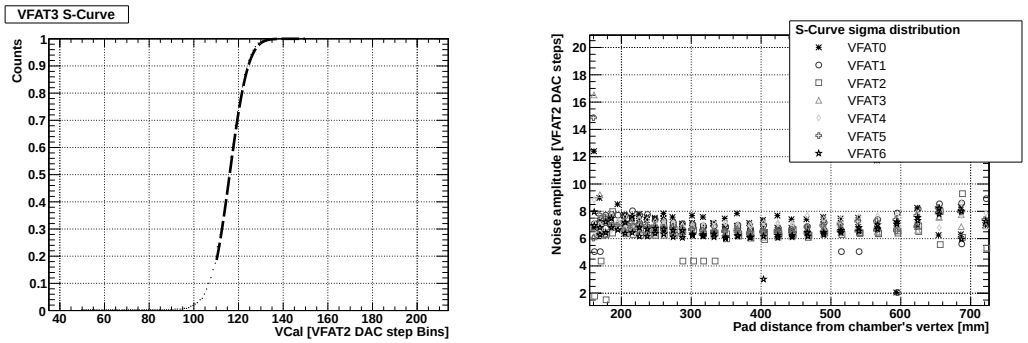
Being noise proportional to the electrode capacitance [6], we decided to run a noise measurement for each channel. This way we could collect information about the capacitance of each readout pad.

An off-beam analysis was thus performed, injecting calibration charge pulses of increasing amplitude to all of the pads via the VFAT2 chips, and fitting the *S-Curve*s of each channel with an `erf` function.

The `erf` function, also known as **Gaussian error function**, is defined as:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

After we define a threshold for the VFAT2 chips, and we start injecting charge, as long as the injected potential ($Q = CV$) is under threshold ($\Delta V < th$) the output of the comparator is low. Then, like an **heaviside step function**, it should go high as soon as the pulse amplitude overcomes



(a) S-Curve **(b)** S-Curve *sigma* vs pad largeness

Figure 5.6: An S-Curve fitted by an `erf` function. On the right, the standard deviation (*sigma*) of the pad's S-Curve is plotted as a function of the radial position of the pad in the detector, and thus as a function of its increasing largeness.

the threshold. Actually, if we take the contribution of the noise to the shapes of the calibration pulses into account (see Figure 5.5 on the preceding page), we can only predict that the output of the comparator goes high at a certain input charge value with a Gaussian probability, centered around the given threshold value.

An **S-Curve** is a histogram of hits for a given threshold, counted while varying the injected charge [5]; it corresponds to the shape of the `erf` function. Indeed, the same plot can be obtained by integrating the Gaussian distribution of the noise for any VFAT2 channel. Once an S-Curve is fitted with an `erf` function, the noise distribution *sigma* is the same as that of the `erf` function, and the mean value corresponds to the threshold potential[1].

Figure 5.6(b) allows us to state that the lack of efficiency of *zone P* is not caused by parasitic capacitance of its pads: the larger pads (those at the right side of the plot) do not show a significant increase of noise.

[1]ROOT stores the fit functions as objects, and makes their parameters (mean, RMS, *sigma*, and so on) available via calls to the objects themselves. To compute the *sigma*s of Figure 5.6(b), we fitted the S-Curve of each channel with `erf` functions and plotted their *sigma* parameters.

According to VFAT2 specifications in [6], each channel should have

$$Q_{noise}^{EQ} = 400\mathrm{e}^- + 50\frac{\mathrm{e}^-}{pF}$$

where Q_{noise}^{EQ} is the equivalent charge due to the noise, and the picofarads refer to the channel input capacitance. Assuming $300 \div 400$ electrons per VFAT2 DAC step bin, with an S-Curve sigma $\sigma \simeq 7$ for all channels we can estimate an average input capacitance of about $48pF$.

Another reason for the difference in response between *point P* and *point A* is to be found. More tests will be performed as soon as possible.

5.2 Efficiency radius lenght: noise checks

In Chapter 3.3.1 on page 33 I explained how we compute efficiency. If a hit occurs in the Large GEM within a given radius (*efficiency radius*) from the projection of the track of the particle, then we say that the chamber has been efficient. If there are no hits within the efficiency radius, the chamber has been inefficient.

What happens if we set a wrong efficiency radius? There are two scenarios:

1. the radius is too short. In a large-pads area we may not include a whole pad within the radius, and the efficiency-computing algorithm might miss some hits.
2. the radius is too long. Some noise hits on adjacent pads may be mistaken as efficient hits.

Figure 5.7 on the next page shows, for every zone we tested, the fluctuation of the computed efficiency for increasing efficiency radius values. It is clear then that $20mm \leq effrad \leq 30mm$ would be appropriate for every purpose.

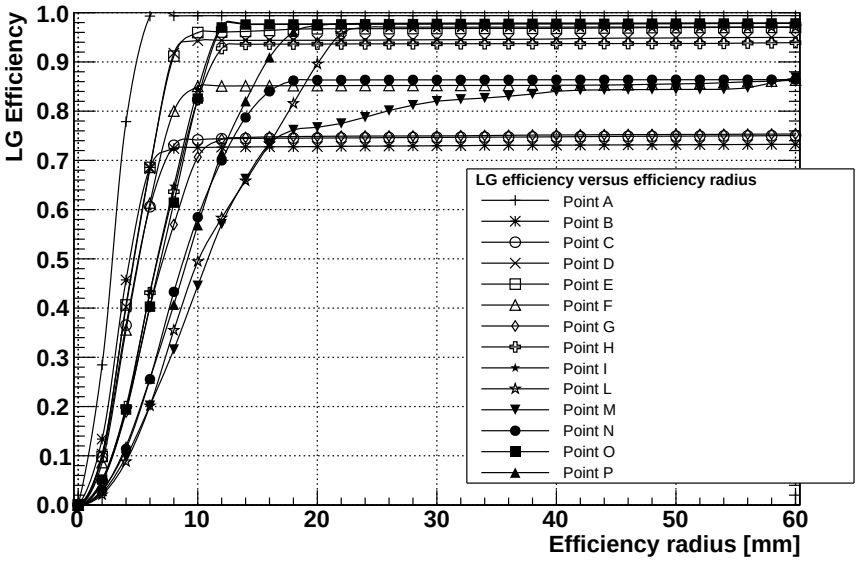
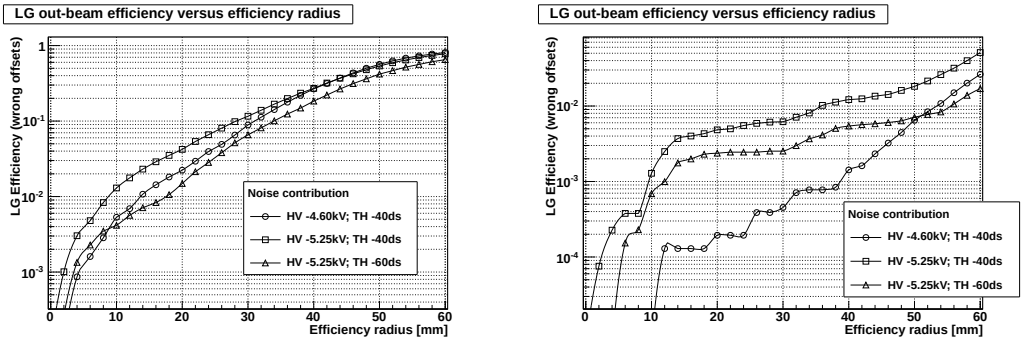


Figure 5.7: Efficiency radius scan: how changing the radius influences the efficiency-computing algorithm. Large GEM working point: $HV = -5.15kV$, $th = -40ds$

Actually, all the results discussed here were computed trying to minimize the efficiency radius for each zone of the chamber, in order to include as few noise hits as possible. For each zone, the efficiency radius was set to the minimum length at which the efficiency stops rising.

The plot of Figure 5.7 is also satisfying because it shows that the noise level is low for all zones, with the exception of *point M*: the *plateau*s are very flat, which means that stretching the radius does not mean to include much more noise. *Zone M* was probably close to two noisy channels, one at a distance of about $20mm$ and the other, less noisy, about $35mm$ far, as it can be inferred from the plot.

Playing with the efficiency radius may provide further precise results. For example, we can take a couple of the *HV scan* data sets shown in Figure 4.2(a) on page 42, and arbitrarily set wrong *offsets* (see the following section for a deeper explanation). This means that we are looking for hits in a part of the chamber which was not irradiated. In that part, a scan like that of Figure 5.7



(a) Zone A **(b)** Zone P

Figure 5.8: Off-beam efficiency computation: evaluating the contribution of noise

will show the contribution of pure noise to the computation of the detector efficiency. Stretching the radius, we indeed allow the software to look for hit channels in a larger region of the chamber, therefore we include more noise. Table 5.1 on page 68 shows, for each scanned zone:

- the best efficiency radius, according to Figure 5.7;
- the level of efficiency of that chamber sector, computed with the chosen best efficiency radius;
- the contribution of noise to the computation of efficiency, at the same efficiency radius. The noise values are plotted like in Figure 5.8[2], which show the efficiency of a non-irradiated sector as a function of the efficiency radius.

The errors in the efficiency column are statistical:

$$\Delta\varepsilon = \frac{\varepsilon(1-\varepsilon)}{\sqrt{n}}$$

where n is the number of events collected and ε is the computed efficiency. These errors must be added to those of the following column.

[2]Figure 5.8(a) shows that *zone A* is close to some noisy channels. Indeed, with an efficiency radius of $60mm$ the efficiency grows to significant values; however, $effrad = 6mm$ excludes virtually all noise from the scans.

Table 5.1 therefore shows how noise must be computed to determine which errors affect the efficiency values. If we made this scan on-beam, these effects would be negligible, because the charge released by incoming particles would dominate.

5.3 Analysis algorithms: cuts

Chapter 3.3.1 on page 33 points out that tracks are reconstructed only in some simple cases. In addition, during the analysis process we only used the tracks satisfying the following conditions:

1. there is exactly one *x-track* and one *y-track* per event;
2. $\chi^2 < 10$ for both the *x-track* and the *y-track*;
3. the distance between the hits on the tracker chambers and their first order polinomial fit defining the track is $< 10mm$ for every hit.

Inside a ROOT framework, this means to declare a `TCut` object that will be used as a mandatory option while selecting the data to process:

```
TCut goodtr("goodtr","trackx@.GetEntries()==1 && tracky@.GetEntries()==1 && trackx[0].
    chi2<10 && tracky[0].chi2<10 && residualx[0]<10 && residualy[0]<10")
```

Most of the analysis processes exploit the ROOT `Draw()` command, that accepts `TCut` objects as options. For the data sets we have seen so far, whose acquisition needed the chamber to be shifted 14 times, we had to declare the (x, y) offsets of the chamber with respect to the center of the beam each time. We achieved that again via `TCut` objects, which also contained a definition of efficiency radius. The following declarations were used:

```
TCut Aeff("Aeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 240.8,tracky[0]->q +
    6.3)<6")
TCut Beff("Beff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 303.2,tracky[0]->q -
    34.4)<7")
TCut Ceff("Ceff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 362.9,tracky[0]->q -
    34)<8.5")
TCut Deff("Deff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 362.9,tracky[0]->q +
    5.6)<9")
TCut Eeff("Eeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 362.7,tracky[0]->q +
    46.9)<9")
TCut Feff("Feff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 362.8,tracky[0]->q +
    86.2)<9")
```

Zone	**Best *effrad***	**Efficiency**	**Contribution of noise**
A	$6mm$	$(99.40 \pm 0)\%$	0.5%
B	$7mm$	$(73.35 \pm 0.11)\%$	0.01%
C	$8.5mm$	$(74.60 \pm 0.11)\%$	0.01%
D	$9mm$	$(94.43 \pm 0.03)\%$	0.07%
E	$9mm$	$(96.76 \pm 0.02)\%$	0.35%
F	$9mm$	$(85.21 \pm 0.10)\%$	0.03%
G	$12mm$	$(74.95 \pm 0.14)\%$	0.02%
H	$12mm$	$(93.68 \pm 0.04)\%$	0.02%
I	$12mm$	$(97.73 \pm 0.02)\%$	0.13%
L	$23.5mm$	$(97.07 \pm 0.02)\%$	0.04%
M	$30mm$	$(80.28 \pm 0.12)\%$	13.38%
N	$18mm$	$(86.35 \pm 0.09)\%$	0.05%
O	$12mm$	$(97.67 \pm 0.02)\%$	1.01%
P	$18mm$	$(97.66 \pm 0.03)\%$	0.43%

Table 5.1: Contribution of noise to the computation of LG efficiency. Efficiency was detected at: $HV = -5,15kV$, $th = -60ds$, $MSPL = 4clk$

```
TCut Geff("Geff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 482.6,tracky[0]->q -
    34.1)<12")
TCut Heff("Heff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 482.6,tracky[0]->q +
    6.7)<12")
TCut Ieff("Ieff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 482.7,tracky[0]->q +
    86.3)<12")
TCut Leff("Leff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 663.4,tracky[0]->q -
    1.1)<23.5")
TCut Meff("Meff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 664.9,tracky[0]->q +
    108)<30")
TCut Neff("Neff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 662.5,tracky[0]->q +
    207.3)<18")
TCut Oeff("Oeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 483.8,tracky[0]->q +
    46.7)<12")
TCut Peff("Peff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 695.2,tracky[0]->q +
    49.3)<18")
```

where the first two numbers in each string represent the (x, y) position of the chamber for the specific data set, and the last one is the best fitting efficiency radius.

