Etude statistique des particules de haute énergie issues du rayonnement cosmique

Comment détecter le flux de muons, particules issues du rayonnement cosmique, arrivant sur Terre?

Donatella AVONI
Tomasz BIALAS
Nils VAN WEELDEREN

En classe de 1^{è}S2

TPE 2014-2015

Professeurs encadrant: Madame CAILLAREC et Monsieur WAITIER

Introduction

Les rayons cosmiques ne sont pas un sujet dont on parle tous les jours; pourtant, cela pourrait nous surprendre si l'on disait que chaque seconde, des milliers de particules issues de ces rayons nous traversent sans qu'on s'en aperçoive. C'est donc un esprit de curiosité qui nous a donné la motivation pour entreprendre ce projet.

Tout est parti de notre réalité: un TPE à réaliser ayant pour matières maths et physique et aucune idée sur où tourner nos recherches. Cependant, habiter à coté du CERN, ce voisin que l'on considérait de façon presque indifférente, n'était pas un aspect à négliger: au contraire, c'était une richesse que nous avions la possibilité d'exploiter.

On pourrait penser que tout est parti du hasard... La retrouvaille d'un vieux détecteur de muons inutilisé et placé sur le site de l'expérience CMS nous a donné une opportunité et un défi ambitieux: restaurer ce détecteur et l'utiliser dans notre TPE pour en savoir plus sur les rayons cosmiques.

Il faut savoir aussi que c'est l'étude de ce rayonnement qui a permis la naissance et le développement de la physique des particules, grâce à laquelle les physiciens découvrent et étudient chaque jour d'autres particules inconnues: cela a été le cas pour le muon, le positron (la première particule d'antimatière découverte) et pour d'autres encore, dans une époque où les accélérateurs n'étaient pas aussi performants qu'aujourd'hui!

Nous avons organisé notre travail pour répondre à cette question:

Comment détecter le flux de muons, particules issues du rayonnement cosmique, arrivant sur Terre?

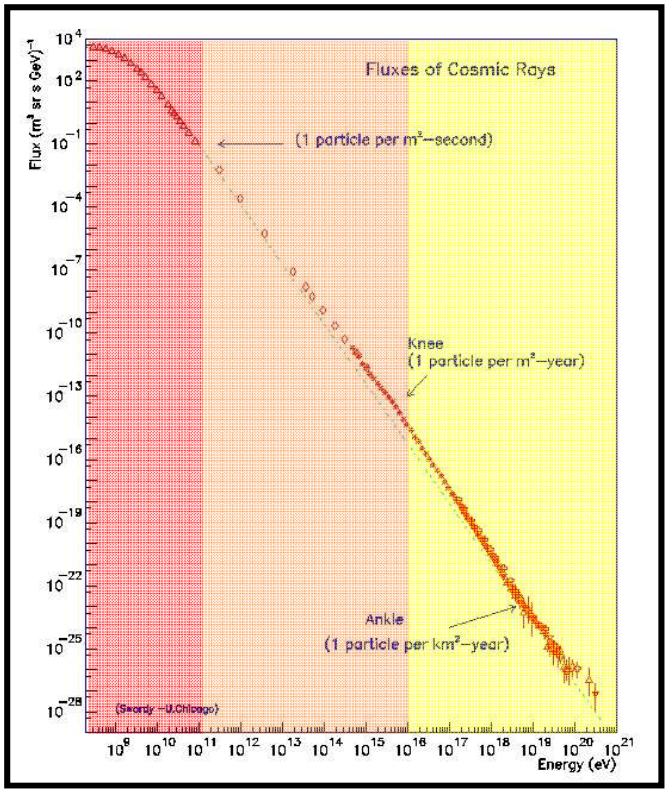
Sommaire:

1) Que sont les rayons cosmiques?

Avant de commencer toute expérience, il faut d'abord comprendre ce à quoi on a affaire: nous avons donc fait des recherches pour en savoir plus sur les rayons cosmiques. Notamment, nous avons retracé les étapes qui ont permis leur découverte, nous avons cherché d'où ces rayons viennent et de quoi ils sont composés, pour finalement se concentrer sur l'étude d'une particule très utilisée pour les mettre en évidence: le muon.

Les rayons cosmiques (RC) sont des radiations à très haute énergie provenant pour la plupart **d'en dehors du système solaire**. Leur étude intrigue beaucoup les atrophysiciens, puisque ces rayons présentent encore des **mystères**: en effet, ces chercheurs ne savent pas avec exactitude quelles sont leurs origines, ni pourquoi ils sont si énergétiques. De plus, ces rayons représentent une grande opportunité pour eux, puisqu'ils leur permettraient de voir l'univers avec **un autre messager que la lumière** (la plupart des études sont faites aujourd'hui avec elle, en effet) et ainsi ils pourraient étudier l'univers sous un autre angle pour mieux le comprendre.

Schéma du flux de RC en fonction de leur énergie



Plus les RC sont énergétiques, plus leur flux diminue et plus leur origine est lointaine.

Les RC peuvent être divisés en 3 catégories selon leur énergie:

- RC de basse énergie (entre 10^8 et 10^{11} eV) → une particule par cm^2 de surface terrestre et par seconde (partie rouge)
- RC de moyenne énergie (entre 10^{11} et 10^{18} eV) → une particule par m^2 de surface terrestre et par seconde (partie orange)
- RC de très haute énergie (entre 10^{18} et 10^{21} eV) → 2 particules par km^2 de surface terrestre et par siècle (partie jaune). Cette énergie correspond à celle d'une balle de tennis frappée à environ 100 km/h, mais elle est concentrée sur un seul proton!

Les premiers sont généralement produits par le **Soleil** ou par d'autres étoiles proches du système solaire. Les deuxièmes proviennent de notre galaxie entière, en particulier des **supernovae**, c'est-à-dire de l'explosion d'étoiles dans l'univers suite à l'effondrement de leur coeur. Cependant, pour ce qui regarde les RC très énergétiques, nous ne savons pas exactement quelles sont leurs sources et il y a plusieurs raisons à cela.

Premièrement, les RC sont essentiellement composés de **particules chargées** qui, depuis leur source, sont déviées de nombreuses fois avant d'arriver sur Terre, puisque l'Univers est parsemé de champs magnétiques. Il est donc difficile de retrouver leur source simplement en regardant leur direction d'arrivée. Quelques chercheurs proposent alors de s'intéresser à l'étude des neutrinos et des rayons γ (gamma, c'est-à-dire des photons à très haute énergie) produits lors de la collision des rayons cosmiques avec les photons de l'univers: en effet ils n'ont pas de charge et ne sont donc pas déviés, pouvant nous donner des informations précieuses sur la trajectoire des rayons cosmiques. Toutefois, beaucoup de recherches ont encore lieu sur comment les détecter, puisque les moyens de détection de ces particules ne sont pas encore complètement mis en place.

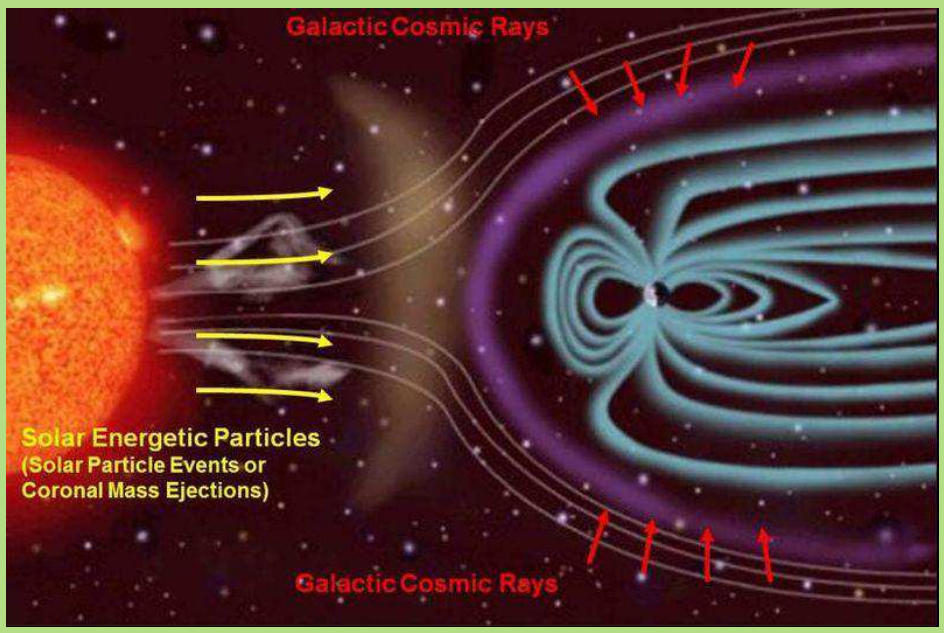
Deuxièmement, les astrophysiciens supposent que les RC proviennent de **phénomènes très violents de l'univers**: supernovae, pulsars (une partie de la matière qu'une supernova a éjecté), radiogalaxies, amas de galaxies, sursauts gamma. Ces phénomènes sont, malheureusement, peu connus et très exotiques, donc les physiciens n'ont pas assez d'informations pour les étudier et déterminer ainsi comment ils produisent des RC.

Troisièmement, **le flux de RC** arrivant sur Terre diminue au fur et à mesure que leur énergie augmente. Du coup, dans le cas des RC de ultra haute energie, il en arrive environ 2 particules par kilomètre carré et par siècle. Cependant, grâce à de grands observatoires tel que l'Observatoire Pierre Auger en Argentine, qui recouvre une surface de 3000 km^2, les physiciens arrivent à détecter environ **2 particules par mois**, une quantité qui est toutefois encore trop petite pour étudier ces rayons très énergétiques de façon plus approfondie.

Box d'approfondissement: les RC ne sont pas à confondre avec le vent solaire (vent stellaire pour les étoiles en général)

Le vent stellaire est un phénomène qui se vérifie chez toutes les étoiles. C'est un flux de plasma (phase de la matière constituée essentiellement de particules chargées, d'électrons et de ions) éjecté de l'atmosphère de l'étoile: c'est donc une perte de sa matière. Ce flux d'éjection de plasma varie en vitesse et en température au cours du temps, ceci en fonction de l'activité de l'étoile. Chez le Soleil, la composition du plasma est identique à celle de la couronne solaire : 73 % d'hydrogène et 25 % d'hélium.
Chaque seconde, le Soleil perd environ 1×10^9 kg (soit un million de tonnes) de matière sous forme de vent solaire.

Grâce aux études du physicien autrichien Victor Franz Hess, nous savons qu'en plus du vent solaire, le Soleil émet des RC, qui sont déviés par le champ magnétique terrestre.



1) a - Histoire

La recherche sur les RC débute en 1900, lorsque le physicien Charles Thomson Rees Wilson découvre, suite à ses études, que **l'atmosphère terrestre est continuellement ionisée** : il crée alors une méthode de détection capable de révéler ces radiations ionisantes, **la chambre à brouillard**, pour laquelle il reçoit le prix Nobel en 1923.

Peu de temps après, entre 1901 et 1903, des chercheurs s'aperçoivent que même si un électroscope était isolé, il détectait quand même du signal : il supposèrent alors que la radiation détectée devait être **très pénétrante**.

Toutefois, le temps des grandes découvertes débute en 1912, lorsque les études contemporaines et indépendantes des physiciens **Victor Franz Hess** (qui reçut le Nobel pour la Physique en 1936) et Domenico Pacini démontrent que ces radiations étaient extraterrestres. En effet, Hess s'est servi d'un ballon aérostatique sur lequel il avait placé un oscilloscope pour mesurer l'évolution de la quantité de radiations en fonction de l'altitude: il a remarqué qu'elles augmentaient au fur et à mesure que l'on montait. En même temps, Pacini a fait des études sur les lacs de Livourne et de Bracciano, remarquant que l'intensité des radiations ionisantes diminuait au fur et à mesure que l'on descendait en profondeur. Ces deux expériences ont donc prouvé que ces radiations n'étaient pas produites par la Terre elle-même (comme des scientifiques avaient pensé), mais qu'au contraire, **elles provenaient de l'espace**.

Hess démontre aussi que le Soleil était une des principales sources de ces rayons, qui seront appelés « **rayons cosmiques** » par Robert Andrews Millikan en 1929. Ce physicien émet l'hypothèse qu'ils étaient formés essentiellement de rayons γ (gamma, donc de photons, particules de l'énergie électromagnétique, dont fait partie la lumière), hypothèse aussitôt rejetée par les études de Arthur Compton. Le physicien démontre, en effet, que le rayonnement était **variable** en fonction de la latitude où il était mesuré et qu'il était composé de **particules chargées**, influencées par le champ magnétique terrestre: ceci prouvait donc que les RC ne pouvaient pas être formés essentiellement de photons, particules qui ne possèdent pas de charge. Compton reçut le Nobel pour ses études en 1927.

En 1929, pendant qu'il était en train d'étudier les rayons cosmiques grâce à la chambre à brouillard de Wilson, Dmitri Skolbeltsyn remarque une particules qui se comportait comme un électron, mais qui déviait dans la direction opposée face à un champ magnétique: c'est la découverte du **positron**, la première particule d'antimatière. Quelques années plus tard, sont découvertes d'autres particules, telles que les muons, les pions, les kaons…

Vers 1930, Pierre Auger découvre que les particules des rayons cosmiques entrant en contact avec l'atmosphère terrestre, déclenchaient des **gerbes atmosphériques**.

Ensuite, dans les années quarante, le physicien Enrico Fermi émet l'hypothèse que les rayons cosmiques pouvaient être **accélérés dans les restes d'une supernova**: elle est démontrée en 2013 grâce au Large Area Telescope, au bord du téléscope spatial Fermi (NASA).

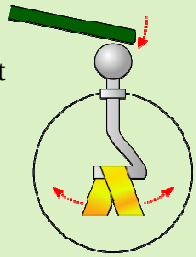
C'est vers 1950 qu'apparaissent les premiers **accélérateurs de particules** à haute énergie, qui se basaient beaucoup sur **l'étude des rayons cosmiques pour découvrir et étudier d'autres particules**, issues de ces rayons. Entre autres, le CERN (1954) ajoute dans sa «Liste de Recherche» les rayons cosmiques.

Vers 1960, Giuseppe Cocconi, scientifique au CERN, démontre que **les rayons les plus énergétiques étaient extragalactiques** et que leur émission s'accompagnait de rayons γ.

Aujourd'hui, les rayons cosmiques n'ont pas fini de nous raconter leurs secrets sur le monde qui nous entoure. D'autres particules restent à découvrir, beaucoup de questions sont encore sans réponse et les chercheurs sont assoiffés de savoir:

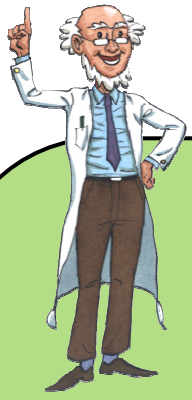
la recherche continue!

Box d'approfondissement: l'électroscope



Il fut inventé par l'abbé Jean-Antoine Nollet vers 1750, et il servait à **détecter et quantifier les charges électriques**. Il fut beaucoup utilisé, surtout au début du XXè siècle, lorsque la découverte des rayons X et de la radioactivité mit en mouvement la communauté scientifique, mais il n'est plus utilisé aujourd'hui.

Le principe est simple: lorsqu'on touche l'électrode avec un corps chargé, une partie de sa charge va vers les 2 feuilles d'or. Du coup, elle entre dans les deux feuilles qui, possédant la même charge, se repoussent.
En ragardant les feuilles d'or on peut:
1) déterminer si un corps est chargé (dans ce cas les feuilles se repoussent)
2) déterminer la charge (positive au négative) d'un corps inconnu. Pour cela, on touche l'électroscope avec un corps dont la charge est connue et ensuite on fait la même chose avec le corps que l'on veut étudier. Si les deux feuilles s'écartent encore plus, alors le deuxième corps a le même charge que le premier; si au contraire les feuilles se rapprochent, alors il est de charge opposée.



Et que peut-on dire sur l'hypothèse de Fermi, validée en 2013? Jetez un coup d'oeil dans le box!

Les **rayons cosmiques** sont des particules chargées de très haute énergie et depuis toujours les astrophysiciens se sont demandés - scientifiquement - pouquoi ces particules sont si **énergétiques**.
Il y a plus de 70 ans, le physicien Enrico Fermi a proposé en premier un mécanisme capable d'expliquer l'énorme énergie accumulée par les rayons cosmiques, qui traversent constamment notre galaxie. Selon sa théorie, élaborée en 1949, les RC devaient **s'accélérer** dans des nuages de gaz magnétisé et en mouvement. Malheureusement, Fermi mourut avant de pouvoir démontrer son idée, qui pendant beaucoup d'années s'est révélée être le seul procédé praticable, mais inexplicable.

Pendant les 50 dernières années, les astrophysiciens ont trouvé dans les supernovaes les sites les mieux adaptés pour créer les conditions prévues par Fermi, et ceci grâce au développement de puissants téléscopes. En 2008, pour vérifier l'hypothèse de Fermi, le **Large Area Telescope (LAT)**, un détecteur de rayons gamma à haute énergie, a été ainsi construit et placé sur le satellite de la NASA nommé Fermi, en l'honneur du célèbre physicien.

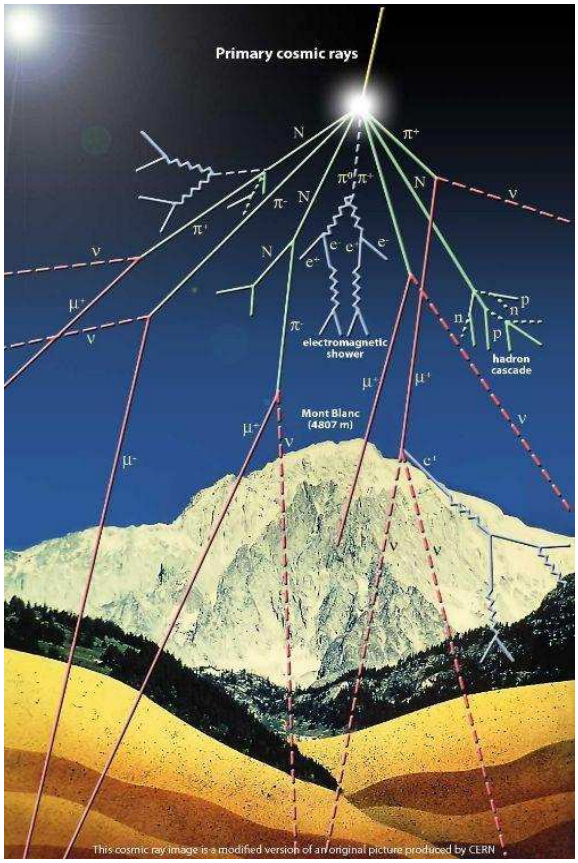
Ainsi, on a pu observer une intense émission de **rayons gamma** (c'est-à-dire des photons à très haute énergie) dans plusieurs restes de **supernovaes**. En effet, il a été vérifié que dans la matière éjectée par les supernovaes pendant leur explosion, il se forme un champ magnétique qui emprisonne les **protons**. Ceux-si se déplacent de façon aléatoire le long du front d'onde de l'explosion et ils tentent à plusieurs reprises de le dépasser. Toutefois, le champ magnétique les repousse en arrière, jusqu'à ce que les protons aient accumulée tellement d'énergie qu'ils peuvent se libérer de lui. Ces protons ainsi libérés se propagent dans l'espace, mais ils entrent en collision avec des nuages de gaz émis par les supernovaes. La collision d'un proton avec un atome de ce gaz donne vie à un pion neutre, une particule instable qui se dèsintègre aussitôt en 2 rayons gamma.

Les observations du satellite Fermi ont mis en évidence ce mécanisme dans 4 supernovaes d'âge différent: des extrêmement jeunes (donc de quelques centaines d'années) à d'autres qui sont datées de plusieurs milliers d'années.

Ainsi, à plus de 70 ans de distance, l'hypothèse de Enrico Fermi a enfin trouvé une confimation expérimentale!

1) b - Composition

Les RC se divisent en 2 catégories:

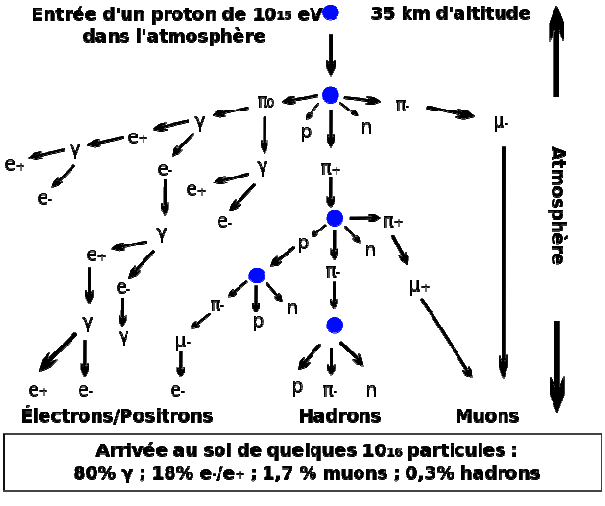


- **RC primaires:**
  Ce sont tous les rayons au-delà de l'atmosphère terrestre, constitués de environ:
  - 90 % de protons,
  - 9% de particules α (en pratique, des noyaux d'hélium)
  - 1% de électrons, neutrinos, photons et antimatière (positrons)

  Lorsque les particules des RC primaires arrivent dans l'atmosphère terrestre, elles entrent en collision avec les molécules qui la composent (comme l'oxygène ou l'azote) et perdent leur énergie, créant une "pluie" de particules secondaires.

- **RC secondaires:**
  Celles-ci peuvent à leur tour collisionner avec d'autres molécules et donner naissance à d'autres particules. Cette cascade continue jusqu'à ce que les particules finissent leur énergie ou soient absorbées par l'atmosphère. Cette "pluie" de particules, issues de la collision d'une particule des RC primaires avec l'atmosphère, forme une gerbe atmosphérique et l'ensemble des gerbes forme les RC secondaires.
  Ils sont composés de:
  - 30% de composante "molle"(surtout pions, kaons, neutrons, neutrinos, électrons, rayons γ) qui pénètre peu dans les milieux denses
  - 70% de composante "dure" (muons) qui arrive au niveau de la mer et pénètre plus dans les milieux denses

N.B. Les RC les plus énergétiques génèrent de grandes gerbes atmosphériques de plus de 10 milliards de particules secondaires, qui peuvent être détectées par des détecteurs de particules lorsqu'elles se propagent sur des aires de environ 20 km^2 à la surface de la Terre.



Une gerbe atmosphérique:

D'un seul proton (la particule la plus abondante chez les RC) émergent une grande quantité de particules, mais pendant que les électrons et les pions se désintègrent à haute altitude, les muons avec leurs neutrinos correspondants arrivent au sol.

π = pions
e- = électrons
μ= muons
n= neutron ,
e+ = positrons
γ= photons
p= protons

En conclusion, après toutes ces années de recherche, aujourd'hui les astrophysiciens savent que:

- les rayons cosmiques sont une radiation extraterrestre très pénétrante,
- leur flux est variable en fonction de la latitude où ils sont mesurés,
- ils sont composés de particules chargées (donc influencées par le champ magnétique de l'univers),
- ils se décomposent en gerbes atmosphériques au contact de l'atmosphère terrestre,
- ils sont accompagnés de rayons γ dans leurs émissions,
- ils sont plus énergétiques en fonction de leur éloignement.

1) c - Les muons et les particules élémentaires

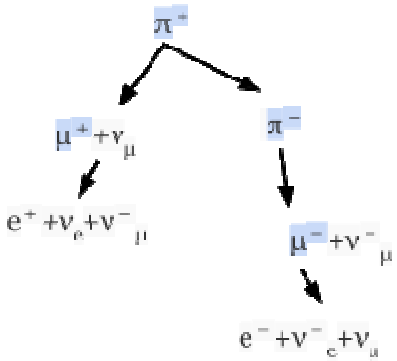
C'est un peu le hasard qui a déterminé la découverte des muons (comme souvent, en physique). En effet, en 1936, Carl David Anderson et Seth Neddermeyer étaient en train d'étudier les RC grâce à la chambre à brouillard de Wilson. Tout d'un coup, ils s'aperçurent que, face à un champ magnétique, quelques particules **déviaient** de façon différente par rapport à celles que l'on connaissait. En effet, si elles déviaient du même côté que l'électron, elles avaient une courbature mineure par rapport à lui, mais supérieure à celle du proton. Les deux physiciens supposèrent donc que ces particules devaient avoir la même charge que l'électron et une masse entre la sienne et celle du proton. Ils décidèrent donc d'appeler cette particule mystérieuse ***mésotron*** (car en grec ancien le préfixe *meso-* signifie "intermédiaire").

Plus les années passaient, plus d'autres particules à masse intermédiaire venaient découvertes et furent appelées *mesons*. Pour les différencier d'eux, le mésotron fut donc appelé **meson μ** ("mu").

Toutefois, les physiciens se rendirent compte aussitôt que le meson μ était une particule... spéciale. En effet, il se comportait différemment par rapport aux autre mesons: par exemple, lorsqu'il se désintégrait, il donnait naissance à un neutrino et un antineutrino, contrairement aux autres mesons qui n'en généraient qu'un seul. Suite à des recherches, on comprit que si une particule était un meson, alors elle devait posséder **2 quarks**. Ainsi, en 1945, fut organisée une expérience, dans laquelle on découvrit que les mesons μ ne possédaient pas de quarks: ils nétaient pas sujets à l'interaction forte.

Ainsi, à la fin on comprit que ces particules ne pouvaient plus s'appeler mesons μ: elles furent appelées ***muons*** et vinrent composer la catégorie des particules élémentaires et plus précisément la faille des **leptons** auxquels appartient l'électron.

Les muons sont généralement produits de la désintégration de mesons π ("pi"): les pions, qui est illustrée dans ce schéma. Le muon est une particule instable qui se désintègre généralement en ν_μ, $\bar{\nu}_\mu$ et ν_e, $\bar{\nu}_e$ (sont respectivement les neutrinos et les antineutrinos du muon et de l'électron): c'est pour cela que les muons ont aussi très importants pour l'étude des neutrinos.



déviaient: On peut reconnaître et identifier une particule par rapport à sa déviation dans un champ électrique. Par exemple, une particule de charge positive sera déviée par le champ électrique positif avec une courbature qui dépend de sa masse (plus sa masse est petite, plus elle sera déviée). C'est la méthode qui est utilisée au CERN, par exemple, pour étudier les particules.

leptons: en physique des particules, un lepton est une particule élémentaire de spin 1/2 qui n'est pas sensible à l'interaction forte (c'est la force qui, chez un atome, fait en sorte que les protons, qui ont une même charge et qui devraient donc se repousser, soient "collés" ensemble et que les quarks qu'ils contiennent soient liés entre eux).

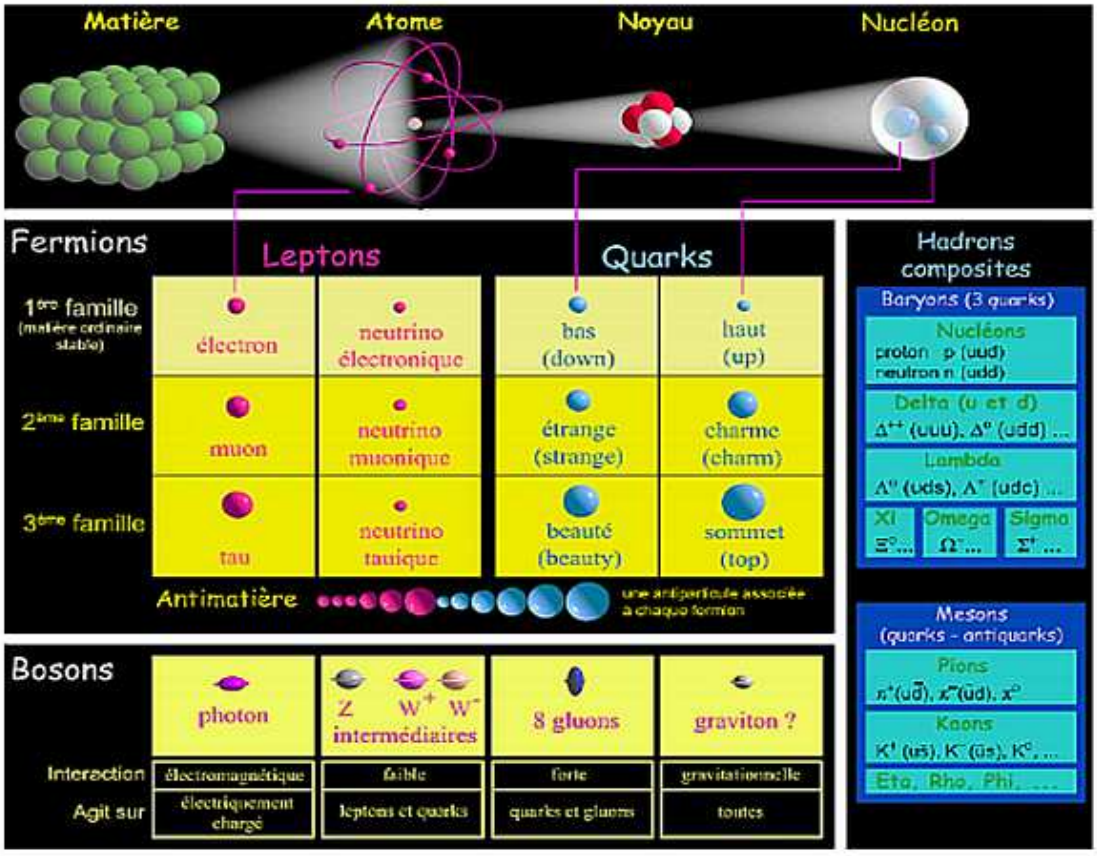
Lorsque les particules des RC primaires entrent en contact avec l'atmosphère, elles interagissent avec les différentes particules qui la composent pour en générer d'autres, en particulier des pions. Ceux-ci se désintègrent en émettant des muons, qui contrairement aux pions n'interagissent pas fortement avec la matière et peuvent voyager au travers de l'atmosphère pour arriver au niveau de la mer. Il en arrive environ un par seconde dans un volume à peu près égal à la tête d'une personne.

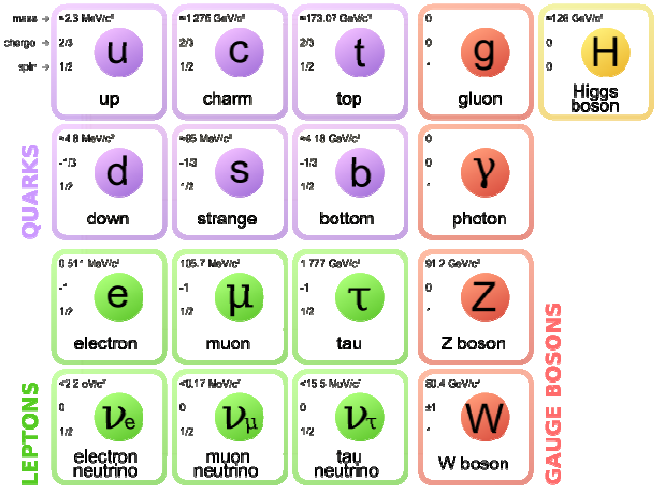
Pour détecter les RC à notre échelle de lycée, il faut que nous puissions voir les particules qui atteignent le niveau de la mer, et les plus facilement exploitables sont justement les **muons**.

Les particules élémentaires:

En physique, les muons font partie des **particules élémentaires**, c'est à dire les «briques» fondamentales et indivisibles de la matière, à l'intérieur desquelles - pour le moment - il n'y a pas d'autres particules plus simples. Les particules élémentaires sont également responsables des interactions entre matière et radiations, dont elles sont les éléments fondamentaux.

La physique des particules élémentaires s'intéresse à la **découverte** des **éléments fondamentaux** qui constituent tout ce qui nous entoure et à **l'étude** de leur comportement: pour cela, des physiciens des particules détectent et quantifient chaque jour électrons, pions, muons… grâce à leurs détecteurs. Toutefois, la physiques des particules ne s'utilise pas que dans le domaine de la physique: par exemple, le scanner pour tomographie à émission de positrons (**PET scan**) est souvent utilisés en médecine nucléaire, notamment pour détecter des cancers et contrôler les fonctions des organes internes.





Souvent, le muon est considéré comme un éléctron 207 fois plus lourd, car comme lui il a une charge négative, mais une masse 207 fois plus importante:

En effet, d'après la théorie de la relativité de Einstein où $E= mc^2$: $m= E/c^2$

énergie de masse d'un e- : 0,511 MeV/c^2
énergie de masse d'un muon: 106 MeV/c^2

106 / 0,511 = 207,43

Box d'approfondissement: comment les muons font-ils à arriver sur la surface de la Terre en voyageant presque à la vitesse de la lumière, si leur durée de vie est d'environ 2 microsecondes?

En 2 μs, en effet, il est impossible de parcourir une distance de 35 km (l'épaisseur de l'atmosphère terrestre), même à la vitesse de la lumière, comme le démontre ce calcul:

v=d/t
d= t x v avec t en mètres et v en mètres par seconde
$d= 2x10^{-6} \times 3{,}00x10^{8}$
d= 600 m

Ce qui arrive, c'est que les muons appliquent la théorie de la **relativité**: du coup, l'espace-temps se dilate et ils arrivent au sol avant de se désintégrer. Voici ce qui se passe:

Imaginons d'avoir une pendule ayant un intervalle de 2 μs.
De notre point de vue, nous savons qu'en 2 μs, un muon voyageant à la vitesse de la lumière ne peut parcourir que 600 mètres.
C'est donc trop peu, si on considère que l'atmosphère terrestre fait environ 35 kilomètres. Il faudrait ainsi que la pendule oscille environ 25 fois avant que le muon atteigne le niveau de la mer et entretemps, il se passerait environ 50 μs (15 000m /2μs x 600m = 50 μs).
Toutefois, il ne faut pas oublier que le muon voyage à la vitesse de la lumière, c'est-à-dire environ 300 000 km/s. Or, la relativité nous enseigne que si un corps voyage à la vitesse de la lumière, alors **l'espace-temps, pour lui, se déforme**.
Du coup, pour le muon, le temps ralentit et l'espace se rétrécit de sorte qu'il peut atteindre la surface terrestre avant de se désintégrer; alors que nous, paradoxalement, nous verrons la pendule osciller environ 25 fois avant de "voir" le muon atterrir.