This approach worked fine. The software did not run across any problems when variables were declared this way, and selecting tracks according to strict rules did not affect the analysis process, as the amount of data acquired during the test-beam period was huge.

Chapter 6

Conclusions

6.1 Quality of the large GEM prototype

The performances of the detector were within the expectations. We did not encounter any problems during the test beam DAQ sessions: the prototype resisted and detected both muon and pion beams of various intensities.

As expected, the use of CF_4 improves the time response of the detector, even if its fields and internal structure[1] are left unchanged. In order to get even better performances, a revision is needed of the detector design (including its high voltage divider). This might be achieved by joining our efforts with the CMS experiment and the group of A. Sharma, whose recent studies were in the same direction as ours.

According to the manufacturers, the difficulties encountered during the assembly do not justify the advantage of splicing GEM foils together in order to cover larger areas and thus to reduce the number of detectors. For example, there are currently no proper machines for stretching large foils with an irregular shape.

As we will discuss in Chapter 6.2, we needed to test if the detector, with its

[1] Most of all, the height of the internal gaps.

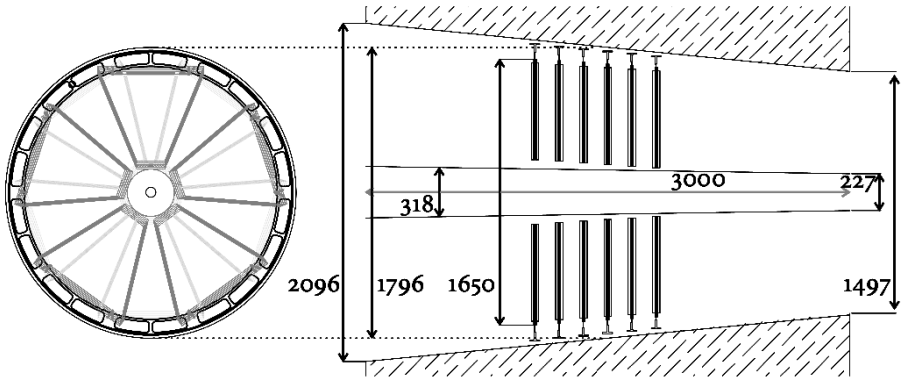


Figure 6.1: An arrangement to replace the current **T1** CSC telescope of TOTEM [12]

large capacitance readout electrodes, could work efficiently with the TOTEM readout system. The initial intense noise was removed by improving the chip grounding on the detector. Additional VFAT2 features, such as *TrimDAC*, may be used in order to improve the global SNR ratio.

As a final remark, it should be mentioned that these deep investigations were made possible thanks to the RD51-GDD tracker telescope, which could largely improve the resolution and comprehension of the data we collected.

6.2 Large GEMs for TOTEM and CMS

The GDD group, and ours, conceived the *"Large GEM"* as a replacement for the current **T1** forward telescope of the TOTEM experiment at the LHC, which is now made of Cathode Strip Chamber detectors. T1 consists of two arms, each of them precisely fitting in a gap around the beam pipe and inside the inner surface of the CMS detectors, in symmetric positions around *Interaction Point 5*. It covers the pseudorapidity region $3,1 \leq |\eta| \leq 4,7$.

A renovation is needed for T1, as its CSC chambers may start ageing fast when the LHC machine will start to run at luminosity $\mathcal{L} \geq 10^{31} \frac{1}{cm^2 \cdot s}$. In particular, it is expected that if luminosity overcomes that level by two orders of magnitude, the CSCs would age in a few months [10]. However, according

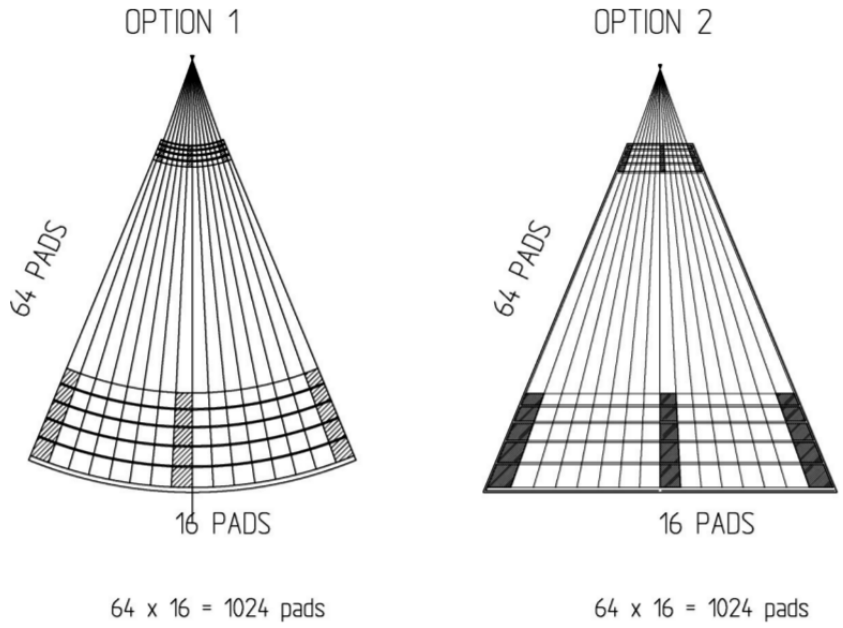


Figure 6.2: Two readout board options for a large triple GEM detector

to informal reports of ongoing T1 data acquisition sessions, discharges are already being observed at a luminosity $\mathcal{L} \simeq 10^{30} \frac{1}{cm^2 \cdot s}$.

The same structure can be built with large GEM chambers, arranging six discs for each arm, where a disc is made of two planes of 5 chambers in a back-to-back arrangement (Figure 6.1 on the preceding page). Overlap regions would ensure a 360° coverage and allow to adjust the radius of the six discs on demand [13].

As an alternative [10], each disc may be made of six detectors on a single plane; they could be alternated in back-to-back configuration, each disc offset by 30° from the adjacent ones, to allow overlay. Four such sets would be enough to equip an arm of the future T1.

As in Figure 6.2, readout boards would be pad based, like in the prototype, featuring $16 \cdot 64 = 1024$ pads of various dimensions.

With respect to CSC, GEM chambers offer high rate tolerance, limited discharge probability (less than 10^{-12} at gain $G \simeq 10^4$), high time resolution. For the prototype analized in this thesis, $RMS \simeq 11.8ns$ (see Figure 4.9 on page 52), and it may be improved by adding CF_4.

In conclusion, T1 had been designed for a bunch crossing rate $f \geq 75ns$ and luminosity $\mathcal{L} \leq 10^{31}$, while in principle a GEM based substitute could survive for years at $\mathcal{L} = 10^{33}$ and would be suitable for faster bunch crossing by just adding a percentage of CF_4 to the internal gases [10].

The CMS experiment is also developing similar large GEM prototypes, with a compatible readout, foreseeing a project for muon detecting in the forward region. $50cm$ wide GEM foils are being produced with no need for splicing.

At the same time, the INFN section in Bari is working on LASER ablation of GEM foils, a technique that could make automation of mass-manufacturing easier, and possibly increase hole precision and density.

A new generation of GEM detectors has definitely been started. The large active surface and quite simple readout system of these new detectors will prove useful to better reveal high energy particles both from accelerators and cosmic rays. They might also bear significant improvements in different research fields, such as low-energy or medical physics, and path a way to possible new applications.

Appendix A

ROOT analysis routines

Two young researchers[1] from **RD51** built the low-level macros used to convert the binary output of the detectors to much more physicist-friendly ROOT `n-tuple`s.

My personal work consisted in writing down some routines in order to extract efficiency values from specific *n-tuple*s, and make the data accessible via plots. The following pages collect the most significant pieces of code I wrote. In order to meet different data sets, raw data file names and break conditions should be modified.

A.1 Data reconstructing algorithms

Listing A.1: Builder.C

```
#include <string>
#include <iostream>
#include <TTree.h>
#include <TFile.h>


int EventBuilderVFAT(const char* rawfilename,
                     const char* rootfilename,
                     const int readmaxevent);

using namespace std;
```

[1] Matteo Alfonsi (CERN) and Gabriele Croci (PhD student at the University of Siena)

```
void Builder(string rawfile_address){

  size_t spos;
  size_t substring_lenght;
  string slash = "/";
  spos = rawfile_address.rfind(slash);
  substring_lenght = 7;

  // Extracts the string "Run####" from the rawfile name
  string rawfile_name = rawfile_address.substr(spos+5,substring_lenght);
  // Sets the output file name
  string rootfile_address = "../RootData/" + rawfile_name + ".root";
  // Builds a .root file from a rawfile
  EventBuilderVFAT(rawfile_address.c_str(),rootfile_address.c_str(),100000000);
  // Sets the reco file name
  string recofile_address = "../RootData/" + rawfile_name + "_reco.root";
  // Creates a reco file using Offset_Settings.txt
  TFile file0(rootfile_address.c_str());
  TTree* t = dynamic_cast<TTree*>(file0.Get("rd51tb"));
  t->Process("Reco2d_ps.C+",recofile_address.c_str());

}
```

Listing A.2: Recoizer.C

```
#include <string>
#include <iostream>
#include <TTree.h>
#include <TFile.h>

using namespace std;

void Recoizer(string rootfile_name){

  string run_number = rootfile_name.substr(15,4);

  // Sets the reco file name
  string recofile_name = "../RootData/Run" + run_number + "_reco.root";

  // Creates a reco file using Offset_Settings.txt
  TFile file0(rootfile_name.c_str());
  TTree* t = dynamic_cast<TTree*>(file0.Get("rd51tb"));
  t->Process("Reco2d_ps.C+",recofile_name.c_str());

}
```

Listing A.3: effex2.C

```
#include "TFile.h"
#include "TTree.h"
#include "TH1.h"
#include <iostream>

using std::cout;

double effex2(TFile& file0){

  TH1F* heffbgch = dynamic_cast<TH1F*>(file0.Get("heffbgch"));
  double efficiency;
  efficiency = 1 - heffbgch->GetBinContent(1) / heffbgch->GetEntries();
  return efficiency;

}
```