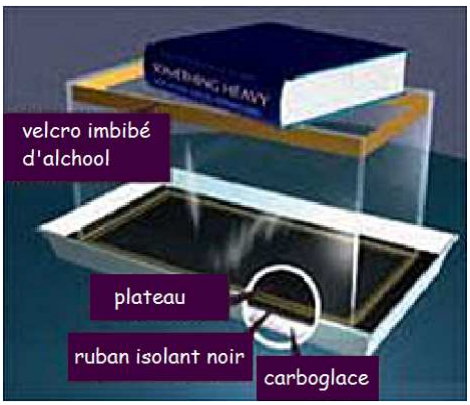
2) Comment détecter les muons sur Terre?

Maintenant que nous avons découvert les rayons cosmiques, voyons quelles sont les différentes façon de les détecter, notamment grâce aux muons:

- détecter les rayons γ émis lorsque les rayons cosmiques traversent l'atmosphère (mais il faut des télescopes spéciaux et très coûteux)
- détecter les particules issues des gerbes atmosphériques à l'aide de:
    - ballons-sonde à haute altitude (comme a fait Hess)
    - la chambre à brouillard
    - la chambre à étincelle
    - un détecteur de muons à scintillation

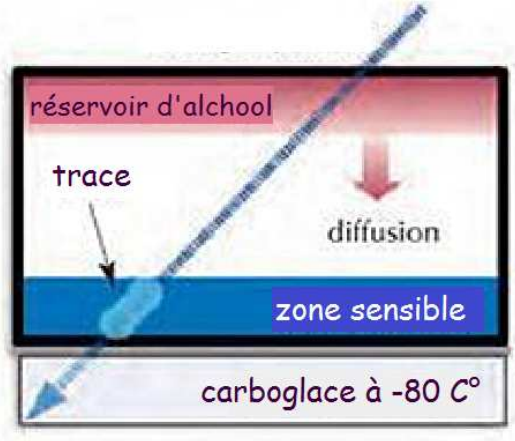
Avant de parler du détecteur de muons que nous avions disposition, nous allons d'abord analyser le principe de la chambre à brouillard et de la chambre à étincelle.

a) La chambre à brouillard



La chambre à brouillard est le premier appareil imaginé par les physiciens pour mettre en évidence les RC.
Le principe de la chambre est de détecter les particules issues des RC atteignant la surface de la Terre (notamment des muons) en voyant les effets qu'elles ont sur cette vapeur d'alcool instable, qui réagit donc facilement. En effet, comme on ne peut voir aucune particule directement, on cherche à observer leurs effets.

Réalisation:

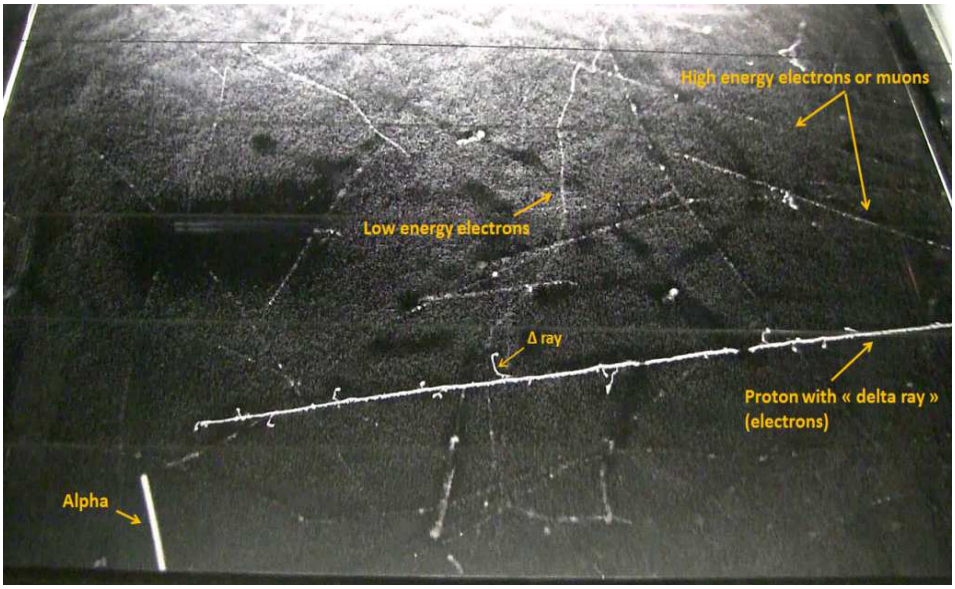


Des rubans de velcro imbibés d'alcool isopropilique sont collés le long des côtés au fond d'un récipient en plastique ou en verre aux paroies droites, un aquarium, par exemple. Sur la partie ouverte de ce récipient, on colle une plaque d'aluminium recouverte d'un ruban isolant noir, de sorte que le récipient soit complètement étanche.
Sur un plateau, on met de la neige carbonique (du dioxyde de carbone solide restant à la température constante d'environ -80 °C pendant sa sublimation) et, au-dessus, on place l'aquarium de sorte que l'aluminium isolé soit au contact avec la neige carbonique.
Enfin, on pose un gros livre sur la partie supérieure de l'aquarium, pour que tout reste stable.

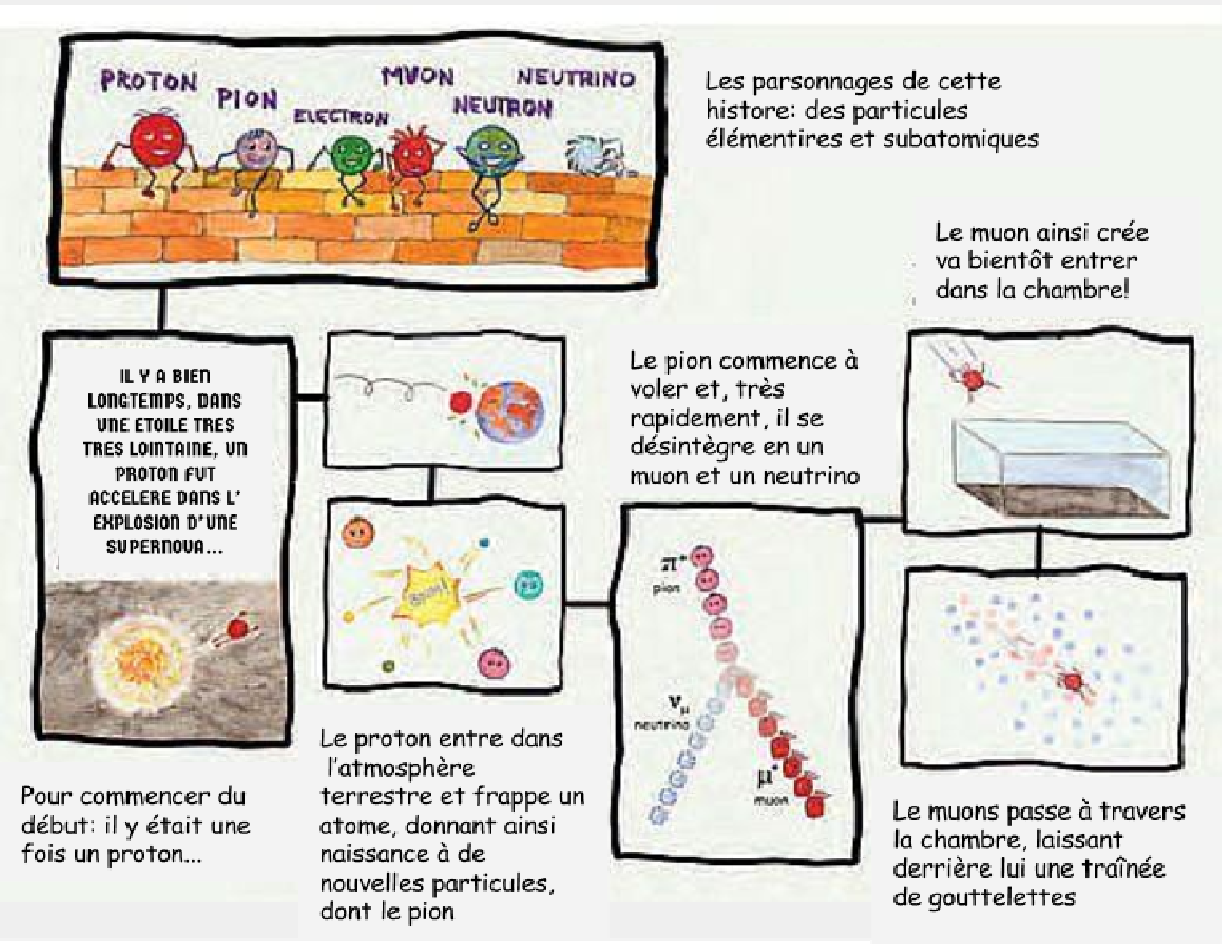
Principe:

La chambre à brouillard est un récipient étanche rempli d'une atmosphère mixte d'air et de vapeur d'alcool. En pratique, l'alcool liquide s'évapore des rubans de velcro qu'il imbibe et se diffuse à travers l'air de la chambre, mais lorsqu'il rencontre la zone refroidie par la neige carbonique, elle créé une zone sensible, saturée de vapeur d'alcool.
Cette couche sensible est instable: la vapeur d'alcool très froide est trop lourde pour son équilibre.

Cependant, le passage d'une particule chargée ayant suffisamment d'énergie pour **ioniser** les atomes sur son passage (la ionisation est le phénomène selon lequel un atome électriquement neutre perd ou gagne un ou plusieurs électrons, ce qui le charge électriquement le faisant devenir un ion) fait en sorte que la vapeur d'alcool se condense.
Ainsi, dans leur passage, ces particules ionisent le milieu et laissent une traînée de particules que l'on voit dans cette sorte de "brouillard" à travers des gouttelettes: les traces courtes et épaisses sont les particules alpha et les longues et fines sont les partcules béta. On peut voir ces traces si on place la chambre dans une salle sombre et on l'éclaire avec des lampes de poche.



Cette BD que nous avons tiré et traduit du site de Science in School, permet de mieux voir ce qui se passe:



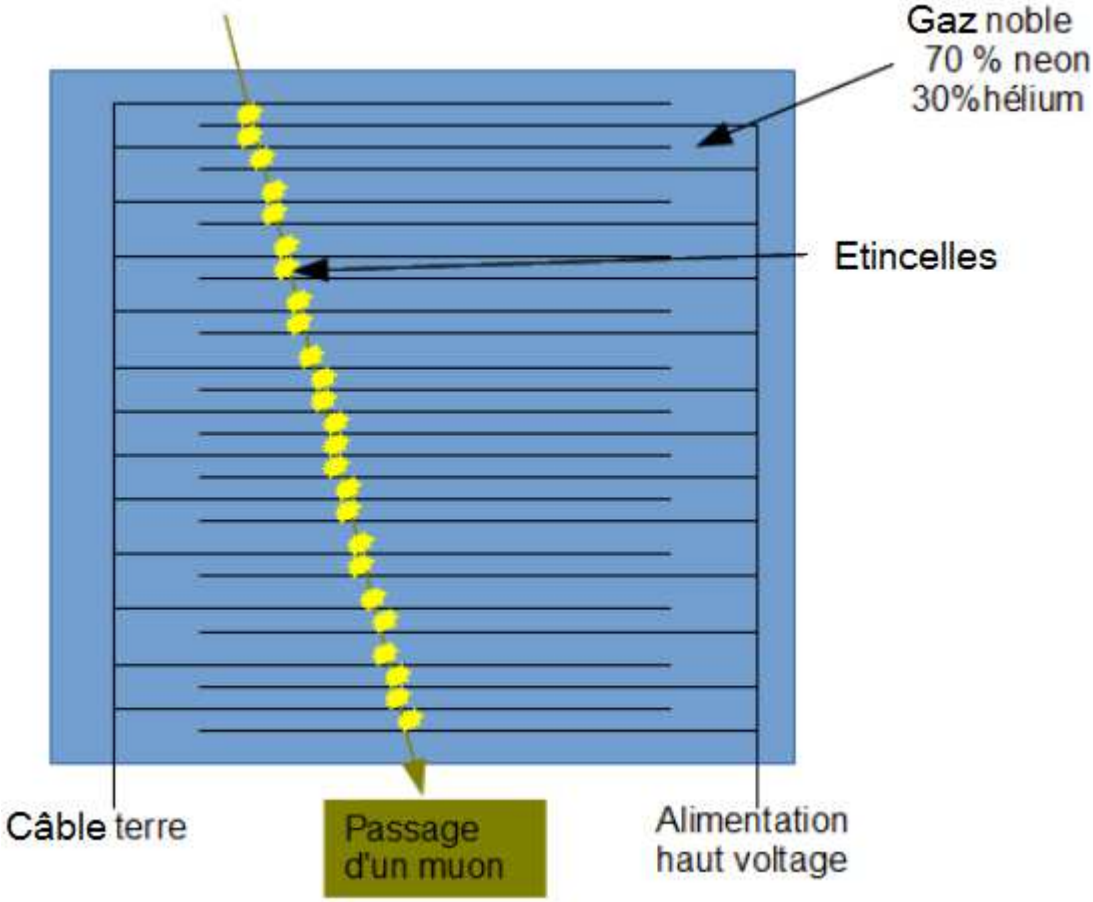
b) La chambre à étincelle

La "chambre à étincelles " est un dispositif qui permet de se rendre compte de l'existence des particules cosmiques portant une charge électrique. Cet appareillage les détecte et permet la visualisation de leur passage grâce à des étincelles produites le long de leur chemin.

Dans la chambre à étincelles des feuilles métalliques conductrices sont placées parallèlement dans une enceinte remplie d'un mélange de gaz nobles, néon et hélium en général. Une plaque sur deux est mise à haute tension. Lorsqu'une particule chargée traverse la chambre, elle heurte les atomes de gaz et leur arrache des électrons : elle « ionise » le gaz. Des étincelles éclatent alors le long de la trajectoire suivie par la particule, le gaz étant devenu momentanément conducteur à cet endroit.

La ionisation et le fait de retirer ou d'ajouter un ou plusieurs électrons à un atome électriquement neutre le faisant devenir un ion et le chargeant électriquement. Si il a gagné des électrons, alors in devient de charge négative, si il a au contraire perdu des électrons, alors il devient de charge positive.

Des étincelles peuvent se développer là où le gaz a été ionisé par le passage de la particule chargée. Les étincelles sont alignées le long de la trace et visualisent la trajectoire de la particule, en léger différé.



c) La détection grâce à la fluorescence

Un détecteur à scintillation est un instrument composé d'un matériau qui émet de la lumière suite à un dépôt d'énergie due à un rayonnement.
Les scintillateurs peuvent être soit organiques, soit non-organiques ce qui leur confère certains caractères spécifiques.
Les scintillateurs organiques peuvent se trouver sous la forme plastique, liquide ou cristalline.
Leur mécanisme de fluorescence est associé aux états excités des molécules. La lumière émise couvre généralement un spectre large dans les UV et le visible.
Les scintillateurs plastiques ou liquides sont constitués de deux composants fondamentaux: un solvant et un (ou plusieurs) soluté(s). Le solvant absorbe l'énergie et son excitation est transmise au(x) solvant(s) qui émet(ent) de la lumière.
Les scintillateurs plastique sont obtenus par polymérisation. La présence d'impurtés pourrais fausser son bon fonctionnement.

Une propriété remarquable des scintillateurs organique est leur réponse très rapide (de l'ordre de quelques nanosecondes), ce qui permet de les utiliser dans notre problématique où nous mesurons des temps de coïncidences.

d) Notre détecteur

d-1 Origine du matériel

Lors de la période de choix de notre sujet de TPE, nous avons appris la présence d'un détecteur de muons sur le site de l'expérience du CERN CMS (Cessy, point P5).

Le détecteur en question est un appareil de type "CRIL" (anglais "Cosmic Ray Interactive Laboratory", "Laboratoire Interactif de Rayons Cosmiques") développé à l'université de Notre-Dame aux Etats-Unis. Il s'agit d'un détecteur de muons dont la construction est basée sur les composants du projet Quarknet mené par le laboratoire Fermilab aux Etat-Unis, qui vise à donner accès aux élèves d'écoles américaines à des activités concernant la physique de particules à haute énergie.

Ce détecteur était utilisé comme pièce d'exposition au CERN, mais il n'était plus en état de marche car l'ordinateur servant d'interface était manquant. Sinon, à part une antenne du module GPS intégré et l'ordinateur, tous les composants de l'appareil étaient présents. Nous avons alors décidé de le remettre en état et de faire nos expériences à l'aide de ce détecteur.

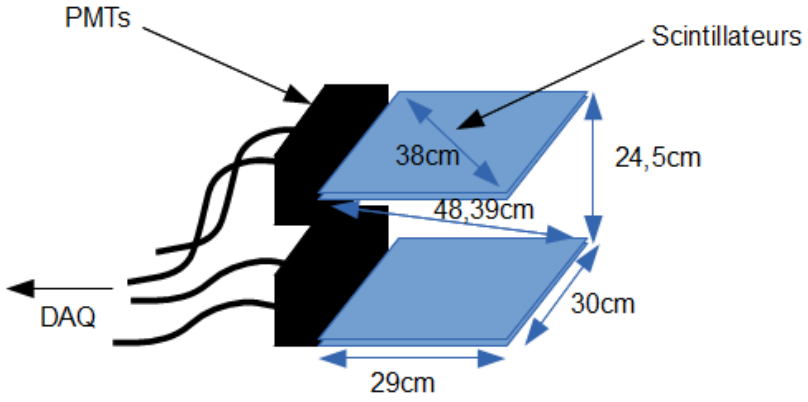
Ainsi, nous l'avons déplacé du site CMS au point CERN de Meyrin afin de pouvoir y accéder et y travailler plus facilement, le site CMS ayant des réstrictions sur la présence de personnes étrangères au service. Le détecteur a donc été installé dans un bureau, après avoir enlevé la vitrine de plexiglass dans laquelle il se trouvait pour l'exposition. Depuis, la pièce centrale du détecteur (les quatre scintillateurs avec les tubes photomultiplicateurs et le socle) est restée dans ce bureau.

d-2 Matériel et fonctionnement

Le détecteur utilisé tout au cours de l'expérience est basé sur le matériel mis à disposition des écoles par le projet Quarknet, ainsi que certains éléments qui y ont été ajoutés ensuite:

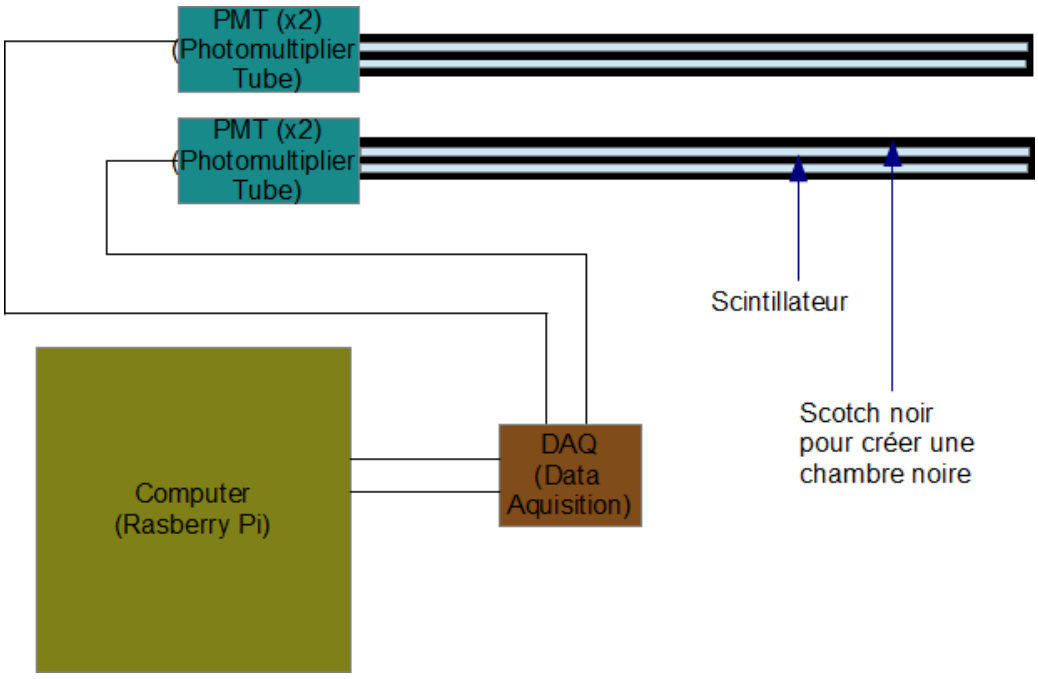
- 1 carte d'acquisition de données Quarknet série 6000, version du micrologiciel 1.01
- 2 modules de scintillateurs plastiques et photomultiplicateurs assemblés sur une structure métallique rotative, soit 4 photomultiplicateurs, 4 scintillateurs et 2 transformateurs à haute tension pour alimenter les photomultiplicateurs
- 1 module GPS avec une antenne
- 4 câbles coaxials avec connecteurs LEMO-00 50Ω et 4 adaptateurs BNC (câbles standards utilisés dans la physique des particules pour le transfert de signaux analogiques)
- 1 microordinateur Raspberry Pi, avec une carte SD et un câble Ethernet
- 4 blocs d'alimentation, soit 2 blocs d'alimentation pour les photomultiplicateurs, 1 bloc d'alimentation pour la carte d'acquisition de données, et un bloc d'alimentation pour le micro ordinateur Raspberry Pi

Dimensions du récepteur:



Épaisseur des blocs de 2 scintillateurs : 2,5 cm
Un scintillateur : 1,25 cm

Schéma technique de la composition du détecteur



Le detecteur :

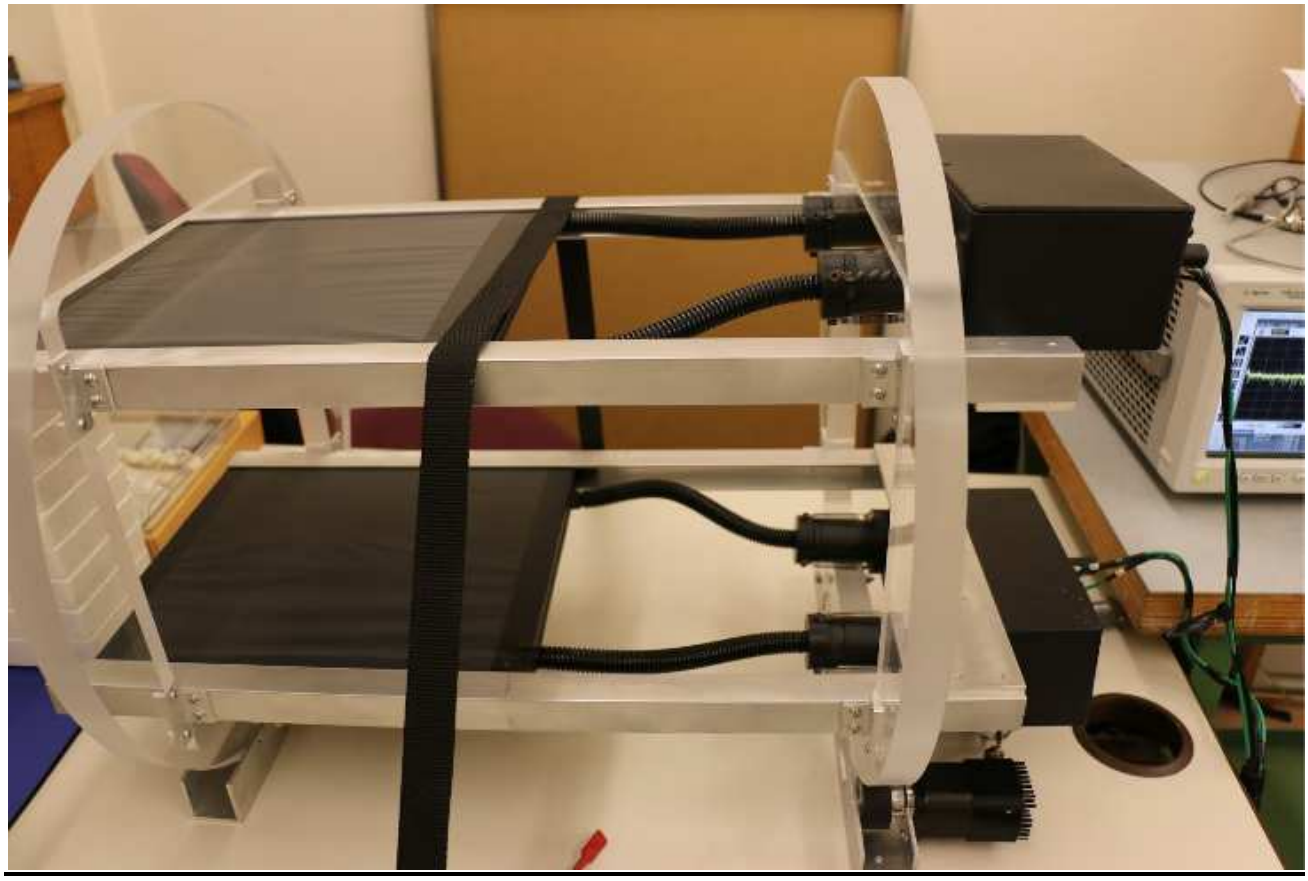


Figure 1: La partie réceptrice du détecteur

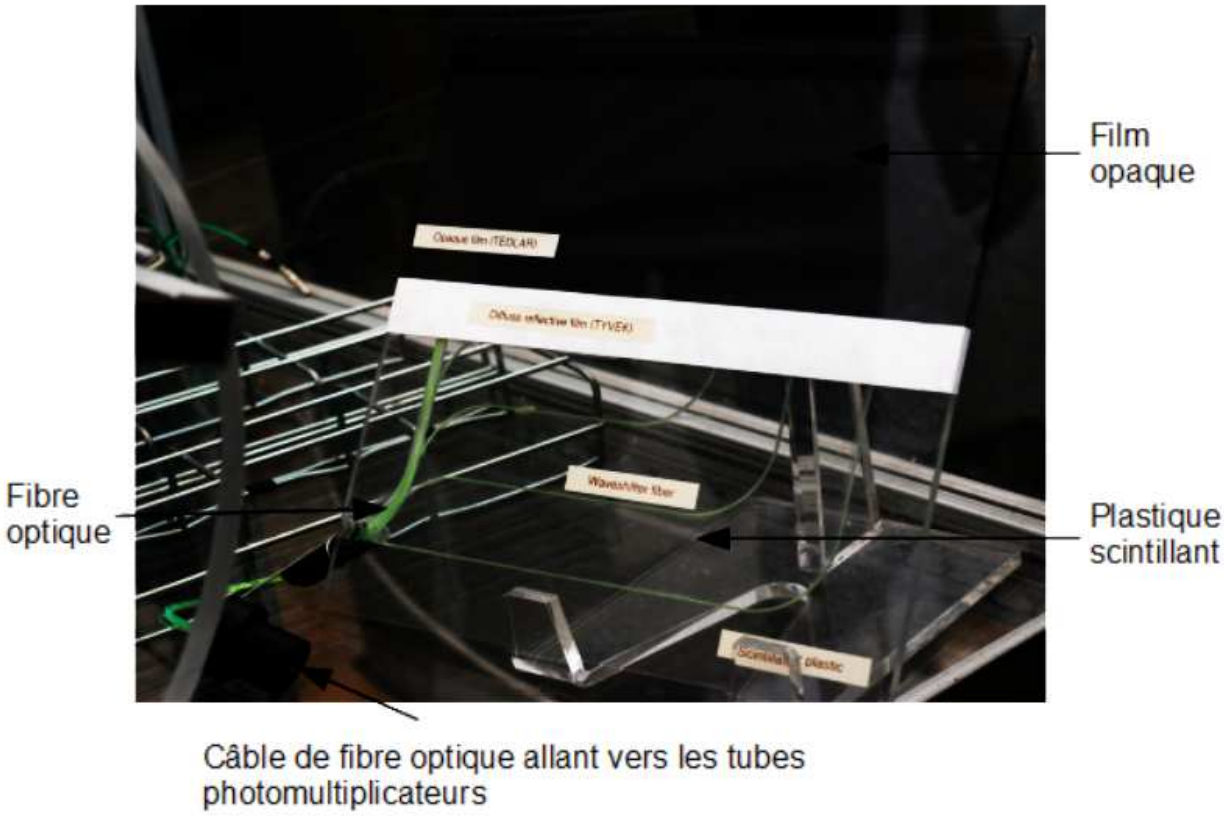


Figure 2: Le scintillateur

Le détecteur de muons utilisé dans cette expérience s'appuie sur un type particulier de fluorescence: la **scintillance d'un matériau**. Selon ce principe, une entité chimique (molécule ou atome) est excitée après absorption d'un rayonnement électromagnétique; **un ou plusieurs photons sont alors émis** lors de la désexcitation, afin que l'entité puisse revenir à son état fondamental.

Le détecteur est représenté sur la figure 1. Lors du passage d'un muon, le scintillateur plastique émet une impulsion lumineuse très faible; les photons émis sont ensuite redirigés dans un réseau de fibres optiques par réflexion; les fibres optiques du détecteur transforment la longueur d'onde d'entrée des photons en émettant un rayonnement du spectre visible de la lumière à la sortie, tout en acheminant la lumière dans les tubes photomultiplicateurs, qui transforment les photons en **signal éléctrique**.

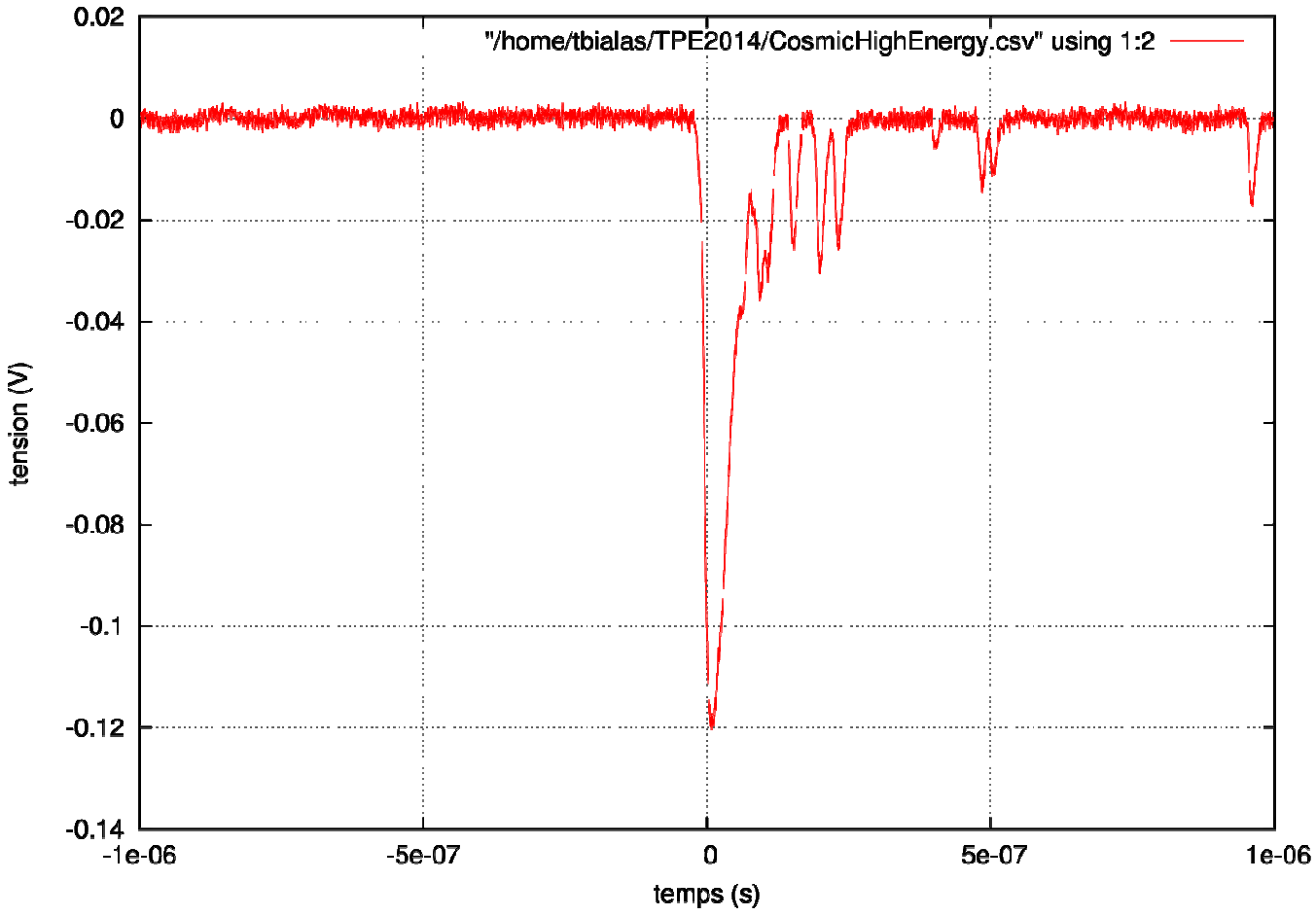


Figure 3: Signal analogique lors du passage d'un muon mesuré à l'aide d'un oscilloscope