Listing A.4: efferrors.cc

```
#include <string>
#include <iostream>
#include <sstream>
#include <TTree.h>
#include <TFile.h>
#include <cstdio>
#include <TH1.h>
#include <TVectorT.h>

using namespace std;

double effex2(TFile& file0);

// Gives a fast print of efficiency and
// corresponding statistical errors
void efferrors(long int run_number, long int last_run){

  Int_t size = last_run - run_number + 1;

  TVectorD nRun (size);
  TVectorD run_eff (size);
  TVectorD errors (size);
  Int_t count = 0;
  long int j = run_number;

  for (long int i=j;i<last_run+1;i++)
    {
      string s;
      char run_n[3];
      sprintf(run_n,"%ld",i);
      s = string(run_n);
      string rootfile_name = "../RootData/Run0" + s + ".root";
      string recofile_name = "../RootData/Run0" + s + "_reco.root";

      TFile file0(recofile_name.c_str());
      double eff = effex2(file0);
      run_eff[count] = eff;
      nRun[count] = i;
      TH1F* heffbgch = dynamic_cast<TH1F*>(file0.Get("heffbgch"));
      double n = heffbgch->GetEntries();
      double staterror = eff * (1 - eff) / sqrt( n );
      errors[count] = staterror;
      count = count + 1;
      file0.Close();
    }

}
```

Listing A.5: GetOffsets.cpp

```
// Given user-measured offsets, this
// computes and prints the exact offsets
// (the coordinates of the irradiated zones)
{
          TCut goodtr("goodtr","trackx@.GetEntries()==1 && tracky@.GetEntries()==1 &&
     trackx[0].chi2<10 && tracky[0].chi2<10 && residualx[0]<10 && residualy[0]<10");

// Uncomment the desired zone
// where to compute offsets
          TCut Aeff("Aeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 263,tracky
     [0]->q + 13.5)<30");
//        TCut Beff("Beff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 323,tracky
     [0]->q - 26.5)<30");
//        TCut Ceff("Ceff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 383,tracky
     [0]->q - 26.5)<30");
```

```
//          TCut Deff("Deff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 383,tracky
    [0]->q + 13.5)<30");
//          TCut Eeff("Eeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 383,tracky
    [0]->q + 53.5)<30");
//          TCut Feff("Feff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 383,tracky
    [0]->q + 93.5)<30");
//          TCut Geff("Geff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 503,tracky
    [0]->q - 26.5)<30");
//          TCut Heff("Heff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 503,tracky
    [0]->q + 13.5)<30");
//          TCut Ieff("Ieff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 503,tracky
    [0]->q + 93.5)<30");
//          TCut Leff("Leff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 683,tracky
    [0]->q - 13.5)<30");
//          TCut Meff("Meff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 683,tracky
    [0]->q + 113.5)<30");
//          TCut Neff("Neff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 683,tracky
    [0]->q + 213.5)<30");
//          TCut Oeff("Oeff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 503,tracky
    [0]->q + 53.5)<30");
//          TCut Peff("Peff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 715,tracky
    [0]->q + 53.5)<30");

// Replace "Aeff" with the uncommented TCut
          rd51tb->Draw("GetY(bgch.ch)",goodtr && Aeff);
          double bgY = htemp->GetMean();
          rd51tb->Draw("tracky[0].q",goodtr && Aeff);
          double trkY = htemp->GetMean();
          double offsety = bgY - trkY;
          rd51tb->Draw("GetX(bgch.ch)",goodtr && Aeff);
          double bgX = htemp->GetMean();
          rd51tb->Draw("trackx[0].q",goodtr && Aeff);
          double trkX = htemp->GetMean();
          double offsetx = bgX - trkX;
          std::cout << "offsetX; offsetY" << std::endl << offsetx << "; " << offsety <<
    std::endl;
}
```

Listing A.6: Macro-CalibrationScanAndFit.C

```
#include "TROOT.h"
#include "Riostream.h"
#include "TF1.h"
#include "TH1.h"
#include "TMath.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TGraph.h"
#include "TCanvas.h"


void CalPulseAnalizer(TString inputfile="CPScan.dat", TString outputfile="CPscan.root",
    const Int_t nlines=0, Int_t vfatnumber=0)
{

        // Preparing evironment and new rootfile
        gROOT->Reset();
        TFile *f = new TFile(outputfile,"RECREATE");

        // Reading the input data file
        ifstream in;
        in.open(inputfile);

        // Creating ntuple...
        Double_t VCal[nlines],VFAT0Counts[nlines],VFAT1Counts[nlines],VFAT2Counts[nlines
    ],VFAT3Counts[nlines],VFAT4Counts[nlines],VFAT5Counts[nlines],VFAT6Counts[nlines],
```

```
VFAT7Counts[nlines];
   TNtuple *CalPulse = new TNtuple("CalPulse","data from ascii file","VCal:
VFAT0Counts:VFAT1Counts:VFAT2Counts:VFAT3Counts:VFAT4Counts:VFAT5Counts:VFAT6Counts:
VFAT7Counts");

   // Initializing ranges
   Double_t MaxRangeVFAT0 = 255;
   Double_t MaxRangeVFAT1 = 255;
   Double_t MaxRangeVFAT2 = 255;
   Double_t MaxRangeVFAT3 = 255;
   Double_t MaxRangeVFAT4 = 255;
   Double_t MaxRangeVFAT5 = 255;
   Double_t MaxRangeVFAT6 = 255;
   Double_t MaxRangeVFAT7 = 255;

   Double_t MinRangeVFAT0 = 0;
   Double_t MinRangeVFAT1 = 0;
   Double_t MinRangeVFAT2 = 0;
   Double_t MinRangeVFAT3 = 0;
   Double_t MinRangeVFAT4 = 0;
   Double_t MinRangeVFAT5 = 0;
   Double_t MinRangeVFAT6 = 0;
   Double_t MinRangeVFAT7 = 0;

   // Setting flags
   Bool_t  MaxRange0=0;
   Bool_t  MaxRange1=0;
   Bool_t  MaxRange2=0;
   Bool_t  MaxRange3=0;
   Bool_t  MaxRange4=0;
   Bool_t  MaxRange5=0;
   Bool_t  MaxRange6=0;
   Bool_t  MaxRange7=0;

   Bool_t  MinRange0=0;
   Bool_t  MinRange1=0;
   Bool_t  MinRange2=0;
   Bool_t  MinRange3=0;
   Bool_t  MinRange4=0;
   Bool_t  MinRange5=0;
   Bool_t  MinRange6=0;
   Bool_t  MinRange7=0;

   Double_t MinPercent=0.15;

   Int_t Steps=0;

   for (Int_t i = 0; i < nlines; i++)
   {
           in >> VCal[i] >> VFAT0Counts[i] >> VFAT1Counts[i] >> VFAT2Counts[i] >>
VFAT3Counts[i] >> VFAT4Counts[i] >> VFAT5Counts[i] >> VFAT6Counts[i] >> VFAT7Counts[
i];

           // Setting the range for the fit around the mean value
           if (!MinRange0 && VFAT0Counts[i]>=MinPercent)
           {
                   MinRangeVFAT0 = VCal[i];
                   MinRange0 = 1;
           }
           if (!MinRange1 && VFAT1Counts[i]>=MinPercent)
           {
                   MinRangeVFAT1 = VCal[i];
                   MinRange1 = 1;
           }
           if (!MinRange2 && VFAT2Counts[i]>=MinPercent)
           {
                   MinRangeVFAT2 = VCal[i];
```

```
        MinRange2 = 1;
}
if (!MinRange3 && VFAT3Counts[i]>=MinPercent)
{
        MinRangeVFAT3 = VCal[i];
        MinRange3 = 1;
}
if (!MinRange4 && VFAT4Counts[i]>=MinPercent)
{
        MinRangeVFAT4 = VCal[i];
        MinRange4 = 1;
}
if (!MinRange5 && VFAT5Counts[i]>=MinPercent)
{
        MinRangeVFAT5 = VCal[i];
        MinRange5 = 1;
}
if (!MinRange6 && VFAT6Counts[i]>=MinPercent)
{
        MinRangeVFAT6 = VCal[i];
        MinRange6 = 1;
}
if (!MinRange7 && VFAT7Counts[i]>=MinPercent)
{
        MinRangeVFAT7 = VCal[i];
        MinRange7 = 1;
}


if (!MaxRange0 && VFAT0Counts[i]>=0.95)
{
        MaxRangeVFAT0 = VCal[i];
        MaxRange0 = 1;
}
if (!MaxRange1 && VFAT1Counts[i]>=0.95)
{
        MaxRangeVFAT1 = VCal[i];
        MaxRange1 = 1;
}
if (!MaxRange2 && VFAT2Counts[i]>=0.95)
{
        MaxRangeVFAT2 = VCal[i];
        MaxRange2 = 1;
}
if (!MaxRange3 && VFAT3Counts[i]>=0.95)
{
        MaxRangeVFAT3 = VCal[i];
        MaxRange3 = 1;
}
if (!MaxRange4 && VFAT4Counts[i]>=0.95)
{
        MaxRangeVFAT4 = VCal[i];
        MaxRange4 = 1;
}
if (!MaxRange5 && VFAT5Counts[i]>=0.95)
{
        MaxRangeVFAT5 = VCal[i];
        MaxRange5 = 1;
}
if (!MaxRange6 && VFAT6Counts[i]>=0.95)
{
        MaxRangeVFAT6 = VCal[i];
        MaxRange6 = 1;
}
if (!MaxRange7 && VFAT7Counts[i]>=0.95)
{
        MaxRangeVFAT7 = VCal[i];
```

```
                MaxRange7 = 1;
            }


            if (!in.good()) break;

            // Filling ntuple
            CalPulse->Fill(VCal[i],VFAT0Counts[i],VFAT1Counts[i],VFAT2Counts[i],
VFAT3Counts[i],VFAT4Counts[i],VFAT5Counts[i],VFAT6Counts[i],VFAT7Counts[i]);
            Steps = i;
    }

    // Uncomment to print number of VCal steps found
    //      printf(" found %d points\n",Steps);
    in.close();


    TCanvas *c1 = new TCanvas ("c1","S-Curve Fit");

    // Plotting VFAT0 counts VS VCal steps
    TGraph *gr0 = new TGraph(Steps,VCal,VFAT0Counts);
    gr0->SetTitle("VFAT0 S-Curve");
    gr0->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr0->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT1 counts VS VCal steps
    TGraph *gr1 = new TGraph(Steps,VCal,VFAT1Counts);
    gr1->SetTitle("VFAT1 S-Curve");
    gr1->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr1->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT2 counts VS VCal steps
    TGraph *gr2 = new TGraph(Steps,VCal,VFAT2Counts);
    gr2->SetTitle("VFAT2 S-Curve");
    gr2->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr2->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT3 counts VS VCal steps
    TGraph *gr3 = new TGraph(Steps,VCal,VFAT3Counts);
    gr3->SetTitle("VFAT3 S-Curve");
    gr3->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr3->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT4 counts VS VCal steps
    TGraph *gr4 = new TGraph(Steps,VCal,VFAT4Counts);
    gr4->SetTitle("VFAT4 S-Curve");
    gr4->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr4->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT5 counts VS VCal steps
    TGraph *gr5 = new TGraph(Steps,VCal,VFAT5Counts);
    gr5->SetTitle("VFAT5 S-Curve");
    gr5->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr5->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT6 counts VS VCal steps
    TGraph *gr6 = new TGraph(Steps,VCal,VFAT6Counts);
    gr6->SetTitle("VFAT6 S-Curve");
    gr6->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr6->GetYaxis()->SetTitle("Counts");

    // Plotting VFAT7 counts VS VCal steps
    TGraph *gr7 = new TGraph(Steps,VCal,VFAT7Counts);
    gr7->SetTitle("VFAT7 S-Curve");
    gr7->GetXaxis()->SetTitle("VCal [VFAT2 DAC step Bin]");
    gr7->GetYaxis()->SetTitle("Counts");
```

```
    // Defining fit function (= error function)
    TF1 *sigmoid = new TF1("sigmoid", "[0] * (1 + TMath::Erf( (x - [1])
/[2]/1.41421356 )) / 2");
    sigmoid->SetParNames("Amplitude", "Mean", "Sigma");

    // Normalizing the sigmoid
    sigmoid->FixParameter(0, 1);

    // Setting fit parameters' ranges and initial values
    sigmoid->SetParameter(1, 100);
    sigmoid->SetParError(1, 0.1);
    sigmoid->SetParameter(2, 5);
    sigmoid->SetParError(2, 0.01);
    sigmoid->SetParLimits(2, 2, 20);

    // Fitting the specified S-Curve with the sigmoid in a specified range
    switch (vfatnumber)
      {
      case (0):
        // Uncomment a Draw command to check the specified fit
        //      gr0->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT0, MaxRangeVFAT0);
        gr0->Fit(sigmoid,"Q","",MinRangeVFAT0,150);
        gr0->GetFunction("sigmoid")->Write();
        gr0->Write();
        break;
      case (1):
        //      gr1->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT1, MaxRangeVFAT1);
        gr1->Fit(sigmoid,"Q","",MinRangeVFAT1,150);
        gr1->GetFunction("sigmoid")->Write();
        gr1->Write();
        break;
      case (2):
        //      gr2->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT2, MaxRangeVFAT2);
        gr2->Fit(sigmoid,"Q","",MinRangeVFAT2,150);
        gr2->GetFunction("sigmoid")->Write();
        gr2->Write();
        break;
      case (3):
        //      gr3->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT3, MaxRangeVFAT3);
        gr3->Fit(sigmoid,"Q","",MinRangeVFAT3,150);
        gr3->GetFunction("sigmoid")->Write();
        gr3->Write();
        break;
      case (4):
        //      gr4->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT4, MaxRangeVFAT4);
        gr4->Fit(sigmoid,"Q","",MinRangeVFAT4,150);
        gr4->GetFunction("sigmoid")->Write();
        gr4->Write();
        break;
      case (5):
        //      gr5->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT5, MaxRangeVFAT5);
        gr5->Fit(sigmoid,"Q","",MinRangeVFAT5,150);
        gr5->GetFunction("sigmoid")->Write();
        gr5->Write();
        break;
      case (6):
        //      gr6->Draw("AP");
        sigmoid->SetParLimits(1, MinRangeVFAT6, MaxRangeVFAT6);
        gr6->Fit(sigmoid,"Q","",MinRangeVFAT6,150);
        gr6->GetFunction("sigmoid")->Write();
        gr6->Write();
```

```
            break;
          case (7):
            //       gr7->Draw("AP");
            sigmoid->SetParLimits(1, MinRangeVFAT7, MaxRangeVFAT7);
            gr7->Fit(sigmoid,"Q","",MinRangeVFAT7,150);
            gr7->GetFunction("sigmoid")->Write();
            gr7->Write();
            break;
          }

        // Uncomment to print fit parameters found
        //       printf("\n\n Amplitude=%f,\t Mean=%f,\t Sigma=%f\n\n",Amplitude,Mean,
    Sigma);

        // Saves the objects created in the current rootfile

        CalPulse->Write();

        f->Write();
        f->Close();

}
```

A.2 Scan-specific algorithms

Listing A.7: EffRadius.cc

```
#include <stdlib.h>
#include <string>
#include "TVectorT.h"
#include <iostream>
#include <sstream>
#include <TTree.h>
#include <TFile.h>
#include <TH1F.h>
#include <TGraph.h>
#include <TAxis.h>
#include <TCut.h>
#include <stdio.h>
#include <cstdio>
#include "TotemMap.hpp"

using namespace std;

TVectorD EffRadius(long int run_number, double xoffsetmm, double yoffsetmm, long int
    minrad, Int_t nsteps){
          TVectorD rad_eff (nsteps+1);
          string run;                                  //Run number to be converted to string
          char run_n[30];
          string offsetx;                              //Offsets to be converted to strings
          string offsety;

          sprintf(run_n,"%ld",run_number);             //Converting run number to a string
          run = string(run_n);
          {                                            //Converting x offset to string
            ostringstream xx;
            xx << xoffsetmm;
            offsetx = xx.str();
          }
          {                                            //Converting y offset to string
            ostringstream yy;
```

```
            yy << yoffsetmm;
            offsety = yy.str();
          }

          /*
          End converting numbers to strings ****************************************
          */

          string rootfile_name;
          string recofile_name;

      if (run_number < 10)
        {
          rootfile_name = "../RootData/Run000" + run + ".root";
          recofile_name = "../RootData/Run000" + run + "_reco.root";
        }
      else
        {
          if (run_number < 100)
            {
              rootfile_name = "../RootData/Run00" + run + ".root";
              recofile_name = "../RootData/Run00" + run + "_reco.root";
            }
          else
            {
              rootfile_name = "../RootData/Run0" + run + ".root";
              recofile_name = "../RootData/Run0" + run + "_reco.root";
            }
        }

          TFile file0(rootfile_name.c_str());
          TTree* t = dynamic_cast<TTree*>(file0.Get("rd51tb"));
          t->AddFriend("recotree",recofile_name.c_str());

          //Selecting high quality tracks and events
          TCut goodtr("goodtr","trackx@.GetEntries()==1 && tracky@.GetEntries()==1 &&
    trackx[0].chi2<10 && tracky[0].chi2<10 && residualx[0]<10 && residualy[0]<10");
          Int_t count = 0;

          //Scans radius from mirad to maxrad every 5 millimeters
          for (long int i = minrad; i < 2*nsteps + 1; i = i + 2)
            {
              string irad;                          //Converts current radius to string
              ostringstream rad;
              rad << i;
              irad = rad.str();

              string draw_string = "Sum$((dist(GetX(bgch.ch),GetY(bgch.ch),(trackx[0]->q
     - trackx[0]->m*400" + offsetx + "),(tracky[0]->q -tracky[0]->m*400 + " + offsety +
    ")))<" + irad + ") >>h(5,0,5)";

              t->Draw(draw_string.c_str(),goodtr);

              TH1F *h = (TH1F*)gDirectory->Get("h");

              Double_t ieff = 1. - h->GetBinContent(1)/h->GetEntries();

              rad_eff[count] = ieff;
              count++;
            }

          return rad_eff;

}
```

Listing A.8: EffOverJunction.cpp

```
// Plots an efficiency scan over the foils seam
// Change the run number to change data set
{
        #include "TGraphErrors.h"

        TFile f("../RootData/Run0703.root");
        rd51tb->AddFriend("recotree","../RootData/Run0703_reco.root");
        TCut Beff("Beff","dist(GetX(bgch.ch),GetY(bgch.ch),trackx[0]->q - 303.2,tracky
    [0]->q - 34.4)<7");
        TCut goodtr("goodtr","trackx@.GetEntries()==1 && tracky@.GetEntries()==1 &&
    trackx[0].chi2<10 && tracky[0].chi2<10 && residualx[0]<10 && residualy[0]<10");
        rd51tb->Draw("tracky->q >>h2(60,0,60)",goodtr);
        rd51tb->Draw("tracky->q >>h1(60,0,60)",goodtr && Beff);
        TH1F *h2 = (TH1F*)gDirectory->Get("h2");
        TH1F *h1 = (TH1F*)gDirectory->Get("h1");
        Double_t efficiency[60];
        Double_t x[60];
        Double_t errors[60];
        Double_t errorsx[60];
        for (Int_t i = 1; i < 60 + 1; i++)
          {
            if (h2->GetBinContent(i) != 0)
              {
                efficiency[i-1] = h1->GetBinContent(i) / h2->GetBinContent(i);
              }
            else efficiency[i-1] = 0;
            errors[i-1] = (efficiency[i-1] * (1 - efficiency[i-1])) / sqrt(h2->
    GetBinContent(i));
            errorsx[i-1] = 0;
            x[i-1] = i - 1;
          }
        TGraphErrors effgraph(60,x,efficiency,errorsx,errors);
        effgraph.GetYaxis()->SetRangeUser(0,1);
        effgraph.GetYaxis()->SetDecimals();
        effgraph.SetTitle("LG efficiency over GEM foils junction");
        effgraph.GetXaxis()->SetTitle("Y [mm]");
        effgraph.GetYaxis()->SetTitle("LG efficiency");
        effgraph.GetXaxis()->SetRangeUser(15,49);
        effgraph.SetMarkerStyle(7);
        effgraph.Draw("ALP");

}
```

Listing A.9: EffradNoise.C

```
#include <string>
#include <iostream>
#include <sstream>
#include <TTree.h>
#include <TFile.h>
#include <TH1F.h>
#include <TCut.h>
#include <cstdio>
#include "TMultiGraph.h"
#include "TAxis.h"
#include "TGraph.h"
#include "TLegend.h"
#include "TVectorD.h"
#include "TCanvas.h"


TVectorD EffRadius(long int run_number, double xoffsetmm, double yoffsetmm, long int
     minrad, Int_t nsteps);

void EffradNoise(double offsetX, double offsetY){
```

```
// I make a multigraph with all the efficiency VS radius graphs for the points around
   the LG.
// Change offsets to move around the LG
Int_t steps = 30;
long int begin = 0;
TCanvas* c1;
long int run1 = 550;
long int run2 = 380;
long int run3 = 371;

// Declaring efficiency arrays
// 2 data sets should be enough
TVectorD run550 (steps+1); // HV -4.60 th 40
TVectorD run380 (steps+1); // HV -5.25 th 40
TVectorD run371 (steps+1); // HV -5.25 th 60

// Declaring and initializing list of efficiency radiuses
TVectorD rads (steps+1);
for (int i = 0; i<=steps; i++) {rads[i]=2*i;};

// Filling efficiency arrays
run550 = EffRadius(run1,offsetX,offsetY,begin,steps);
run380 = EffRadius(run2,offsetX,offsetY,begin,steps);
run371 = EffRadius(run3,offsetX,offsetY,begin,steps);

TGraph *graph550 = new TGraph (rads,run550);
graph550->SetTitle("HV -4.60kV; TH -40ds");
graph550->SetMarkerColor(kBlack);
graph550->SetMarkerStyle(24);
graph550->SetLineColor(kBlack);
TGraph *graph380 = new TGraph (rads,run380);
graph380->SetTitle("HV -5.25kV; TH -40ds");
graph380->SetMarkerColor(kBlack);
graph380->SetMarkerStyle(25);
graph380->SetLineColor(kBlack);
TGraph *graph371 = new TGraph (rads,run371);
graph371->SetTitle("HV -5.25kV; TH -60ds");
graph371->SetMarkerColor(kBlack);
graph371->SetMarkerStyle(26);
graph371->SetLineColor(kBlack);

TMultiGraph *multigraph = new TMultiGraph();
multigraph->Add(graph550);
multigraph->Add(graph380);
multigraph->Add(graph371);

multigraph->Draw("ACP");

multigraph->GetXaxis()->SetTitle("Efficiency radius [mm]");
multigraph->GetYaxis()->SetTitle("LG Efficiency (wrong offsets)");
multigraph->GetYaxis()->SetDecimals();
multigraph->GetXaxis()->SetRangeUser(0,60);
multigraph->SetTitle("LG out-beam efficiency versus efficiency radius");

TLegend *legend = new TLegend (0.1,0.4,0.4,0.9);
legend->SetHeader("Noise contribution");
legend->AddEntry(graph550,graph550->GetTitle(),"lp");
legend->AddEntry(graph380,graph380->GetTitle(),"lp");
legend->AddEntry(graph371,graph371->GetTitle(),"lp");
legend->Draw();
c1->Update();
c1->SetGridx();
c1->SetGridy();

c1->Update();
```

```
}
```