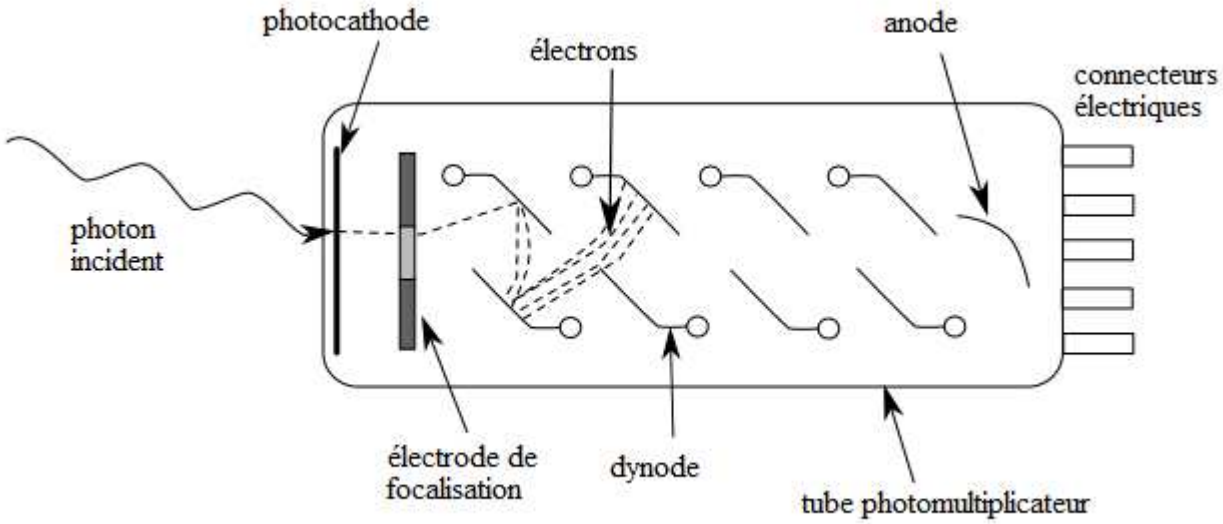


Figure 4: Le photomultiplicateur

Un photomultiplicateur est une chambre sous vide, sur laquelle on applique une tension électrique élevée (1000V) qui crée un champ électrique. Le photomultiplicateur se base sur **le principe d'émission secondaire**, selon lequel un matériau émet un électron lorsqu'il est soumis à un rayonnement ou un flux de particules incident. Un photon qui entre dans le tube photomultiplicateur incide sur une photocathode; par effet d'émission secondaire, un électron, appelé "photoélectron", est émis. Le photoélectron arrive dans l'électromultiplicateur: il est accéléré par le champ électrique, et incide à son tour sur une première dynode; d'autres électrons sont émis à nouveau par principe d'émission secondaire électronique et sont accélérés, et incident ensuite à leur tour sur des dynodes successives, en étant toujours accélérés et de plus en plus nombreux. Les électrons résultants de ce parcours sont ensuite absorbés par l'anode; cela ferme le circuit composé de l'alimentation de haute tension et du tube lui-même, et une impulsion électrique apparaît sur les bornes externes du tube photomultiplicateur. Cette impulsion est enfin normalisée par des composants électroniques, afin de suivre le standard NIM de mesure utilisé dans le domaine de la physique des particules.

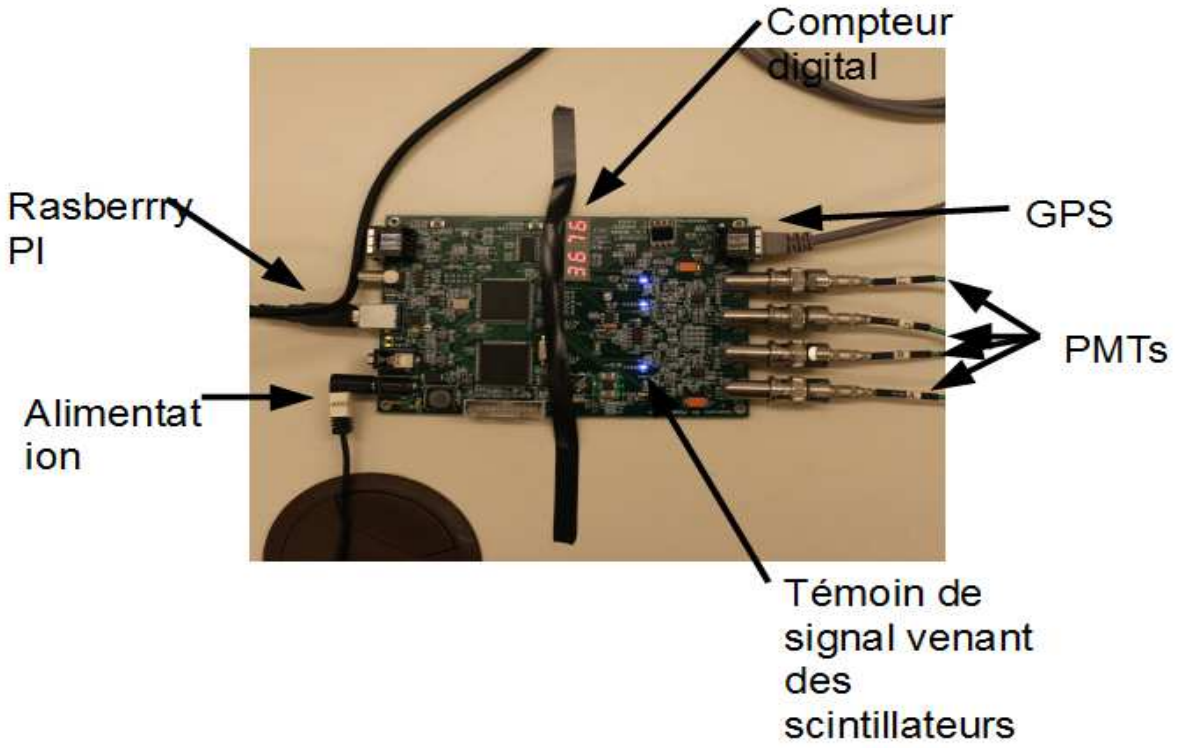


Figure 5: La Carte d'Acquisition de Données (DAQ)

L’impulsion lumineuse, et par conséquent l’impulsion électrique résultante, sont **proportionnels à l'énergie du muon** qui a traversé le scintillateur.

Les impulsions électriques sont transmises à la carte d’acquisition de données Quarknet; sur celle-ci, les signaux des 4 canaux sont amplifiés par un facteur x10. Ensuite, des comparateurs assurent le rôle de discriminant; en effet, lorsque le signal analogique est inférieur à un seuil de détection prédéfini, aucun signal n’est émis par le comparateur (valeur logique 0). En revanche, si le signal analogique dépasse le seuil de détection, un signal haut est transmis (valeur logique 1). Sur chaque canal, une puce de type “Time to Digital Converter” (convertisseur temps vers valeur digitale) enregistre le temps auquel le seuil a été dépassé et le temps auquel le signal est retombé en-dessous du seuil de détection. Ces signaux sont transmis à un circuit de logique programmable (CPLD en anglais) qui assure le rôle de module de coïncidence (“coincidence unit” en anglais): si les signaux des canaux sélectionnés sont en valeur logique 1 en même temps, alors le circuit de logique transmet les valeurs de temps de montée et de descente du signal sur chaque canal au microprocesseur (également appelé module de logique lente, “slow logic” en anglais). Enfin, le microprocesseur collecte les données du circuit de logique programmable et celles du module GPS, pour enfin les envoyer par connexion en port série à l’ordinateur (Raspberry Pi) sous forme de texte ASCII.

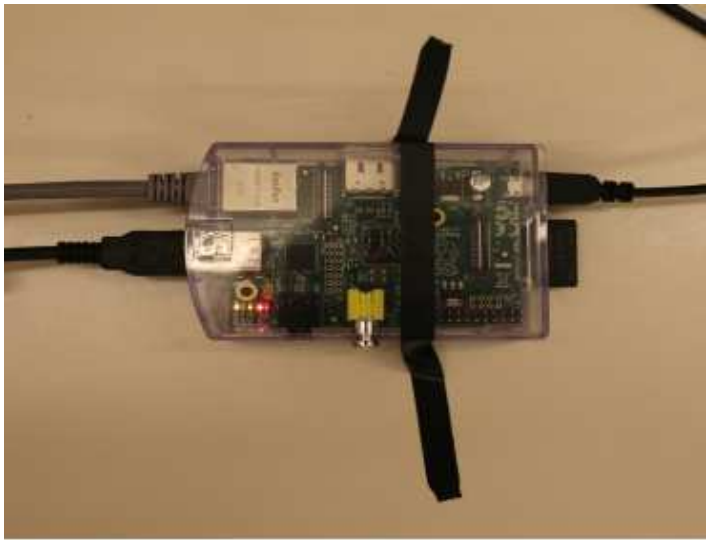


Figure 6: La Raspberry PI

Les donnés transmises au Rasberry PI par la DAQ sont de la forme:

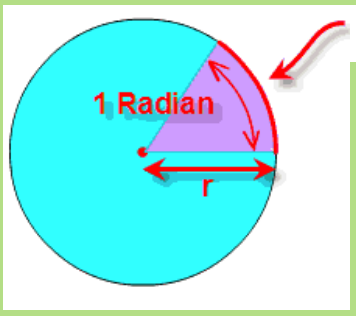
80EE0049 80 01 00 01 38 01 3C 01 7EB7491F 202133,242 080803 A 04 2 -0389
80EE004A 24 3D 25 01 00 01 00 01 7EB7491F 202133,242 080803 A 04 2 -0389
80EE004B 20 01 00 23 00 01 00 01 7EB7491F 202133,242 080803 A 04 2 -0389
80EE004C 01 2A 00 01 00 01 00 01 7EB7491F 202133,242 080803 A 04 2 -0389
80EE004D 00 01 00 01 00 39 32 2F 81331170 202133,242 080803 A 04 2 +0610

Le Rasberry Pi est un micro-ordinateur de base que nous avons laissé en marche au CERN. Il a été connecté au réseau du CERN pour qu’il soit possible d’y accéder depuis un ordinateur personnel à distance, par connexion SSH (Secure SHell, invite de commande crypté pour systèmes de type UNIX) ou VNC (vue d’interface graphique à distance).

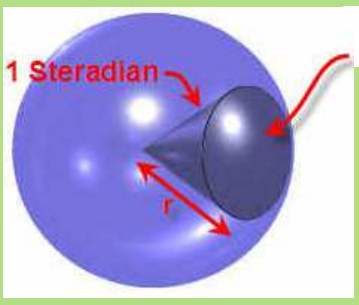
d-3 Calcul de l'angle de détection

Box d'approfondissement: angle solide et stéradians

On utilise les stéradians pour mesurer les angles solides, tout comme on utilise les radians pour mesurer les angles sur une surface plane. L'angle solide est l'équivalent de l'angle plan en trois dimensions.



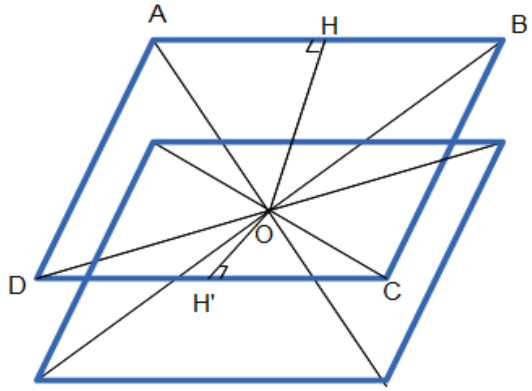
Si dans un plan il y a un point et qu'on relie ce point avec les extrémités d'un segment, les deux droites obtenues renferment une partie du plan: c'est l'angle sous lequel le segment est vu par le point. Si on applique ce principe à un cercle, l'origine est reliée aux 2 extrémités d'une partie de la circonférence par deux droites: si on mesure l'angle en radians, la partie de la circonférence équivaut à r.



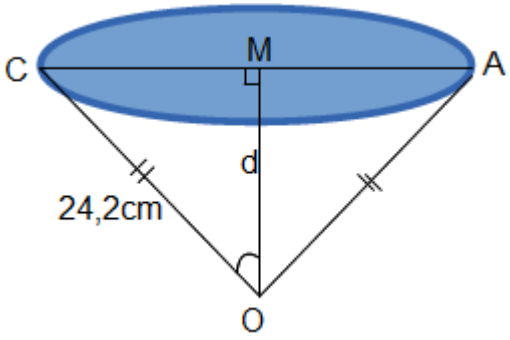
De la même manière, s'il y a un point et qu'on le relie au contour d'une surface dans l'espace, on obtient une surface qui renferme une partie de l'espace. Généralement, c'est un cône. C'est donc l'angle solide sous lequel la surface est vue par le point. Si on applique ce principe à une sphère, l'origine est reliée à une partie de la surface de la sphère, qui est égale à r^2 si on mesure en stéradians.

La mesure d'un angle en stéradians équivaut au rapport entre la surface de la sphère et le rayon au carré. Du coup, l'angle solide total (celui qui comprend toute la surface de la sphère) vaut $4\pi R^2/R^2 = 4\pi$ stéradians.

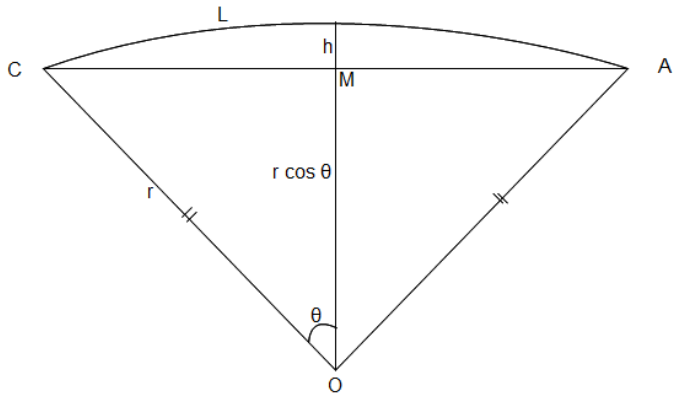
Il est intéressant de connaître l'angle de détection pour savoir précisément ce que nous détectons. Pour cela, nous allons calculer un encadrement de l'angle solide de l'angle de détection de notre détecteur.



<u>Calcul de l'angle de détection 1:</u>



On considère le cône de révolution de sommet O et de base de diamètre CO=24.2cm et où d=12.25cm. On va chercher la mesure de l'angle CÔM.



Dans le triangle OCM rectangle en M,

cos CÔM=12.25/24.2
cos CÔM≈0.506
CÔM=59.6°

On sait que le radian (rad) correspond à l'arc de cercle (L) divisé par le rayon (r).

rad=L/r

On peut transposer cela dans un espace pour trouver une mesure en stéradians (Ω).
Ω=A (aire du "chapeau") /r²

On connait la surface d'une sphère: 4∏r², donc: 0≤Ω≤4∏

A=2∏rh or h=r-r cos θ
h=r(1- cos θ)

donc si on remplace:

Ω=A/r²
Ω=2∏rh/r²
Ω=2∏r² (1- cos θ) /r²
Ω=2∏ (1- cos θ)

si maintenant on remplace par des valeurs:

Ω=2∏ (1- cos θ)
Ω≈2∏ (1-0.506)
Ω≈0.988∏
<u>Calcul de l'angle de détection 2:</u>

On a le triangle ABO isocèle en O avec AB=30cm et AO=BO= 24.2cm
Si on trace la hauteur [OH]
On se place dans le triangle AHO rectangle en H,

D'apres le théorème de Pythagore,

HO²=AO²-AH² HO=√360.64
HO²=585.64-225 <u>HO≈18.99 cm</u>
HO²=360.64

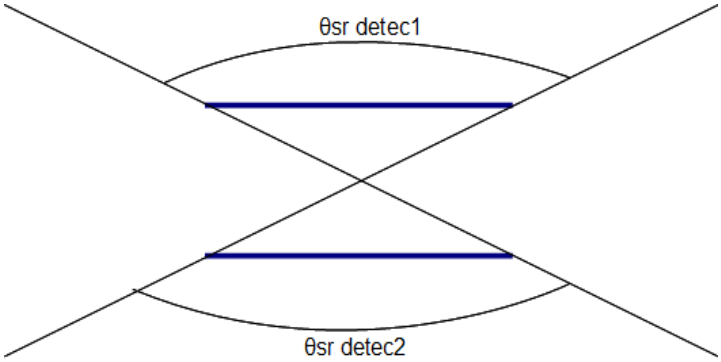
On considère le triangle HÔM où HO≈18.99cm et d=12.25cm:

cos HÔM= 12.25/18.99
cos HÔM≈0.645

On va réutiliser la formule utilisé précédamment:

Ω'=2∏ (1- cos θ')
Ω'≈2∏ (1-0.645)
Ω'≈0.71∏

Notre angle de détection pour les muons arrivant par le haut du détecteur est alors compris entre 0.71∏sr et 0.988∏sr:
0.71∏≤θsrdetec1≤0.988∏



Or un même angle est présent sur l'autre face du détecteur: θsrdetec2
Or θsrdetec2=θsrdetec1

donc θsrdetec=2θsrdetec1

donc **0.71∏≤θsrdetec1≤0.988∏**
1.42∏≤θsrdetec≤1.976∏

Donc notre angle de détection de muons est compris entre 1.42∏sr et 1.976∏sr:
1.42∏≤θsrdetec≤1.976∏

d-4 Premières données, calibration et mise en marche

Calcul du temps de voyage du muon entre les scintillateurs

Pour pouvoir compter les muons, il faut pouvoir les différencier du "bruit" extérieur. Pour cela il faut qu'il traversent les 4 scintillateurs quasiment en même temps. Mais pour le réglage du temps de coïncidence, il fallait savoir combien d temps le muon mettait pour traverser les 4 scintillateurs. Il fallait que ce temps soit inférieur au temps conseillé pour le temps de coïncidence: 40 nanosecondes.
Pour réaliser le calcul ci-dessous, nous avons utilisé comme distance la distance entre les deux scintillateurs les plus extérieurs.