Listing A.10: HVscanPmacro.C

```
#include <TGraph.h>
#include <TGraphErrors.h>
#include <TCanvas.h>
#include <TAxis.h>
#include <stdio.h>
#include <cstdio>
#include "TVectorT.h"
#include <stdlib.h>
#include <iostream>
#include "TMultiGraph.h"
#include "TAxis.h"
#include <TFile.h>
#include <algorithm>


TVectorD nevents(long int run_number, long int last_run);
TVectorD efficiency(long int run_number, long int last_run);

// HV scan for point p ranges from run 0021 to run 0108
void HVscanPmacro(long int first_run = 21, long int last_run = 108)
{

  Int_t size = last_run - first_run + 1;
  Int_t thsize = 7;

  TVectorD efficiencies (size);
  TVectorD n_events (size);

  TVectorD Th40 (thsize); // Data arrays
  TVectorD Th60 (thsize);
  TVectorD Th80 (thsize);
  TVectorD Th100 (thsize);
  TVectorD Th40ey (thsize); // Error arrays
  TVectorD Th60ey (thsize);
  TVectorD Th80ey (thsize);
  TVectorD Th100ey (thsize);
  TVectorD ex (thsize);

  TVectorD HV (thsize);
  HV[0] = 4.60;
  HV[1] = 4.80;
  HV[2] = 5.00;
  HV[3] = 5.10;
  HV[4] = 5.15;
  HV[5] = 5.20;
  HV[6] = 5.25;

  for (Int_t i = 0; i < thsize; i++)
    {
      ex[i] = 0;
    }

  std::cout << "Number of runs: " << size << std::endl;

  efficiencies = efficiency(first_run,last_run);
  n_events = nevents(first_run,last_run);

  // Filling efficiency arrays for different thresholds
  Int_t count40 = 0;
  Int_t count60 = 0;
  Int_t count80 = 0;
  Int_t count100 = 0;
```

```
for (Int_t i = 0; i < size - 4; i = i + 12)
  {
    Th100[count100] = max( max(efficiencies[i],efficiencies[i+1]) , efficiencies[i+2]
   );
    Int_t j;
    if (efficiencies[i] > efficiencies[i+1]) {j = i;}
    else {j = i + 1;};
    if (efficiencies[i+2] > efficiencies[j]) {j = i + 2;};
    Th100ey[count100] = sqrt( Th100[count100] * (1 - Th100[count100]) / n_events[j] );
    count100++;
  }

for (Int_t i = 3; i < size - 4; i = i + 12)
  {
    Th80[count80] = max( max(efficiencies[i],efficiencies[i+1]) , efficiencies[i+2] );
    Int_t j;
    if (efficiencies[i] > efficiencies[i+1]) {j = i;}
    else {j = i + 1;};
    if (efficiencies[i+2] > efficiencies[j]) {j = i + 2;};
    Th80ey[count80] = sqrt( Th80[count80] * (1 - Th80[count80]) / n_events[j] );
    count80++;
  }

for (Int_t i = 6; i < size - 4; i = i + 12)
  {
    Th60[count60] = max( max(efficiencies[i],efficiencies[i+1]) , efficiencies[i+2] );
    Int_t j;
    if (efficiencies[i] > efficiencies[i+1]) {j = i;}
    else {j = i + 1;};
    if (efficiencies[i+2] > efficiencies[j]) {j = i + 2;};
    Th60ey[count60] = sqrt( Th60[count60] * (1 - Th60[count60]) / n_events[j] );
    count60++;
  }

for (Int_t i = 9; i < size - 4; i = i + 12)
  {
    Th40[count40] = max( max(efficiencies[i],efficiencies[i+1]) , efficiencies[i+2] );
    Int_t j;
    if (efficiencies[i] > efficiencies[i+1]) {j = i;}
    else {j = i + 1;};
    if (efficiencies[i+2] > efficiencies[j]) {j = i + 2;};
    Th40ey[count40] = sqrt( Th40[count40] * (1 - Th40[count40]) / n_events[j] );
    count40++;
  }

Int_t j = 0;

Th100[count100 - 1] = max( efficiencies[size - 4], Th100[count100 - 1]);
if (efficiencies[size - 4] > Th100[count100 - 1]) {j = size - 4;}
else {j = (count100 - 1) * 12;};
Th100ey[count100 - 1] = sqrt( Th100[count100 - 1] * (1 - Th100[count100 - 1]) /
   n_events[j] );
j = 0;

Th80[count80 - 1] = max( efficiencies[size - 3], Th80[count80 - 1]);
if (efficiencies[size - 3] > Th80[count80 - 1]) {j = size - 3;}
else {j = (count80 - 1) * 12 + 3;};
Th80ey[count80 - 1] = sqrt( Th80[count80 - 1] * (1 - Th80[count100 - 1]) / n_events[j]
    );
j = 0;

Th60[count60 - 1] = max( efficiencies[size - 2], Th60[count60 - 1]);
if (efficiencies[size - 2] > Th60[count60 - 1]) {j = size - 2;}
else {j = (count60 - 1) * 12 + 6;};
Th60ey[count60 - 1] = sqrt( Th60[count60 - 1] * (1 - Th60[count60 - 1]) / n_events[j]
   );
```

```
j = 0;

Th40[count40 - 1] = max( efficiencies[size - 1], Th40[count40 - 1]);
if (efficiencies[size - 1] > Th40[count40 - 1]) {j = size - 1;}
else {j = (count40 - 1) * 12 + 9;};
Th40ey[count40 - 1] = sqrt( Th40[count40 - 1] * (1 - Th40[count40 - 1]) / n_events[j]
   );
j = 0;

// Plotting HV scan
TCanvas* c1 = new TCanvas ("c1","Point P high voltage scan");

TGraphErrors *HVScan100 = new TGraphErrors (HV,Th100,ex,Th100ey);
HVScan100->SetTitle("Threshold -100 DAC steps");
HVScan100->SetMarkerColor(21);
HVScan100->SetLineColor(21);
TGraphErrors *HVScan80 = new TGraphErrors (HV,Th80,ex,Th80ey);
HVScan80->SetTitle("Threshold -80 DAC steps");
HVScan80->SetMarkerColor(2);
HVScan80->SetLineColor(2);
TGraphErrors *HVScan60 = new TGraphErrors (HV,Th60,ex,Th60ey);
HVScan60->SetTitle("Threshold -60 DAC steps");
HVScan60->SetMarkerColor(3);
HVScan60->SetLineColor(3);
TGraphErrors *HVScan40 = new TGraphErrors (HV,Th40,ex,Th40ey);
HVScan40->SetTitle("Threshold -40 DAC steps");
HVScan40->SetMarkerColor(4);
HVScan40->SetLineColor(4);

TMultiGraph *HVScan = new TMultiGraph();
HVScan->Add(HVScan100);
HVScan->Add(HVScan80);
HVScan->Add(HVScan60);
HVScan->Add(HVScan40);

HVScan->Draw("ALP");
c1->SetGridx();
c1->SetGridy();
c1->SetTickx();
c1->SetTicky();
HVScan->GetYaxis()->SetRangeUser(0,1);
HVScan->GetXaxis()->SetRangeUser(4.6,5.25);
HVScan->GetYaxis()->SetDecimals();

HVScan->SetTitle("Point P high voltage scan");
HVScan->GetXaxis()->SetTitle("Divider HV [kV]");
HVScan->GetYaxis()->SetTitle("LG efficiency");

TLegend *legend = new TLegend (0.1,0.4,0.4,0.9);
legend->SetHeader("High Voltage scan (point P)");
legend->AddEntry(HVScan100,HVScan100->GetTitle(),"lp");
legend->AddEntry(HVScan80,HVScan80->GetTitle(),"lp");
legend->AddEntry(HVScan60,HVScan60->GetTitle(),"lp");
legend->AddEntry(HVScan40,HVScan40->GetTitle(),"lp");
legend->Draw();

TF1 *f1 = new TF1 ("f1","x",767,875);
f1->Update();

TGaxis *current = new TGaxis (4.6, 1, 5.25, 1, "f1", 1020, "-");
current->SetLabelOffset(0.005);
current->SetLabelSize(0.04);
current->SetTickSize(0.03);
current->SetGridLength(0);
current->SetTitleOffset(1.1);
current->SetTitleSize(0.04);
current->SetTitleColor(1);
```

```
  current->SetTitleFont(62);
  current->SetTitle("Divider current [uA]");
  current->Draw();
  c1->Update();

}
```