Pour distance 25cm :

v=d/t
t=d/v

Δt=0,25m/c
Δt=0,25m/3,00*10 puissance 8m/s
Δt=8,33*10 puisssance -10 s
Δt=0,833ns

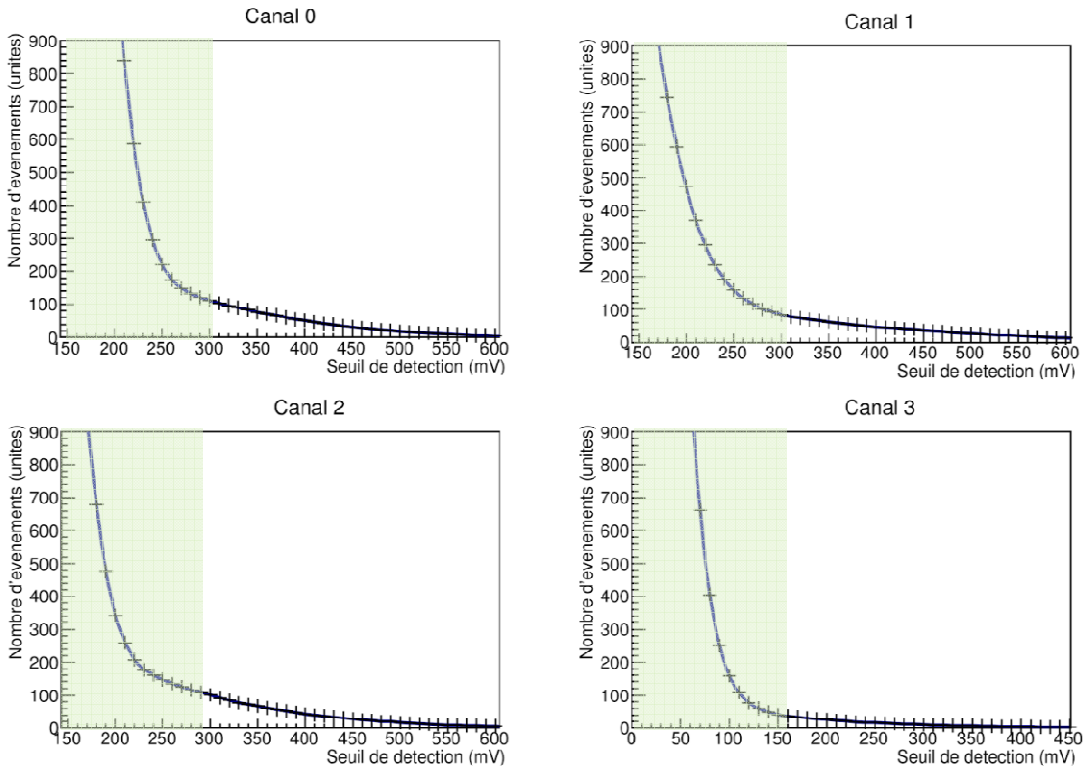
Cette durée est négligeable si on considère un temps de coïncidence de 40 nanosecondes. Donc nous pouvons conclure que le muon traverse les 4 scintillateurs presque en même temps.

Préparation aux expériences - calibration du détecteur

Afin de réaliser des expériences de mesures concernant les muons, il est nécessaire de calibrer l'équipement afin de limiter le bruit lors des mesures; en effet, le bruit de mesure, qui inclut tous les signaux indésirables qui viennent se superposer sur le signal étudié, peut être, entre autres, d'origine thermique ou électrique (variations de tensions indésirables sur l'alimentation secteur, ou variations de tensions dues aux composants du détecteur lui-même). Afin de séparer les vrais évènements d'évènements générés par du bruit sur le signal, il est nécessaire de déterminer un seuil de détection.

Nous avons écrit un programme afin d'automatiser le processus et permettre d'effectuer des mesures précises, un ordinateur ayant un délai de réaction bien inférieur à une personne. Le programme "CosmicCalibrator" est placé en Appendix A de ce document.

Nous avons mesuré le nombre d'évènements enregistrés par le détecteur pendant un temps défini de 300 secondes, et cela pour des valeurs de seuils de détection entre 10mV et 1500mV en incréments de 10mV (10mV, 20mV, 30mV, …) pour les 4 canaux. Nous avons ensuite tracé les graphiques du nombre d'évènements enregistrés en fonction du seuil de détection pour chaque canal; ces graphiques apparaissent ci-dessous.



Nombre d'évènements enregistrés selon le seuil de détection

Une fois les graphiques construits, il est possible de déterminer un point auquel le nombre de détections se stabilise et cesse de diminuer très rapidement; au seuil de détection correspondant a l'abscisse de ce point, le bruit cesse d'être enregistré, et le signal devient plus "propre". L'abscisse identifiée est alors le seuil de détection du canal correspondant au graphique.

Nous avons déterminé les seuils de détection suivants:
Canal 0: threshold_ch0 = 300mV
Canal 1: threshold_ch1 = 300mV
Canal 2: threshold_ch2 = 280mV
Canal 3: threshold_ch3 = 150mV

De plus, afin de limiter le bruit, toutes les mesures sont effectuées en niveau de coïncidence 4; la carte d'acquisition de données n'enregistre un évènement si et seulement si les signaux des 4 canaux coïncident au même moment.

3) Quelles sont les caractéristiques du flux de muons qui nous parviennent?

Hypothèse: Le flux de rayons cosmiques varie en fonction de l'inclinaison du détecteur, mais pas en fonction du temps.

3) a - Etude des variations du flux de muons en fonction du temps

Nous avons placé le détecteur à un angle d'inclinaison constant θ=0°, et nous avons ensuite lancé une série d'acquisition de données à l'aide de notre programme "CosmicDAQ" pendant quelques jours, afin d'avoir assez de données pour pouvoir effectuer l'étude.

Le traitement de données a nécessité le développement du programme "CosmicInterpreter", l'utilisation de l'outil ROOT du CERN et le développement de plusieurs macros (composants logiciels) pour ROOT.

L'acquisition des données analysées sur la figure 1 ci-dessous sont le résultat de 82h d'acquisition de données , ayant enregistré un total de 1226851 évènements.

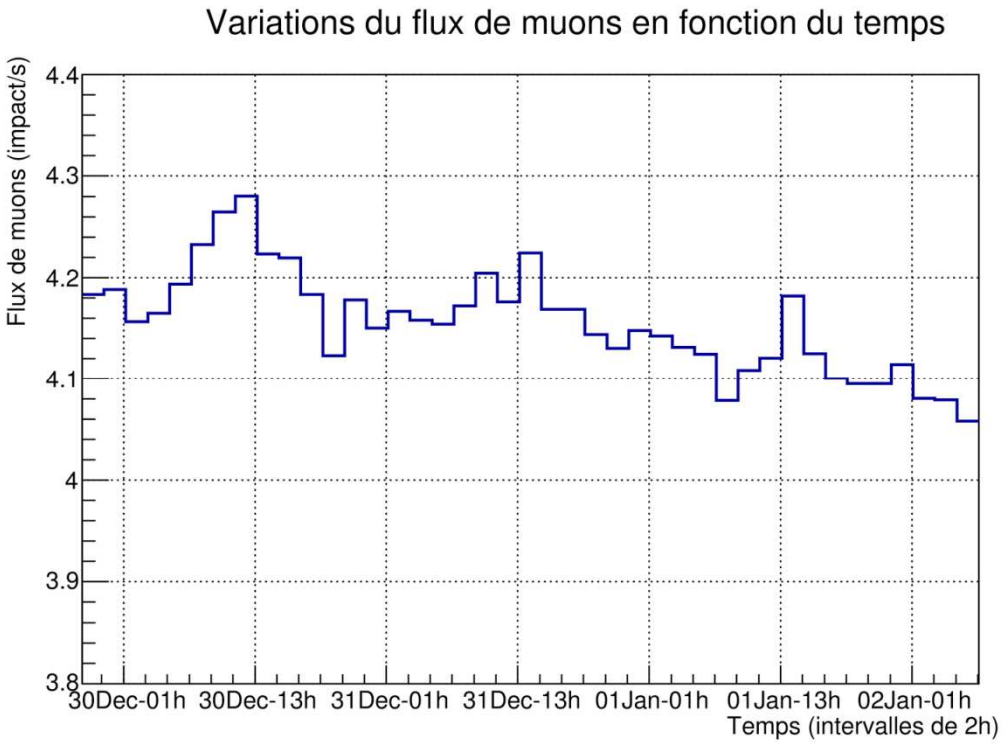


fig. 1: Histogramme représentant les variations du flux de muons en fonction du temps

L'utilisation d'intervalles de 2h sur la figure 1 fait apparaître plus clairement les variations lentes. Il est possible d'observer deux types de variations sur le graphique:

- des fluctuations récurrentes du flux de muons sur des cycles de 24h de l'ordre de ± 0.06 impact/s,
- une diminution constante du flux de muons, observable avec la pente du graphique.

Il est alors possible d'émettre l'hypothèse d'une corrélation entre l'intensité du flux de muons détecté et le cycle journalier de 24h. Concernant la diminution constante observée sur le graphique, il est également possible d'émettre l'hypothèse d'une corrélation entre l'intensité du flux et un cycle annuel.

Les résultats de cette expérience sont néanmoins non concluants; en effet, il nous est impossible de déterminer si les variations observées représentent les phénomènes décrits dans les hypothèses, ou bien si ces variations n'ont aucune corrélation avec le paramètre étudié, et ne sont que le résultat de l'instabilité du détecteur et/ou d'un matériel défectueux ou peu fiable. Un diagnostic approfondi du matériel, ainsi qu'une étude sur une échelle de temps plus importante seraient nécessaire afin de pouvoir confirmer ou réfuter les hypothèses émises lors de cette expérience.

3) b - Distribution des valeurs de flux de muons

Les données utilisées lors de la première expérience peuvent être réutilisées afin de déterminer la distribution des valeurs de flux de muons, ainsi que l'erreur de mesure du détecteur. On effectue un histogramme similaire à la figure 1, mais en utilisant des intervalles de 300 secondes (5 minutes); cet histogramme apparaît sur la figure 2:

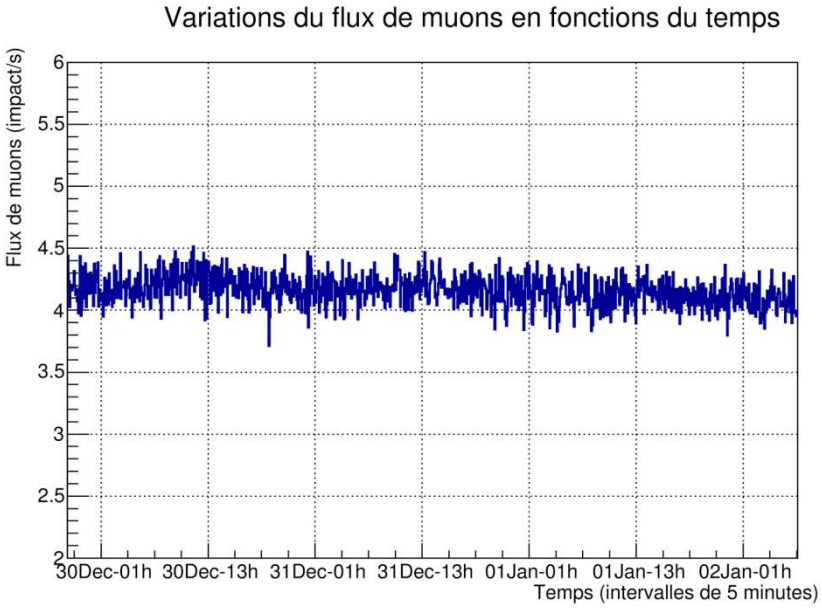


fig.2: Histogramme représentant les variations du flux de muons en fonction du temps

A partir de la figure 2, on trace la distribution des valeurs afin d'obtenir la distribution des valeurs du flux de muons; le résultat apparaît dans la figure 3:

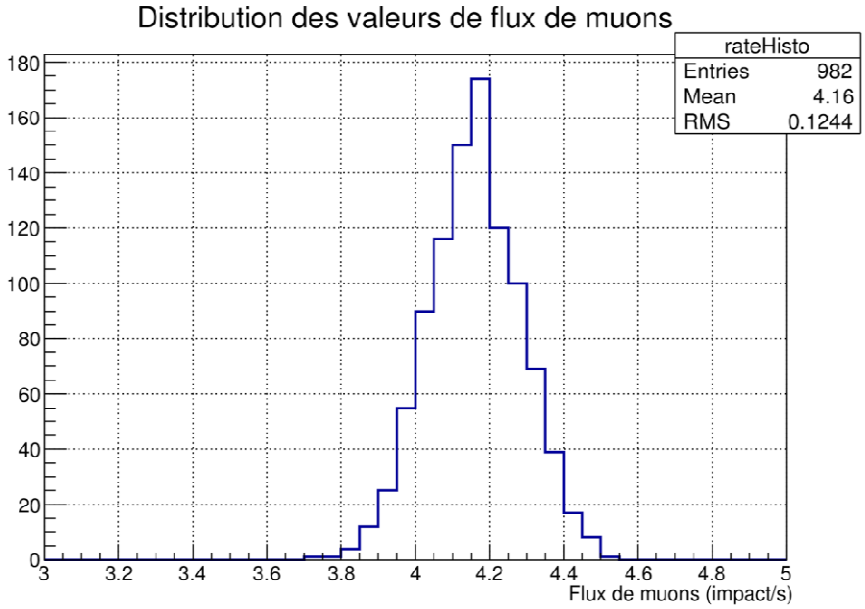


fig.3: Histogramme représentant la distribution des valeurs de flux de muons

A l'aide de l'outil informatique ROOT, on obtient la valeur moyenne $\bar{x}$ ("Mean" en anglais), ainsi que l'écart-type σ (sur le graphique, l'écart-type est donné par la valeur "RMS", qui, en anglais, désigne "Root Mean Square", soit la moyenne quadratique; il s'agit d'une erreur de terme, l'écart-type et la moyenne quadratique étant utilisés comme synonymes dans le jargon des physiciens. La valeur représentée est bien l'écart-type).
Ces valeurs sont obtenues respectivement par les formules:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$ avec N le nombre de valeurs sur la fig.3
x_i la valeur de la fig.3 d'index i

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Selon le graphique,
$\bar{x}$ =4.16 impact/s
σ=0.124 impact/s

On définit l'erreur absolue par la valeur σ.
On calcule l'erreur relative e:
e = σ / $\bar{x}$ = 0.124/4.16 =2.98%

Nous avons pu alors déterminer la distribution des valeurs du flux de muons, et ainsi déterminer l'erreur relative et absolue des mesures, afin de pouvoir l'exploiter dans l'expérience suivante.

3)c - étude du flux de muons incident détecté en fonction de l'inclinaison du détecteur, et par conséquent, sous l'angle d'écartement su zénith

Nous avons effectué des séries d'acquisition de données de 300 secondes (5 minutes) pour chaque angle d'inclinaison θ de 0° à 90° en incréments de 15°, pour ensuite déterminer les valeurs du flux de muons incident moyen pour chaque angle. L'acquisition de données a été effectuée avec notre programme "CosmicDAQ", inclus avec ce document en Appendix A. Les résultats de l'expérience apparaissent dans la figure 4.

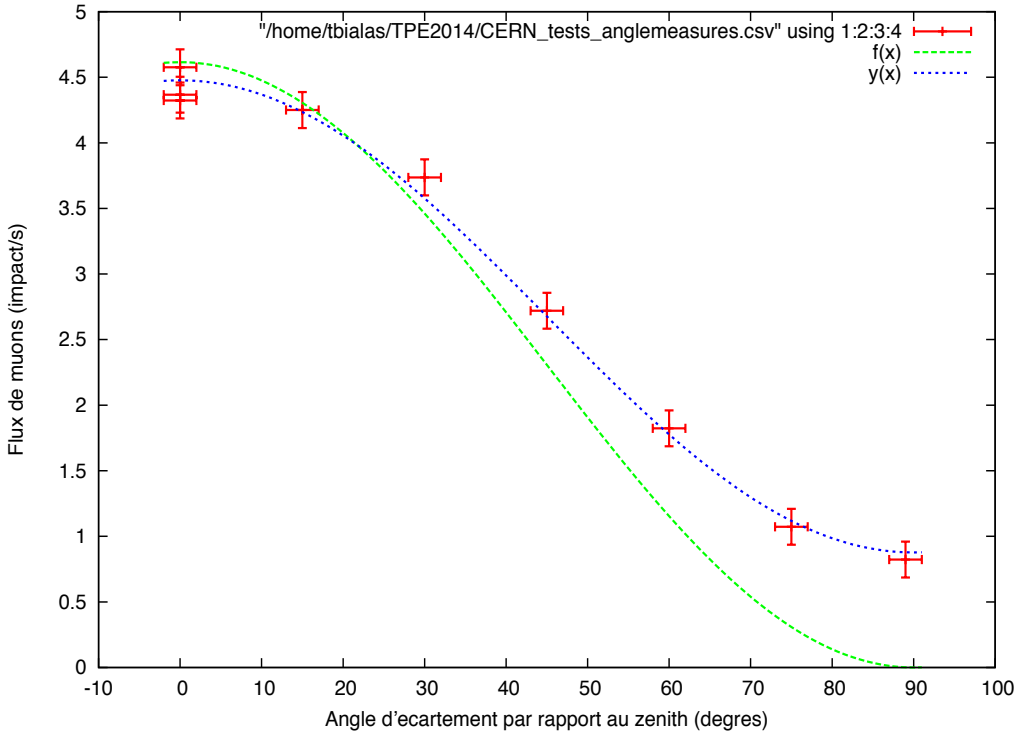


fig.4: Évolution du flux de muons détectable en fonction de l'angle θ

Sur la figure 4, les courbes de fonctions suivantes ont également été ajustées aux points expérimentaux et tracées sur l'ensemble de définition *Df*=[0;90]:

$f(\theta)=a*\cos^2(\theta)$ avec a=4.6151 (±0.2353)

$y(\theta)=a*\cos^2(\theta)+b$ avec a=3.59957 (±0.09093) et b=0.876996 (±0.06606)

Les incertitudes de mesure du flux de muons, définies par l'écart type précédemment, sont représentées sur le graphique par des barres d'erreurs verticales.
Les incertitudes de mesure de l'angle sont représentées sur le graphique par des barres d'erreurs horizontales; elles ont une valeur de ±2°.

La fonction fcorrespond à la relation empirique

$$I \propto \cos^2(\theta)$$

entre le flux de muons détectable I et l'angle d'inclinaison θ par rapport au zénith; cette relation fut pour la première fois formulée en 1948 dans une publication de Bruno Rossi sur les rayons cosmiques, document destiné à la préparation de la conférence de Solvay (1948).

Les résultats de notre expérience divergent de la relation théorique, notamment aux angles les plus éloignés au zénith (intervalle θ=[60°;90°]); en effet, la fonction $y(\theta)$est la fonction de la courbe ajustée aux points expérimentaux qui ajoute une constante bà la fonction $f(\theta)$afin de permettre une translation verticale de la courbe, et donc un meilleur ajustement. Cette divergence entre la théorie et le résultat de notre expérience est probablement due à plusieurs facteurs:

- l'angle de détection relativement élevé (calculé précédemment): l'angle de détection indique une ouverture assez importante du détecteur, et donc un ratio d'acceptance de muons plus élevé; par conséquent, les muons du flux mesuré ne sont pas toujours issus de l'angle d'écartement au zénith désiré.
- la radioactivité naturelle de fond; en effet, ayant des propriétés de rayonnement similaire aux rayons cosmiques, elle peut provoquer des détections d'évènements supplémentaires pendant les mesures,
- le bruit électronique: malgré la définition de niveaux de détection adaptés ainsi que l'utilisation d'un niveau de coïncidence 4 afin de limiter le bruit électronique, il existe une probabilité selon laquelle le détecteur enregistrera un signal de bruit sur les 4 canaux en même temps, provoquant l'enregistrement d'un évènement.

Cette expérience est alors un succès, malgré des inexactitudes de mesure majoritairement liées à la configuration géométrique de notre détecteur; par conséquent, il y a bien un lien entre l'intensité du flux de muons détecté et l'angle d'inclinaison par rapport au zénith du détecteur, et notre expérience confirme une relation empirique définie auparavant par des scientifiques.

4) Conclusion, remerciements et appendix

Conclusion

Au début du TPE, nous étions partis avec une question: “Comment détecter le flux de muons, particules issues du rayonnement cosmiques, arrivant sur Terre?” A la fin de cette longue recherche nous sommes donc fiers de pouvoir y répondre. Au cours de ce TPE, nous avons pu découvrir les rayons cosmiques au cours d’une recherche documentaire approfondie. Nous avons pu comprendre leur composition et les méthodes de détection utilisées afin de les étudier. Nous avons également eu l’occasion d’utiliser un matériel professionnel afin d’effectuer une véritable étude scientifique, en alliant physique et mathématiques tout au long du processus.

Nous avons étudié trois méthodes qui permettent de détecter le rayonnement cosmique: la chambre à brouillard, la chambre à étincelles et le détecteur de muons à scintillation.

A la fin de la démarche expérimentale effectuée à l’aide du détecteur à scintillation, nous avons pu confirmer notre hypothèse selon laquelle le flux de muons détectable à la surface de la Terre varie selon l’angle d’écartement par rapport au zénith, ce qui est en accord avec une relation empirique confirmée par des chercheurs. Nous avons également pu définir la distribution des valeurs de flux de muons lors de la mesure précédente, permettant ainsi le calcul de l’erreur de mesure de notre expérience. Nous n’avons toutefois pas pu ni confirmer, ni réfuter notre thèse concernant les variations du flux de muons dans le temps, du fait de l’incertitude liée à la fiabilité du matériel ainsi que d’un trop court laps de temps pour réaliser nos mesures.

Ce TPE s’est révélé vraiment intéressant, puisqu’il nous a permis de faire connaissance avec une partie de la physique la plus théorique: la physique des particules. Le fait que près de chez nous, tous les jours, des physiciens analysent des particules pour en savoir plus sur la matière qui nous entoure, nous a paru impressionnant.

Enfin, le travail de restauration du détecteur, effectué afin de pouvoir réaliser l’expérience, servira à relancer une partie de l’exposition consacrée aux rayons cosmiques du site CMS au CERN; de cette façon, il sera possible de sensibiliser plus de visiteurs au rayonnement cosmique, phénomène invisible et donc méconnu; ainsi que plus généralement, à la physique des particules.

Remerciements

Nous voudrions adresser nos remerciements les plus sincères aux personnes suivantes:

- nos professeurs encadrants, Mr. Waitier et Mme Caillarec
- le personnel du laboratoire CERN, notamment l’équipe du département Physique, Technologie, et de l’expérience CMS
- le personnel de l’université de Notre-Dame dans l'Etat Indiana aux États-Unis
- le personnel du laboratoire Fermilab aux États-Unis

Sources:

Pour les rayons cosmques et leur histoire:
http://it.wikipedia.org/wiki/Raggi_cosmici
http://fr.wikipedia.org/wiki/Rayon_cosmique
http://en.wikipedia.org/wiki/Cosmic_ray
http://home.web.cern.ch/about/physics/cosmic-rays-particles-outer-space
http://www.astronomes.com
http://fr.wikipedia.org/wiki/Physique_des_plasmas
http://fr.wikipedia.org/wiki/Vent_solaire
http://www.pourlascience.fr/ewb_pages/a/actu-les-supernovae-source-de-rayons-cosmiques-31103.php
http://www.nasa.gov/mission_pages/GLAST/news/cosmic-rays-source.html
http://www.astroparticelle.it/presentazione.asp
http://www.canalu.
tv/video/cerimes/a_la_recherche_des_sources_de_rayons_cosmiques_de_ultra_haute_energie.14224
http://www.ilgiornale.it/news/confermate-nasa-sessanta-anni-teorie-fermi-sui-raggi-cosmici.html
http://www.asimmetrie.it/index.php/voci-dell-universo
http://www.nmdb.eu/?q=node/416
Pour les muons et les particules élémentaires:
http://luigispagnolo.it/index.php?option=com_content&view=article&id=46:un-po-di-storia-sulmuone&
catid=39:approfondimenti-fisica&Itemid=80
http://it.wikipedia.org/wiki/Particella_elementare
http://it.wikipedia.org/wiki/Muone
http://www.treccani.it/enciclopedia/muone/
https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Muon.html
http://hyperphysics.phy-astr.gsu.edu/hbase/particles/muonhist.html
http://www.lafisica.info/i_muoni_.html
Le monde des particules (Brian Southworth et Georges Boixader)
Pour les moyens de détection:
http://www.astroparticelle.it/muon-detector.asp
http://cms.web.cern.ch/news/muon-detectors
http://www.scienceinschool.org/2010/issue14/cloud/french
http://ch.lagoute.free.fr/CosmoDCL/RCCosmoDCL.pdf
Planetmaths: http://planetmath.org/solidangle
sphérical cap: http://en.wikipedia.org/wiki/Spherical_cap
http://www.mathsisfun.com/geometry/steradian.html
http://aalem.free.fr/maths/C07-ANGLES-SOLIDES.PDF

Claus Grupen, Boris Shwartz, “Detectors for Particle Radiation”, Cambridge University Press, 1998, ISBN: 978-0-521-84006-4 hardback

Chung Yau Elton Ho, “Cosmic Ray Muon Detection using NaI Detectors and Plastic Scintillators”, University of Virginia, 2013
http://home.fnal.gov/~group/WORK/muonDetection.pdf

Glenn F. Knoll, “Radiation Detection and Measurement, fourth edition”, Wiley, 2010, ISBN: 978-0-470-13148-0

Pour les résultats:

Yi-Hong Kuo, “Determination of the Angular Distribution of Cosmic Rays at sea level”, Bachelor of Science in Physics thesis, Massachusetts Institute of Technology, June 2010

http://hdl.handle.net/1721.1/61208

Bruno Rossi, “Interpretation of Cosmic-Ray Phenomena”, Review of Modern Physics, Volume 20, Number 3, Physics Department and Laboratory for Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1948
http://socrates.berkeley.edu/~phylabs/adv/ReprintsPDF/MUO%20Reprints/02%20-%20Cosmic-Ray%20Phenomena.pdf

Appendix A:

1) “CosmicCalibrator”, programme de calibration

```
/*
 * Cosmic Calibrator v0.1-alpha5
 * Program to find detection thresholds for QuarkNet DAQ cards
 * Requires CERN ROOT
 *
 *
 * Programme ecrit pour un projet de Travaux Pratiques Encadres en classe de
 * 1ere Scientifique (1S2 du lycee de Ferney-Voltaire) par Tomasz BIALAS
 *
 * Groupe: Donatella AVONI, Tomasz BIALAS, Nils VAN WEELDEREN
 *
 * Copyright 2014 Tomasz Bialas <tbx1024 at gmail dot com>
 * TPE 2014, Lycee de Ferney Voltaire
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */

#include <stdlib.h>                  // provides basic features
#include <stdio.h>                   // provides basic i/o features
#include <string.h>                  // provides basic string features
#include <fcntl.h>                   // provides file descriptor manipulation
features
#include <errno.h>                   // provides error output features
#include <termios.h>          // provides TTY features
#include <unistd.h>                  // provides    POSIX system API features (read,
write, close)
#include <time.h>                    // provides local time
#include <sstream>                   // provides C++ streams
#include <TFile.h>                   // CERN ROOT TFile
#include <TNtuple.h>          // CERN ROOT NTuple
#include <TH2.h>                     // CERN ROOT 2D graphing functions
#include <TString.h>          // CERN ROOT TString
#include <Riostream.h>        // C++ iostreams
#include <TSystem.h>          // CERN ROOT TSystem for accessing underlying operating
system
#include <TCanvas.h>          // CERN ROOT TCanvas for graphing
#include <TFrame.h>                  // CERN ROOT TFrame for graphing
#include <TApplication.h>     // CERN ROOT TApplication for standalone out-of-
interpreter running
#include <TGraph.h>                  // CERN ROOT TGraph for graphing
#include <TMultiGraph.h>      // CERN ROOT for creating multiple graphs
#include <TStyle.h>                  // CERN ROOT for applying styles to graphs

#define _POSIX_SOURCE 1
//char data[128];
char input[4096];
char globaloutput[4096];
unsigned long int timetest;
int thresholdtest;
TFile *rootfile;
TNtuple *ntuple;



void process() {
        int s1,s2,s3,s4,s5;
        float f1,f2,f3,f4,f5;
        char* data[8];
        // Parse input string into scalers integers
```

```
        data[0] = strtok(globaloutput," ");

    data[1] = strtok(NULL, " ");
        memmove(data[1], data[1]+3, strlen(data[1]));
    s1 = strtol(data[1],0,16);
    f1 = float(s1)/float(timetest);
    printf("%4.2f\n", f1);

    data[2] = strtok(NULL, " ");
    memmove(data[2], data[2]+3, strlen(data[2]));
    s2 = strtol(data[2],0,16);
    f2 = float(s2)/float(timetest);
    printf("%4.2f\n", f2);

    data[3] = strtok(NULL, " ");
    memmove(data[3], data[3]+3, strlen(data[3]));
    s3 = strtol(data[3],0,16);
    f3 = float(s3)/float(timetest);
    printf("%4.2f\n", f3);

    data[4] = strtok(NULL, " ");
    memmove(data[4], data[4]+3, strlen(data[4]));
    s4 = strtol(data[4],0,16);
    f4 = float(s4)/float(timetest);
    printf("%4.2f\n", f4);

    data[5] = strtok(NULL, " ");
    memmove(data[5], data[5]+3, strlen(data[5]));
    s5 = strtol(data[5],0,16);
    f5 = float(s5)/float(timetest);
    printf("%4.2f\n", f5);

    data[6] = strtok(NULL, " ");
    memmove(data[6], data[6]+3, strlen(data[6]));
    data[7] = strtok(NULL, "\0");

        // Store data to NTuple
        ntuple->Fill(float(thresholdtest),f1,f2,f3,f4,f5);

        }

void send(int tty,char* input){
        char output[4096];
        strcpy(output,"");
        char RX[1024];
        strcpy(RX,"");
        int status = 0;
        for (unsigned int i=0; i<strlen(input); i++){
                // printf("Sending '%c' \n",input[i]);
                status = write(tty,&input[i],1);
                        if (status < 0) {
                                fprintf(stderr,"Error occured while writing to serial
device. Exiting. \n");
                                exit(1);
                        }
                usleep(50);
        }
        write(tty,"\r",1);
        sleep(1);
        while (read(tty,&RX,1) > 0) {
                //printf("|%s",data);
                for (unsigned int y=0; y<strlen(RX); y++){
                        sprintf(output, "%s%s", output, &RX[y]);
                }

        }

        //printf("[DEBUG] DATA OUTPUT IS: %s \n",output);
        strcpy(globaloutput, output);

        }

int prepare(int tty){
        char flooddata[16384];
        int attempt;
        attempt=0;
        // Prepare board for calibration
    printf("Preparing DAQ board for calibration...\n");
    strcpy(input,"");
```

```
    send(tty,input);
    printf("Disabling data output [CD]... \n");
    strcpy(input,"CD");
    send(tty,input);
    while (read(tty,&flooddata,16384) > 0 && attempt > 6) {
                        printf("Received data when not expected, retrying... \n");
                        send(tty,input);
                        sleep(5);
                        attempt++;
                }
    printf("Disabling status messages [ST 0]... \n");
    strcpy(input, "ST 0");
    send(tty,input);
    printf("Setting 4-fold coincidence count and trigger to all channels [WC 0 3F]...
\n");
    strcpy(input, "WC 0 3F");
    send(tty,input);
    printf("Setting gate width to 100ns [WC 2 A]... \n");
    strcpy(input, "WC 2 A");
    send(tty,input);
       printf("Test [DS]... \n");
    strcpy(input, "DS");
    send(tty,input);
       return 0;
       }

int setup(char* filename, int baudrate){
       // Open serial device
       printf("Using device %s with baudrate %i \n",filename,baudrate);
       int tty;
       tty=open(filename, O_RDWR | O_NONBLOCK);
       if(tty < 0) {
              fprintf(stderr,"Error while opening device %s. Exiting. \n",filename);
              exit(1);
              }

       // Setup serial terminal settings
       struct termios terminal;
       memset(&terminal,0,sizeof(terminal));
    terminal.c_cflag=CSTOPB|PARENB|IXON|IXOFF|HUPCL;
    terminal.c_iflag &= ~(ICRNL|IGNCR);
    terminal.c_oflag=0;
    terminal.c_cflag=CS8|CREAD|CLOCAL;
    terminal.c_lflag=0;
    terminal.c_cc[VMIN]=1;
    terminal.c_cc[VTIME]=5;

    // Baudrate setup - check for compatibility with supported DAQ baudrates and set
baudrate if compatible
       bool compatible_baudrate = false;
       if (baudrate == 19200) {
              compatible_baudrate = true;
              cfsetispeed(&terminal,B19200);
              cfsetospeed(&terminal,B19200);
              }
       if (baudrate == 38400) {
              compatible_baudrate = true;
              cfsetospeed(&terminal,B38400);
              cfsetospeed(&terminal,B38400);
              }
       if (baudrate == 57600) {
              compatible_baudrate = true;
              cfsetispeed(&terminal,B57600);
              cfsetospeed(&terminal,B57600);
              }
       if (baudrate == 115200) {
              compatible_baudrate = true;
              cfsetispeed(&terminal,B115200);
              cfsetospeed(&terminal,B115200);
              }
       if (baudrate == 230400) {
              compatible_baudrate = true;
              cfsetispeed(&terminal,B230400);
              cfsetospeed(&terminal,B230400);
              }
       if (baudrate == 460800) {
              compatible_baudrate = true;
              cfsetispeed(&terminal,B460800);
              cfsetospeed(&terminal,B460800);
```

```
                }
        if (baudrate == 921600) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B921600);
                cfsetospeed(&terminal,B921600);
                }

        // If the baudrate is unsupported, exit.
        if (compatible_baudrate == false) {
                fprintf(stderr,"Error: the specified baudrate is not supported by the
DAQ board. Exiting. \n");
                exit(1);
                }

    if (tcsetattr(tty,TCSANOW,&terminal) < 0) {
                fprintf(stderr,"Error: failed to setup serial port. Exiting. \n");
                exit(1);
         }

    tcflush(tty, TCIOFLUSH);
        return tty;
        }



int main(int argc, char **argv)
{
        printf("CosmicCalibrator v0.1-alpha5 \n");

        // Check command line arguments
        if (argc != 6) {
                printf("Usage: cosmiccalibrator <device_name> <baudrate> <start
threshold> <end threshold> <test time (s)> \n");
                fprintf(stderr,"Error: wrong number of arguments specified. Exiting.
\n");
                exit(1);
                }
        TApplication* rootapp = new TApplication("CosmicCalibrator",&argc, argv);
        rootapp->SetReturnFromRun(true);
        // Parse command line arguments
        char filename[512];
        strcpy(filename,argv[1]);
        char baudrate_arg[128];
        strcpy(baudrate_arg,argv[2]);
        char startthreshold[10];
        strcpy(startthreshold, argv[3]);
        char endthreshold[10];
        strcpy(endthreshold,argv[4]);
        char testtimechar[10];
        strcpy(testtimechar, argv[5]);

        // Parse baudrate value
        long int baudrate;
        baudrate = atol(baudrate_arg);

        // Parse start and end thresholds
        unsigned long int start;
        start = atol(startthreshold);

        unsigned long int end;
        end = atol(endthreshold);


        timetest = atol(testtimechar);

        // Get local time
        time_t rawtime;
    struct tm * timeinfo;
    time ( &rawtime );
    timeinfo = localtime ( &rawtime );

    // Prepare ROOT data structures for storing data
    printf("Creating ROOT file... \n");
        char rootfilename[256];
        strftime (rootfilename,256,"calibration_%Y%m%d%H%M.root",timeinfo);
        printf("ROOT file name is: %s\n", rootfilename);
    rootfile = new TFile(rootfilename,"RECREATE");
    printf("Creating TNtuple data structure to store values...\n");
```

```
    ntuple = new TNtuple("data","Coincidence rate of channels by detection
threshold","x:a:b:c:d:e");

        // Init serial communication
        int tty;
        tty = setup(filename,baudrate);
        int attempt = 0;

        while (prepare(tty) != 0 ) {
               printf("Retrying preparation...\n");
               attempt++;
               if (attempt > 5) {
                     fprintf(stderr, "Error: preparation of DAQ board failed after 5
attempts. Exiting. \n");
                     return 1;
                     }
               }

    // START of calibration
    int threshold;
    char thresholdcommand[16];
    printf ( "\n [START] Calibration starting at %s \n", asctime(timeinfo));
    printf ("Testing all thresholds for %lu seconds \n \n", timetest);
    for (unsigned int i=start; i < end; i=i+10) {
               // printf("loop \n");
               //printf("looping %i", i);
               threshold = i;
               thresholdtest = threshold;
               printf(" --> Testing threshold %i mV \n",threshold);
               sprintf(thresholdcommand,"TL 4 %i",threshold);
               send(tty,thresholdcommand);
               // Reset scalers
         strcpy(input, "RB");
         send(tty,input);
         // Wait for the board to acquire data
         sleep(timetest);
         // Get data
         strcpy(input, "DS");
         send(tty,input);
         // Parse data and store to NTuple
         process();
        }

        time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    printf ( "\n [STOP] Calibration ended at %s \n", asctime(timeinfo));

    // Display statistics about NTuple saved data
        ntuple->Print();
        ntuple->Write();

        // Prepare canvas
        gStyle->SetOptFit();
    TCanvas *c1 = new TCanvas("c1","Calibration",1000,700);
        c1->Divide(2,2);

        // Draw graph for channel 0
        c1->cd(1);
        ntuple->Draw("a:x","","acp");
        c1->cd(1)->SetTitle("Canal 0");

        // Draw graph for channel 1
        c1->cd(2);
        ntuple->Draw("b:x","","acp");
        c1->cd(2)->SetTitle("Canal 1");

        // Draw graph for channel 2
        c1->cd(3);
        ntuple->Draw("c:x","","acp");
        c1->cd(3)->SetTitle("Canal 2");

        // Draw graph for channel 2
        c1->cd(4);
        ntuple->Draw("d:x","","acp");
        c1->cd(4)->SetTitle("Canal 3");

        rootapp->Run();

        return 0;}
```