Listing A.11: MSPLscanPmacro.C

```
#include <TGraph.h>
#include <TGraphErrors.h>
#include <TCanvas.h>
#include <TAxis.h>
#include <stdio.h>
#include <cstdio>
#include "TVectorT.h"
#include <stdlib.h>
#include <iostream>
#include "TMultiGraph.h"
#include "TAxis.h"
#include <TFile.h>
#include "TLegend.h"


TVectorD nevents(long int run_number, long int last_run);
TVectorD efficiency(long int run_number, long int last_run);

// MSPL scan for point P ranges from run 0109 to run 0268
TVectorD MSPLscanPmacro(long int first_run = 109, long int last_run = 268)
{

  Int_t size = last_run - first_run + 1;
  Int_t thsize = 10;

  TVectorD efficiencies (size);
  TVectorD n_events (size);

  TVectorD Th40mspl4 (thsize); // Data arrays
  TVectorD Th60mspl4 (thsize);
  TVectorD Th80mspl4 (thsize);
  TVectorD Th100mspl4 (thsize);
  TVectorD Th40mspl4ey (thsize); // Errors arrays
  TVectorD Th60mspl4ey (thsize);
  TVectorD Th80mspl4ey (thsize);
  TVectorD Th100mspl4ey (thsize);
  TVectorD Th40mspl3 (thsize); // Data arrays
  TVectorD Th60mspl3 (thsize);
  TVectorD Th80mspl3 (thsize);
  TVectorD Th100mspl3 (thsize);
  TVectorD Th40mspl3ey (thsize); // Errors arrays
  TVectorD Th60mspl3ey (thsize);
  TVectorD Th80mspl3ey (thsize);
  TVectorD Th100mspl3ey (thsize);
  TVectorD Th40mspl2 (thsize); // Data arrays
  TVectorD Th60mspl2 (thsize);
  TVectorD Th80mspl2 (thsize);
  TVectorD Th100mspl2 (thsize);
  TVectorD Th40mspl2ey (thsize); // Errors arrays
  TVectorD Th60mspl2ey (thsize);
  TVectorD Th80mspl2ey (thsize);
  TVectorD Th100mspl2ey (thsize);
  TVectorD Th40mspl1 (thsize); // Data arrays
  TVectorD Th60mspl1 (thsize);
  TVectorD Th80mspl1 (thsize);
  TVectorD Th100mspl1 (thsize);
  TVectorD Th40mspl1ey (thsize); // Errors arrays
```

```
TVectorD Th60mspl1ey (thsize);
TVectorD Th80mspl1ey (thsize);
TVectorD Th100mspl1ey (thsize);
TVectorD ex (thsize);

TVectorD lat (thsize); // X-axis contains latencies from 10 to 19
for (Int_t i = 10; i < 20; i++)
  {
    lat[i-10] = i;
  }

for (Int_t i = 0; i < thsize; i++) // No X errors
  {
    ex[i] = 0;
  }

efficiencies = efficiency(first_run,last_run);
n_events = nevents(first_run,last_run);

// Filling efficiency arrays for different monostable pulse lenghts and thresholds

for (Int_t i = 0; i < 10; i++)
  {
    Int_t count404 = i; // MSPL = 4clk
    Int_t count604 = i + 10;
    Int_t count804 = i + 20;
    Int_t count1004 = i + 30;
    Int_t count403 = i + 40; // MSPL = 3clk
    Int_t count603 = i + 50;
    Int_t count803 = i + 60;
    Int_t count1003 = i + 70;
    Int_t count402 = i + 80; // MSPL = 2clk
    Int_t count602 = i + 90;
    Int_t count802 = i + 100;
    Int_t count1002 = i + 110;
    Int_t count401 = i + 120; // MSPL = 1clk
    Int_t count601 = i + 130;
    Int_t count801 = i + 140;
    Int_t count1001 = i + 150;

    Th40mspl4[i] = efficiencies[count404];
    Th40mspl4ey[i] = sqrt( efficiencies[count404] * (1 - efficiencies[count404]) /
  n_events[count404] );

    Th60mspl4[i] = efficiencies[count604];
    Th60mspl4ey[i] = sqrt( efficiencies[count604] * (1 - efficiencies[count604]) /
  n_events[count604] );

    Th80mspl4[i] = efficiencies[count804];
    Th80mspl4ey[i] = sqrt( efficiencies[count804] * (1 - efficiencies[count804]) /
  n_events[count804] );

    Th100mspl4[i] = efficiencies[count1004];
    Th100mspl4ey[i] = sqrt( efficiencies[count1004] * (1 - efficiencies[count1004]) /
  n_events[count1004] );


    Th40mspl3[i] = efficiencies[count403];
    Th40mspl3ey[i] = sqrt( efficiencies[count403] * (1 - efficiencies[count403]) /
  n_events[count403] );

    Th60mspl3[i] = efficiencies[count603];
    Th60mspl3ey[i] = sqrt( efficiencies[count603] * (1 - efficiencies[count603]) /
  n_events[count603] );

    Th80mspl3[i] = efficiencies[count803];
    Th80mspl3ey[i] = sqrt( efficiencies[count803] * (1 - efficiencies[count803]) /
```

```
    n_events[count803] );

      Th100mspl3[i] = efficiencies[count1003];
      Th100mspl3ey[i] = sqrt( efficiencies[count1003] * (1 - efficiencies[count1003]) /
    n_events[count1003] );


      Th40mspl2[i] = efficiencies[count402];
      Th40mspl2ey[i] = sqrt( efficiencies[count402] * (1 - efficiencies[count402]) /
    n_events[count402] );

      Th60mspl2[i] = efficiencies[count602];
      Th60mspl2ey[i] = sqrt( efficiencies[count602] * (1 - efficiencies[count602]) /
    n_events[count602] );

      Th80mspl2[i] = efficiencies[count802];
      Th80mspl2ey[i] = sqrt( efficiencies[count802] * (1 - efficiencies[count802]) /
    n_events[count802] );

      Th100mspl2[i] = efficiencies[count1002];
      Th100mspl2ey[i] = sqrt( efficiencies[count1002] * (1 - efficiencies[count1002]) /
    n_events[count1002] );


      Th40mspl1[i] = efficiencies[count401];
      Th40mspl1ey[i] = sqrt( efficiencies[count401] * (1 - efficiencies[count401]) /
    n_events[count401] );

      Th60mspl1[i] = efficiencies[count601];
      Th60mspl1ey[i] = sqrt( efficiencies[count601] * (1 - efficiencies[count601]) /
    n_events[count601] );

      Th80mspl1[i] = efficiencies[count801];
      Th80mspl1ey[i] = sqrt( efficiencies[count801] * (1 - efficiencies[count801]) /
    n_events[count801] );

      Th100mspl1[i] = efficiencies[count1001];
      Th100mspl1ey[i] = sqrt( efficiencies[count1001] * (1 - efficiencies[count1001]) /
    n_events[count1001] );


    }

  // Plotting MSPL scan
  TCanvas* c1 = new TCanvas ("c1","Point P MSPL/Latency scan");

  //MSPL = 4 clk
  TGraphErrors *mspl4th100 = new TGraphErrors (lat,Th100mspl4,ex,Th100mspl4ey);
  mspl4th100->SetTitle("MSPL 4clk | Threshold -100 DAC steps");
  mspl4th100->SetMarkerColor(kBlue+3);
  mspl4th100->SetLineColor(kBlue+3);
  mspl4th100->SetLineWidth(2);

  TGraphErrors *mspl4th80 = new TGraphErrors (lat,Th80mspl4,ex,Th80mspl4ey);
  mspl4th80->SetTitle("MSPL 4clk | Threshold -80 DAC steps");
  mspl4th80->SetMarkerColor(kBlue);
  mspl4th80->SetLineColor(kBlue);
  mspl4th80->SetLineWidth(2);

  TGraphErrors *mspl4th60 = new TGraphErrors (lat,Th60mspl4,ex,Th60mspl4ey);
  mspl4th60->SetTitle("MSPL 4clk | Threshold -60 DAC steps");
  mspl4th60->SetMarkerColor(kBlue-5);
  mspl4th60->SetLineColor(kBlue-5);
  mspl4th60->SetLineWidth(2);

  TGraphErrors *mspl4th40 = new TGraphErrors (lat,Th40mspl4,ex,Th40mspl4ey);
  mspl4th40->SetTitle("MSPL 4clk | Threshold -40 DAC steps");
```

```
mspl4th40->SetMarkerColor(kBlue-10);
mspl4th40->SetLineColor(kBlue-10);
mspl4th40->SetLineWidth(2);
mspl4th40->SetLineStyle(7);

//MSPL = 3clk
TGraphErrors *mspl3th100 = new TGraphErrors (lat,Th100mspl3,ex,Th100mspl3ey);
mspl3th100->SetTitle("MSPL 3clk | Threshold -100 DAC steps");
mspl3th100->SetMarkerColor(kSpring-6);
mspl3th100->SetLineColor(kSpring-6);
mspl3th100->SetLineWidth(2);

TGraphErrors *mspl3th80 = new TGraphErrors (lat,Th80mspl3,ex,Th80mspl3ey);
mspl3th80->SetTitle("MSPL 3clk | Threshold -80 DAC steps");
mspl3th80->SetMarkerColor(kSpring+4);
mspl3th80->SetLineColor(kSpring+4);
mspl3th80->SetLineWidth(2);

TGraphErrors *mspl3th60 = new TGraphErrors (lat,Th60mspl3,ex,Th60mspl3ey);
mspl3th60->SetTitle("MSPL 3clk | Threshold -60 DAC steps");
mspl3th60->SetMarkerColor(kSpring-1);
mspl3th60->SetLineColor(kSpring-1);
mspl3th60->SetLineWidth(2);

TGraphErrors *mspl3th40 = new TGraphErrors (lat,Th40mspl3,ex,Th40mspl3ey);
mspl3th40->SetTitle("MSPL 3clk | Threshold -40 DAC steps");
mspl3th40->SetMarkerColor(kSpring+7);
mspl3th40->SetLineColor(kSpring+7);
mspl3th40->SetLineWidth(2);
mspl3th40->SetLineStyle(7);

//MSPL = 2clk
TGraphErrors *mspl2th100 = new TGraphErrors (lat,Th100mspl2,ex,Th100mspl2ey);
mspl2th100->SetTitle("MSPL 2clk | Threshold -100 DAC steps");
mspl2th100->SetMarkerColor(kOrange+9);
mspl2th100->SetLineColor(kOrange+9);
mspl2th100->SetLineWidth(2);

TGraphErrors *mspl2th80 = new TGraphErrors (lat,Th80mspl2,ex,Th80mspl2ey);
mspl2th80->SetTitle("MSPL 2clk | Threshold -80 DAC steps");
mspl2th80->SetMarkerColor(kOrange+7);
mspl2th80->SetLineColor(kOrange+7);
mspl2th80->SetLineWidth(2);

TGraphErrors *mspl2th60 = new TGraphErrors (lat,Th60mspl2,ex,Th60mspl2ey);
mspl2th60->SetTitle("MSPL 2clk | Threshold -60 DAC steps");
mspl2th60->SetMarkerColor(kOrange-3);
mspl2th60->SetLineColor(kOrange-3);
mspl2th60->SetLineWidth(2);

TGraphErrors *mspl2th40 = new TGraphErrors (lat,Th40mspl2,ex,Th40mspl2ey);
mspl2th40->SetTitle("MSPL 2clk | Threshold -40 DAC steps");
mspl2th40->SetMarkerColor(kOrange-2);
mspl2th40->SetLineColor(kOrange-2);
mspl2th40->SetLineWidth(2);
mspl2th40->SetLineStyle(7);

//MSPL = 1clk
TGraphErrors *mspl1th100 = new TGraphErrors (lat,Th100mspl1,ex,Th100mspl1ey);
mspl1th100->SetTitle("MSPL 1clk | Threshold -100 DAC steps");
mspl1th100->SetMarkerColor(kMagenta+2);
mspl1th100->SetLineColor(kMagenta+2);
mspl1th100->SetLineWidth(2);

TGraphErrors *mspl1th80 = new TGraphErrors (lat,Th80mspl1,ex,Th80mspl1ey);
mspl1th80->SetTitle("MSPL 1clk | Threshold -80 DAC steps");
mspl1th80->SetMarkerColor(kMagenta);
```

```
mspl1th80->SetLineColor(kMagenta);
mspl1th80->SetLineWidth(2);

TGraphErrors *mspl1th60 = new TGraphErrors (lat,Th60mspl1,ex,Th60mspl1ey);
mspl1th60->SetTitle("MSPL 1clk | Threshold -60 DAC steps");
mspl1th60->SetMarkerColor(kMagenta-7);
mspl1th60->SetLineColor(kMagenta-7);
mspl1th60->SetLineWidth(2);

TGraphErrors *mspl1th40 = new TGraphErrors (lat,Th40mspl1,ex,Th40mspl1ey);
mspl1th40->SetTitle("MSPL 1clk | Threshold -40 DAC steps");
mspl1th40->SetMarkerColor(kMagenta-9);
mspl1th40->SetLineColor(kMagenta-9);
mspl1th40->SetLineWidth(2);
mspl1th40->SetLineStyle(7);


TMultiGraph *MSPLscan = new TMultiGraph();
MSPLscan->Add(mspl4th100);
MSPLscan->Add(mspl4th80);
MSPLscan->Add(mspl4th60);
MSPLscan->Add(mspl4th40);
MSPLscan->Add(mspl3th100);
MSPLscan->Add(mspl3th80);
MSPLscan->Add(mspl3th60);
MSPLscan->Add(mspl3th40);
MSPLscan->Add(mspl2th100);
MSPLscan->Add(mspl2th80);
MSPLscan->Add(mspl2th60);
MSPLscan->Add(mspl2th40);
MSPLscan->Add(mspl1th100);
MSPLscan->Add(mspl1th80);
MSPLscan->Add(mspl1th60);
MSPLscan->Add(mspl1th40);

MSPLscan->Draw("ALP");
c1->SetGridx();
c1->SetGridy();
c1->SetTickx();
c1->SetTicky();
MSPLscan->GetYaxis()->SetRangeUser(0,1);
MSPLscan->GetXaxis()->SetRangeUser(10,19);
MSPLscan->GetYaxis()->SetDecimals();
MSPLscan->SetTitle("Point P signal timing scan");
MSPLscan->GetXaxis()->SetTitle("Latency [clock cycles = 25ns steps]");
MSPLscan->GetYaxis()->SetTitle("LG efficiency");


c1->Update();
TLegend *legend = new TLegend (0.1,0.4,0.4,0.9);
legend->SetHeader("#splitline{Latency scan @ Ar-CO2 70/30}{[I=859#muA, th=-60steps]}")
   ;
legend->AddEntry(mspl4th40,mspl4th40->GetTitle(),"lp");
legend->AddEntry(mspl4th60,mspl4th60->GetTitle(),"lp");
legend->AddEntry(mspl4th80,mspl4th80->GetTitle(),"lp");
legend->AddEntry(mspl4th100,mspl4th100->GetTitle(),"lp");
legend->AddEntry(mspl3th40,mspl3th40->GetTitle(),"lp");
legend->AddEntry(mspl3th60,mspl3th60->GetTitle(),"lp");
legend->AddEntry(mspl3th80,mspl3th80->GetTitle(),"lp");
legend->AddEntry(mspl3th100,mspl3th100->GetTitle(),"lp");
legend->AddEntry(mspl2th40,mspl2th40->GetTitle(),"lp");
legend->AddEntry(mspl2th60,mspl2th60->GetTitle(),"lp");
legend->AddEntry(mspl2th80,mspl2th80->GetTitle(),"lp");
legend->AddEntry(mspl2th100,mspl2th100->GetTitle(),"lp");
legend->AddEntry(mspl1th40,mspl1th40->GetTitle(),"lp");
legend->AddEntry(mspl1th60,mspl1th60->GetTitle(),"lp");
legend->AddEntry(mspl1th80,mspl1th80->GetTitle(),"lp");
```

```
  legend->AddEntry(mspl1th100,mspl1th100->GetTitle(),"lp");
  legend->Draw();

  c1->Update();

  return Th60mspl2;

}


// MSPL scan for point p ranges from run 0109 to run 0268
TVectorD MSPLscanPmacroey(long int first_run = 109, long int last_run = 268)
{

  Int_t size = last_run - first_run + 1;
  Int_t thsize = 10;

  TVectorD efficiencies (size);
  TVectorD n_events (size);

  TVectorD Th40mspl4 (thsize); // Data arrays
  TVectorD Th60mspl4 (thsize);
  TVectorD Th80mspl4 (thsize);
  TVectorD Th100mspl4 (thsize);
  TVectorD Th40mspl4ey (thsize); // Errors arrays
  TVectorD Th60mspl4ey (thsize);
  TVectorD Th80mspl4ey (thsize);
  TVectorD Th100mspl4ey (thsize);
  TVectorD Th40mspl3 (thsize); // Data arrays
  TVectorD Th60mspl3 (thsize);
  TVectorD Th80mspl3 (thsize);
  TVectorD Th100mspl3 (thsize);
  TVectorD Th40mspl3ey (thsize); // Errors arrays
  TVectorD Th60mspl3ey (thsize);
  TVectorD Th80mspl3ey (thsize);
  TVectorD Th100mspl3ey (thsize);
  TVectorD Th40mspl2 (thsize); // Data arrays
  TVectorD Th60mspl2 (thsize);
  TVectorD Th80mspl2 (thsize);
  TVectorD Th100mspl2 (thsize);
  TVectorD Th40mspl2ey (thsize); // Errors arrays
  TVectorD Th60mspl2ey (thsize);
  TVectorD Th80mspl2ey (thsize);
  TVectorD Th100mspl2ey (thsize);
  TVectorD Th40mspl1 (thsize); // Data arrays
  TVectorD Th60mspl1 (thsize);
  TVectorD Th80mspl1 (thsize);
  TVectorD Th100mspl1 (thsize);
  TVectorD Th40mspl1ey (thsize); // Errors arrays
  TVectorD Th60mspl1ey (thsize);
  TVectorD Th80mspl1ey (thsize);
  TVectorD Th100mspl1ey (thsize);
  TVectorD ex (thsize);

  TVectorD lat (thsize); // X-axis contains latencies from 10 to 19
  for (Int_t i = 10; i < 20; i++)
    {
      lat[i-10] = i;
    }

  for (Int_t i = 0; i < thsize; i++) // No X errors
    {
      ex[i] = 0;
    }

  efficiencies = efficiency(first_run,last_run);
  n_events = nevents(first_run,last_run);
```

```
// Filling efficiency arrays for different monostable pulse lenghts and thresholds

for (Int_t i = 0; i < 10; i++)
  {
    Int_t count404 = i; // MSPL = 4clk
    Int_t count604 = i + 10;
    Int_t count804 = i + 20;
    Int_t count1004 = i + 30;
    Int_t count403 = i + 40; // MSPL = 3clk
    Int_t count603 = i + 50;
    Int_t count803 = i + 60;
    Int_t count1003 = i + 70;
    Int_t count402 = i + 80; // MSPL = 2clk
    Int_t count602 = i + 90;
    Int_t count802 = i + 100;
    Int_t count1002 = i + 110;
    Int_t count401 = i + 120; // MSPL = 1clk
    Int_t count601 = i + 130;
    Int_t count801 = i + 140;
    Int_t count1001 = i + 150;

    Th40mspl4[i] = efficiencies[count404];
    Th40mspl4ey[i] = sqrt( efficiencies[count404] * (1 - efficiencies[count404]) /
  n_events[count404] );

    Th60mspl4[i] = efficiencies[count604];
    Th60mspl4ey[i] = sqrt( efficiencies[count604] * (1 - efficiencies[count604]) /
  n_events[count604] );

    Th80mspl4[i] = efficiencies[count804];
    Th80mspl4ey[i] = sqrt( efficiencies[count804] * (1 - efficiencies[count804]) /
  n_events[count804] );

    Th100mspl4[i] = efficiencies[count1004];
    Th100mspl4ey[i] = sqrt( efficiencies[count1004] * (1 - efficiencies[count1004]) /
  n_events[count1004] );


    Th40mspl3[i] = efficiencies[count403];
    Th40mspl3ey[i] = sqrt( efficiencies[count403] * (1 - efficiencies[count403]) /
  n_events[count403] );

    Th60mspl3[i] = efficiencies[count603];
    Th60mspl3ey[i] = sqrt( efficiencies[count603] * (1 - efficiencies[count603]) /
  n_events[count603] );

    Th80mspl3[i] = efficiencies[count803];
    Th80mspl3ey[i] = sqrt( efficiencies[count803] * (1 - efficiencies[count803]) /
  n_events[count803] );

    Th100mspl3[i] = efficiencies[count1003];
    Th100mspl3ey[i] = sqrt( efficiencies[count1003] * (1 - efficiencies[count1003]) /
  n_events[count1003] );


    Th40mspl2[i] = efficiencies[count402];
    Th40mspl2ey[i] = sqrt( efficiencies[count402] * (1 - efficiencies[count402]) /
  n_events[count402] );

    Th60mspl2[i] = efficiencies[count602];
    Th60mspl2ey[i] = sqrt( efficiencies[count602] * (1 - efficiencies[count602]) /
  n_events[count602] );

    Th80mspl2[i] = efficiencies[count802];
    Th80mspl2ey[i] = sqrt( efficiencies[count802] * (1 - efficiencies[count802]) /
  n_events[count802] );
```

```
      Th100mspl2[i] = efficiencies[count1002];
      Th100mspl2ey[i] = sqrt( efficiencies[count1002] * (1 - efficiencies[count1002]) /
    n_events[count1002] );


      Th40mspl1[i] = efficiencies[count401];
      Th40mspl1ey[i] = sqrt( efficiencies[count401] * (1 - efficiencies[count401]) /
    n_events[count401] );

      Th60mspl1[i] = efficiencies[count601];
      Th60mspl1ey[i] = sqrt( efficiencies[count601] * (1 - efficiencies[count601]) /
    n_events[count601] );

      Th80mspl1[i] = efficiencies[count801];
      Th80mspl1ey[i] = sqrt( efficiencies[count801] * (1 - efficiencies[count801]) /
    n_events[count801] );

      Th100mspl1[i] = efficiencies[count1001];
      Th100mspl1ey[i] = sqrt( efficiencies[count1001] * (1 - efficiencies[count1001]) /
    n_events[count1001] );


    }

  return Th60mspl2ey;

}
```

Listing A.12: THscanPmacro.C

```
#include <TGraph.h>
#include <TGraphErrors.h>
#include <TCanvas.h>
#include <TAxis.h>
#include <stdio.h>
#include <cstdio>
#include "TVectorT.h"
#include <stdlib.h>
#include <iostream>
#include "TMultiGraph.h"
#include "TAxis.h"
#include <TFile.h>


TVectorD nevents(long int run_number, long int last_run);
TVectorD efficiency(long int run_number, long int last_run);

void THscanPmacro(long int first_run = 269, long int last_run = 336)
{

  Int_t size = last_run - first_run + 1;
  Int_t thsize = 17;

  TVectorD efficiencies (size);
  TVectorD n_events (size);

  TVectorD hv5p15 (thsize); // Data arrays
  TVectorD hv5p10 (thsize);
  TVectorD hv5p05 (thsize);
  TVectorD hv5p00 (thsize);
  TVectorD hv5p15ey (thsize); // Data arrays
  TVectorD hv5p10ey (thsize);
  TVectorD hv5p05ey (thsize);
  TVectorD hv5p00ey (thsize);
  TVectorD ex (thsize);
```

```
TVectorD th (thsize);
th[0] = 10;
th[1] = 20;
th[2] = 30;
th[3] = 40;
th[4] = 50;
th[5] = 60;
th[6] = 70;
th[7] = 80;
th[8] = 90;
th[9] = 100;
th[10] = 120;
th[11] = 140;
th[12] = 160;
th[13] = 180;
th[14] = 200;
th[15] = 225;
th[16] = 250;

for (Int_t i = 0; i < thsize; i++) // No X errors
  {
    ex[i] = 0;
  }

std::cout << "Number of runs: " << size << std::endl;

efficiencies = efficiency(first_run,last_run);
n_events = nevents(first_run,last_run);


for (Int_t i = 0; i < thsize; i++)
  {
    Int_t hv515 = i;
    Int_t hv510 = i + 17;
    Int_t hv505 = i + 34;
    Int_t hv500 = i + 51;

    hv5p15[i] = efficiencies[hv515];
    hv5p15ey[i] = sqrt( efficiencies[hv515] * (1 - efficiencies[hv515]) / n_events[
   hv515] );

    hv5p10[i] = efficiencies[hv510];
    hv5p10ey[i] = sqrt( efficiencies[hv510] * (1 - efficiencies[hv510]) / n_events[
   hv510] );

    hv5p05[i] = efficiencies[hv505];
    hv5p05ey[i] = sqrt( efficiencies[hv505] * (1 - efficiencies[hv505]) / n_events[
   hv505] );

    hv5p00[i] = efficiencies[hv500];
    hv5p00ey[i] = sqrt( efficiencies[hv500] * (1 - efficiencies[hv500]) / n_events[
   hv500] );
  }

// Plotting MSPL scan
TCanvas* c1 = new TCanvas ("c1","Point P MSPL/Latency scan");

//MSPL = 4 clk
TGraphErrors *thscan5p15 = new TGraphErrors (th,hv5p15,ex,hv5p15ey);
thscan5p15->SetTitle("HV -5.15kV (859uA)");
thscan5p15->SetMarkerColor(kBlue);
thscan5p15->SetLineColor(kBlue);
thscan5p15->SetLineWidth(2);

TGraphErrors *thscan5p10 = new TGraphErrors (th,hv5p10,ex,hv5p10ey);
thscan5p10->SetTitle("HV -5.10kV (851uA)");
```

```
  thscan5p10->SetMarkerColor(kSpring);
  thscan5p10->SetLineColor(kSpring);
  thscan5p10->SetLineWidth(2);

  TGraphErrors *thscan5p05 = new TGraphErrors (th,hv5p05,ex,hv5p05ey);
  thscan5p05->SetTitle("HV -5.05kV (841uA)");
  thscan5p05->SetMarkerColor(kOrange);
  thscan5p05->SetLineColor(kOrange);
  thscan5p05->SetLineWidth(2);

  TGraphErrors *thscan5p00 = new TGraphErrors (th,hv5p00,ex,hv5p00ey);
  thscan5p00->SetTitle("HV -5.00kV (834uA)");
  thscan5p00->SetMarkerColor(kMagenta+2);
  thscan5p00->SetLineColor(kMagenta+2);
  thscan5p00->SetLineWidth(2);


  TMultiGraph *THScanPlot = new TMultiGraph();
  THScanPlot->Add(thscan5p15);
  THScanPlot->Add(thscan5p10);
  THScanPlot->Add(thscan5p05);
  THScanPlot->Add(thscan5p00);

  THScanPlot->Draw("ALP");
  THScanPlot->GetXaxis()->SetRangeUser(10,250);
  THScanPlot->GetYaxis()->SetRangeUser(0,1);
  THScanPlot->GetYaxis()->SetDecimals();
  THScanPlot->GetXaxis()->SetTitle("Negative threshold [VFAT2 DAC steps = 3.3mV = 0.045
     fC]");
  THScanPlot->GetYaxis()->SetTitle("LG efficiency");
  c1->Update();
  c1->SetGridx();
  c1->SetGridy();
  c1->SetTickx();
  c1->SetTicky();
  TLegend *legend = new TLegend (0.1,0.4,0.4,0.9);
  legend->SetHeader("#splitline{Threshold scan (point P)}{[MSPL 4clk, lat 14clk]}");
  legend->AddEntry(thscan5p15,thscan5p15->GetTitle(),"lp");
  legend->AddEntry(thscan5p10,thscan5p10->GetTitle(),"lp");
  legend->AddEntry(thscan5p05,thscan5p05->GetTitle(),"lp");
  legend->AddEntry(thscan5p00,thscan5p00->GetTitle(),"lp");
  legend->Draw();
  c1->Update();

}
```

Listing A.13: CPScan.cc

```
#include <string>
#include <TTree.h>
#include <TFile.h>
#include <cstdio>
#include <TGraph.h>
#include <TMultiGraph.h>
#include <TCanvas.h>
#include <TF1.h>
#include <stdio.h>
#include <TAxis.h>
#include "TotemMap.cpp"
#include <TVectorT.h>
#include "TLegend.h"


using namespace std;

// This file contains routines for calibration pulse scans
```

```
void CalPulseAnalizer(TString inputfile, TString outputfile, const Int_t nlines, Int_t
     vfatnumber);
double GetX(int channel);
double GetY(int channel);
double dist(double x0, double y0, double x1, double y1);

Double_t GetR(int channel)
{
  Double_t channelX = GetX(channel);
  Double_t channelY = GetY(channel);
  Double_t radius;
  radius = dist(channelX, channelY, 0, 0);
  return radius;
}


TVectorD CPScan(long int firstPad = 2, long int lastPad = 128, Int_t nFiles = 64, Int_t
     nVFAT = 0){

        // Defining vectors
        TVectorD nPad (nFiles);
        TVectorD nRadius (nFiles);
        TVectorD nSigma (nFiles);
        TVectorD nMean (nFiles);
        // Initializing values
        Double_t Sigma = 0;
        Double_t Mean = 0;
        // Defining filename strings
        string rawfile_name;
        string rootfile_name;

        // Cycling over the channels indicated
        long int count=0;
        for (long int i=firstPad;i<lastPad+1;i=i+2)
        {
              nRadius[count] = GetR(i);

              // Converting i to a string to open the i-th file
              string s;
              char Pad_n[30];
              sprintf(Pad_n,"%ld",i);
              s = string(Pad_n);

              if (i<10)
              {
                rootfile_name = "CP-RootData/CPscan-00" + s + ".root";
                rawfile_name = "CP-RawData/00" + s + ".dat";
              }
              else
              {
                if (i>=10 && i<100)
                {
                        rootfile_name = "CP-RootData/CPscan-0" + s + ".root";
                        rawfile_name = "CP-RawData/0" + s + ".dat";
                }
                else
                {
                        rootfile_name = "CP-RootData/CPscan-" + s + ".root";
                        rawfile_name = "CP-RawData/" + s + ".dat";
                }
              }
              nPad[count] = i;

              CalPulseAnalizer(rawfile_name,rootfile_name,151,nVFAT);

              TFile *rootfile = new TFile(rootfile_name.c_str());
```

```
                TGraph* tempG = dynamic_cast<TGraph*>(rootfile->Get("Graph"));
                Sigma = tempG->GetFunction("sigmoid")->GetParameter(2);
                nSigma[count] = Sigma;
                Mean = tempG->GetFunction("sigmoid")->GetParameter(1);
                nMean[count] = Mean;
                count++;
                delete tempG;
                rootfile->Close();
        }

                  return nSigma;
}


TVectorD CPScanRadius(long int firstPad = 2, long int lastPad = 128, Int_t nFiles = 64,
    Int_t nVFAT = 0){

  TVectorD nPad (nFiles);
  TVectorD nRadius (nFiles);
  TVectorD nSigma (nFiles);
  TVectorD nMean (nFiles);
        // Initializing values
        Double_t Sigma = 0;
        Double_t Mean = 0;
        // Defining filename strings
        string rawfile_name;
        string rootfile_name;

        // Cycling over the channels indicated
        long int count=0;
        for (long int i=firstPad;i<lastPad+1;i=i+2)
        {
                nRadius[count] = GetR(i);

                // Converting i to a string to open the i-th file
                string s;
                char Pad_n[30];
                sprintf(Pad_n,"%ld",i);
                s = string(Pad_n);

                if (i<10)
                {
                  rootfile_name = "CP-RootData/CPscan-00" + s + ".root";
                  rawfile_name = "CP-RawData/00" + s + ".dat";
                }
                else
                {
                  if (i>=10 && i<100)
                  {
                          rootfile_name = "CP-RootData/CPscan-0" + s + ".root";
                          rawfile_name = "CP-RawData/0" + s + ".dat";
                  }
                  else
                  {
                          rootfile_name = "CP-RootData/CPscan-" + s + ".root";
                          rawfile_name = "CP-RawData/" + s + ".dat";
                  }
                }
                nPad[count] = i;

                CalPulseAnalizer(rawfile_name,rootfile_name,151,nVFAT);

                TFile *rootfile = new TFile(rootfile_name.c_str());
                TGraph* tempG = dynamic_cast<TGraph*>(rootfile->Get("Graph"));
                Sigma = tempG->GetFunction("sigmoid")->GetParameter(2);
                nSigma[count] = Sigma;
                Mean = tempG->GetFunction("sigmoid")->GetParameter(1);
```

```
                nMean[count] = Mean;
                count++;
                delete tempG;
                rootfile->Close();
        }

        return nRadius;
}


void MakeSPlot(Int_t nFiles = 64){

  TVectorD nSigma0 (nFiles);
  TVectorD nSigma1 (nFiles);
  TVectorD nSigma2 (nFiles);
  TVectorD nSigma3 (nFiles);
  TVectorD nSigma4 (nFiles);
  TVectorD nSigma5 (nFiles);
  TVectorD nSigma6 (nFiles);
  TVectorD nSigma7 (nFiles);
  TVectorD nRadius (nFiles);

  nSigma0 = CPScan(2,128,64,0);
  nSigma1 = CPScan(2,128,64,1);
  nSigma2 = CPScan(2,128,64,2);
  nSigma3 = CPScan(2,128,64,3);
  nSigma4 = CPScan(2,128,64,4);
  nSigma5 = CPScan(2,128,64,5);
  nSigma6 = CPScan(2,128,64,6);
  nSigma7 = CPScan(2,128,64,7);

  nRadius = CPScanRadius(2,128,64,0);

  TCanvas* c1 = new TCanvas ("c1","S-Curve sigma for different readout pads");

  TGraph *plot0 = new TGraph(nRadius,nSigma0);
  plot0->SetMarkerColor(42);
  plot0->SetTitle("VFAT0");
  plot0->SetMarkerStyle(3);

  TGraph *plot1 = new TGraph(nRadius,nSigma1);
  plot1->SetMarkerColor(43);
  plot1->SetTitle("VFAT1");
  plot1->SetMarkerStyle(4);

  TGraph *plot2 = new TGraph(nRadius,nSigma2);
  plot2->SetMarkerColor(44);
  plot2->SetTitle("VFAT2");
  plot2->SetMarkerStyle(25);

  TGraph *plot3 = new TGraph(nRadius,nSigma3);
  plot3->SetMarkerColor(45);
  plot3->SetTitle("VFAT3");
  plot3->SetMarkerStyle(26);

  TGraph *plot4 = new TGraph(nRadius,nSigma4);
  plot4->SetMarkerColor(46);
  plot4->SetTitle("VFAT4");
  plot4->SetMarkerStyle(27);

  TGraph *plot5 = new TGraph(nRadius,nSigma5);
  plot5->SetMarkerColor(47);
  plot5->SetTitle("VFAT5");
  plot5->SetMarkerStyle(28);

  TGraph *plot6 = new TGraph(nRadius,nSigma6);
  plot6->SetMarkerColor(48);
```

```
  plot6->SetTitle("VFAT6");
  plot6->SetMarkerStyle(30);

  TGraph *plot7 = new TGraph(nRadius,nSigma7);
  plot7->SetLineColor(12);
  plot7->SetMarkerColor(12);
  plot7->SetTitle("VFAT7 channels [DISCONNECTED]");

  TMultiGraph *multiplot = new TMultiGraph;
  multiplot->Add(plot0);
  multiplot->Add(plot1);
  multiplot->Add(plot2);
  multiplot->Add(plot3);
  multiplot->Add(plot4);
  multiplot->Add(plot5);
  multiplot->Add(plot6);

  multiplot->Draw("AP");
  multiplot->GetXaxis()->SetRangeUser(155,725);
  multiplot->GetXaxis()->SetTitle("Pad distance from chamber's vertex [mm]");
  multiplot->GetYaxis()->SetTitle("Noise amplitude [VFAT2 DAC steps]");
  c1->SetGridx();
  c1->SetGridy();
  c1->SetTickx();
  c1->SetTicky();
  c1->Update();

  TLegend *legend = new TLegend (0.1,0.4,0.4,0.9);
  legend->SetHeader("S-Curve Sigma");
  legend->AddEntry(plot0,plot0->GetTitle(),"p");
  legend->AddEntry(plot1,plot1->GetTitle(),"p");
  legend->AddEntry(plot2,plot2->GetTitle(),"p");
  legend->AddEntry(plot3,plot3->GetTitle(),"p");
  legend->AddEntry(plot4,plot4->GetTitle(),"p");
  legend->AddEntry(plot5,plot5->GetTitle(),"p");
  legend->AddEntry(plot6,plot6->GetTitle(),"p");
  legend->Draw();

  c1->Update();

}
```

List of Figures

List of Tables

Listings

Bibliography

[1] Review of particle physics, 2008. Particle Data Group.

[2] TOTEM technical design report, January 2008. CERN-LHCC-2004-002.

[3] The TOTEM experiment at the CERN large hadron collider. In A. Breskin and R. Voss, editors, *The CERN Large Hadron Collider: Accelerator and Experiments*. CERN, 2009. 2008 JINST 3 S08007.

[4] M. Alfonsi, G. Bencivenni, P. De Simone, F. Murtas, M. Poli Lerner, D. Pinci, W. Bonivento, A. Cardini, C. Deplano, D. Raspino, B. Saitta, and S. Baccaro. Operation of triple-GEM detectors with fast gas mixtures, October 2003.

[5] P. Aspell. *VFAT2 - Operating Manual*, October 2008.

[6] P. Aspell. VFAT. In *CMS high-η MPGD workshop*, September 2010.

[7] G. Bencivenni, G. Felici, F. Murtas, P. Valente, W. Bonivento, A. Cardini, A. Lai, D. Pinci, B. Saitta, and C. Bosio. A triple-GEM detector with pad readout for high rate charged particle triggering. *Nuclear Instrumentation and Methods*, A 488(493), 2002.

[8] R. Bouclier, M. Capeáns, W. Dominik, M. Hoch, J. C. Labbé, G. Million, L. Ropelewski, F. Sauli, and A. Sharma. The gas electron multiplier (GEM).

[9] DuPont. Kapton polyimide film: general specifications. Bulletin GS-96-7.

[10] S. Lami. Preliminary study for a possible T1 upgrade with large GEMs. In *CMS high-η MPGD workshop*, September 2010.

[11] E. Oliveri and E. Graverini. Large GEM detector: August TB results. In *6th RD51 Collaboration meeting*, October 2010.

[12] S. D. Pinto. A large area GEM detector: manufacturing of first large prototype. In *CMS high-η MPGD workshop*, September 2010.

[13] S. D. Pinto, M. Alfonsi, I. Brock, G. Croci, E. David, R. de Oliveira, B.-E. Pinchasik, L. Ropelewski, F. Sauli, and M. van Stenis. A large area GEM detector. In *IEEE Nuclear Science Symposium Conference*, 2008.

[14] F. Sauli. Principles of operation of multiwire proportional and drift chambers. Academic training programme lectures, CERN, 1977.

[15] A. Sharma. Properties of some gas mixtures used in tracking detectors.