Exemple de résultat de l'execution du programme:

```
CosmicCalibrator v0.1-alpha3
Warning in <TApplication::GetOptions>: file /dev/ttyUSB0 has size 0, skipping
Creating ROOT file...
ROOT file name is: calibration_201412072022.root
Creating TNtuple data structure to store values...
Using device /dev/ttyUSB0 with baudrate 115200
Preparing DAQ board for calibration...
Disabling data output [CD]...
Disabling status messages [ST 0]...
Setting 4-fold coincidence count and trigger to all channels [WC 0 3F]...
Setting gate width to 100ns [WC 2 A]...
Test [DS]...
[START] Calibration starting at Sun Dec 7 20:22:11 2014
Testing all thresholds for 300 seconds
--> Testing threshold 10 mV
17613.89
10695.25
14824.44
6405.66
7.16
--> Testing threshold 20 mV
16558.59
9689.56
13841.31
4886.66
7.02
--> Testing threshold 30 mV
15489.58
8726.98
12835.79
3562.24
6.98
[...]
--> Testing threshold 1500 mV
0.00
0.01
0.01
0.01
0.00
[STOP] Calibration ended at Mon Dec 8 08:59:49 2014
*******************************************************************************
*Tree :data : Coincidence rate of channels by detection threshold *
*Entries : 150 : Total = 7564 bytes File Size = 0 *
* : : Tree compression factor = 1.00 *
*******************************************************************************
*Br 0 :x : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
*Br 1 :a : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
*Br 2 :b : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
*Br 3 :c : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
*Br 4 :d : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
*Br 5 :e : Float_t *
*Entries : 150 : Total Size= 1208 bytes One basket in memory *
*Baskets : 0 : Basket Size= 32000 bytes Compression= 1.00 *
*.....*
```

Info in <TCanvas::Print>: ps file c1.ps has been created
Info in <TCanvas::SaveAs>: ROOT file ./c1.root has been created

2)“CosmicDAQ”, programme d’acquisition de données

```
/*
 * Cosmic DAQ program v0.1-alpha5
 * Program to acquire raw data from QuarkNet DAQ cards
 *
 * Programme ecrit pour un projet de Travaux Pratiques Encadres en classe de
 * 1ere Scientifique (1S2 du lycee de Ferney-Voltaire) par Tomasz BIALAS
 *
 * Groupe: Donatella AVONI, Tomasz BIALAS, Nils VAN WEELDEREN
 *
 * Copyright 2014 Tomasz Bialas <tbx1024 at gmail dot com>
 * TPE 2014, Lycee de Ferney Voltaire
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */

#include <stdlib.h>                  // provides basic features
#include <stdio.h>                   // provides basic i/o features
#include <string.h>                  // provides basic string features
#include <fcntl.h>                   // provides file descriptor manipulation
features
#include <errno.h>                   // provides error output features
#include <termios.h>          // provides TTY features
#include <unistd.h>                  // provides    POSIX system API features (read,
write, close)
#include <time.h>                    // provides local time
#include <sstream>                   // provides C++ streams
#include <signal.h>                  // provides system signal handling
#include <sys/types.h>        // provides file info
#include <sys/stat.h>         // provides file info stats

#define _POSIX_SOURCE 1

char input[4096];
char globaloutput[4096];
bool stop = false;
char data[1024];
FILE* logfile;
int angle;
int threshold1;
int threshold2;
int threshold3;
int threshold4;
int lines;


FILE* logopen(char* logfilename) {
        // Open file to save data
        FILE* log;
        log = fopen(logfilename, "a");

        return log;
        }


void writelog(char* input1) {
        // Write data to file
        fprintf(logfile,"%s",input1);
        }
```

```
void process() {
        int s1,s2,s3,s4,s5;
        char* data[8];
        // Parse input string into scalers integers
        data[0] = strtok(globaloutput," ");

    data[1] = strtok(NULL, " ");
        memmove(data[1], data[1]+3, strlen(data[1]));
    s1 = strtol(data[1],0,16);
 //   printf("%d\n", s1);

    data[2] = strtok(NULL, " ");
    memmove(data[2], data[2]+3, strlen(data[2]));
    s2 = strtol(data[2],0,16);
  //  printf("%d\n", s2);

    data[3] = strtok(NULL, " ");
    memmove(data[3], data[3]+3, strlen(data[3]));
    s3 = strtol(data[3],0,16);
  //  printf("%d\n", s3);

    data[4] = strtok(NULL, " ");
    memmove(data[4], data[4]+3, strlen(data[4]));
    s4 = strtol(data[4],0,16);
  //  printf("%d\n", s4);

    data[5] = strtok(NULL, " ");
    memmove(data[5], data[5]+3, strlen(data[5]));
    s5 = strtol(data[5],0,16);
    //printf("%d\n", s5);

    data[6] = strtok(NULL, " ");
    memmove(data[6], data[6]+3, strlen(data[6]));
    data[7] = strtok(NULL, "\0");

        printf("[STOP] Scalers status on DAQ stop: s0=%d s1=%d s2=%d s3=%d s4=%d\n",
s1,s2,s3,s4,s5);
        printf("[STOP] Recorded %d events\n",s5);
        char scalerlog[80];
        sprintf(scalerlog, "SCALERS %d %d %d %d %d\n", s1,s2,s3,s4,s5);
        writelog(scalerlog);

        }

void usrterminate(int i) {
        // Stop data reading
        printf("\nReceived interruption signal. Terminating DAQ... \n");
        stop = true;
        }
void send(int tty,char* input){
        char output[4096];
        strcpy(output,"");
        char RX[1024];
        strcpy(RX,"");
        int status = 0;
        for (unsigned int i=0; i<strlen(input); i++){
                // printf("Sending '%c' \n",input[i]);
                status = write(tty,&input[i],1);
                        if (status < 0) {
                                fprintf(stderr,"Error occured while writing to serial
device. Exiting. \n");
                                exit(1);
                        }
                usleep(50);
        }
        write(tty,"\r",1);
        sleep(1);
        while (read(tty,&RX,1) > 0) {
                //printf("|%s",data);
                for (unsigned int y=0; y<strlen(RX); y++){
                        sprintf(output, "%s%s", output, &RX[y]);
                }

        }

//      printf("[DEBUG] DATA OUTPUT IS: %s \n",output);
        strcpy(globaloutput, output);
```

```
        }

int prepare(int tty){
        char flooddata[16384];
        int attempt;
        attempt=0;
        // Prepare board for data acquisition
    printf("Preparing DAQ board for data acquisition...\n");
    strcpy(input,"");
    send(tty,input);
    printf("Disabling data output [CD]... \n");
    strcpy(input,"CD");
    send(tty,input);
    while (read(tty,&flooddata,16384) > 0 && attempt > 6) {
                        printf("Received data when not expected, retrying... \n");
                        send(tty,input);
                        sleep(5);
                        attempt++;
                }
    printf("Disabling status messages [ST 0]... \n");
    strcpy(input, "ST 0");
    send(tty,input);
    printf("Setting 4-fold coincidence count and trigger to all channels [WC 0 3F]...
\n");
    strcpy(input, "WC 0 3F");
    send(tty,input);
    printf("Setting gate width to 100ns [WC 2 A]... \n");
    strcpy(input, "WC 2 A");
    send(tty,input);
        printf("Setting threshold for channel 0 to %i mV ... \n", threshold1); // 300
    sprintf(input, "TL 0 %i", threshold1);
    send(tty,input);
        printf("Setting threshold for channel 1 to %i mV ... \n", threshold2); // 300
    sprintf(input, "TL 1 %i", threshold2);
    send(tty,input);
    printf("Setting threshold for channel 2 to %i mV ... \n", threshold3); // 280
    sprintf(input, "TL 2 %i", threshold3);
    send(tty,input);
    printf("Setting threshold for channel 3 to %i mV ... \n", threshold4); // 150
    sprintf(input, "TL 3 %i", threshold4);
    send(tty,input);
    printf("Enabling status output every minute... [ST 2 1] \n");
    strcpy(input, "ST 2 1");
    send(tty,input);
        printf("Test [DS]... \n");
    strcpy(input, "DS");
    send(tty,input);

        return 0;
        }




int setup(char* filename, int baudrate){
        // Open serial device
        printf("Using device %s with baudrate %i \n",filename,baudrate);
        int tty;
        tty=open(filename, O_RDWR | O_NONBLOCK);
        if(tty < 0) {
                fprintf(stderr,"Error while opening device %s. Exiting. \n",filename);
                exit(1);
                }

        // Setup serial terminal settings
        struct termios terminal;
        memset(&terminal,0,sizeof(terminal));
    terminal.c_cflag=CSTOPB|PARENB|IXON|IXOFF|HUPCL;
    terminal.c_iflag &= ~(ICRNL|IGNCR);
    terminal.c_oflag=0;
    terminal.c_cflag=CS8|CREAD|CLOCAL;
    terminal.c_lflag=0;
    terminal.c_cc[VMIN]=1;
    terminal.c_cc[VTIME]=5;

    // Baudrate setup - check for compatibility with supported DAQ baudrates and set
baudrate if compatible
```

```
        bool compatible_baudrate = false;
        if (baudrate == 19200) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B19200);
                cfsetospeed(&terminal,B19200);
                }
        if (baudrate == 38400) {
                compatible_baudrate = true;
                cfsetospeed(&terminal,B38400);
                cfsetospeed(&terminal,B38400);
                }
        if (baudrate == 57600) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B57600);
                cfsetospeed(&terminal,B57600);
                }
        if (baudrate == 115200) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B115200);
                cfsetospeed(&terminal,B115200);
                }
        if (baudrate == 230400) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B230400);
                cfsetospeed(&terminal,B230400);
                }
        if (baudrate == 460800) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B460800);
                cfsetospeed(&terminal,B460800);
                }
        if (baudrate == 921600) {
                compatible_baudrate = true;
                cfsetispeed(&terminal,B921600);
                cfsetospeed(&terminal,B921600);
                }

        // If the baudrate is unsupported, exit.
        if (compatible_baudrate == false) {
                fprintf(stderr,"Error: the specified baudrate is not supported by the
DAQ board. Exiting. \n");
                exit(1);
                }

    if (tcsetattr(tty,TCSANOW,&terminal) < 0) {
                fprintf(stderr,"Error: failed to setup serial port. Exiting. \n");
                exit(1);
         }

    tcflush(tty, TCIOFLUSH);
        return tty;
        }



int main(int argc, char **argv)
{
        printf("CosmicDAQ v0.1-alpha5 \n");
        // Check command line arguments
        if (argc != 9) {
                printf("Usage: cosmicdaq <device_name> <baudrate> <data filename>
<threshold 1> <threshold 2> <threshold 3> <threshold 4> <angle> \n");
                fprintf(stderr,"Error: wrong number of arguments specified. Exiting.
\n");
                exit(1);
                }

        // Parse command line arguments
        char filename[512];
        strcpy(filename,argv[1]);
        char baudrate_arg[128];
        strcpy(baudrate_arg,argv[2]);
        char logfilename[512];
        strcpy(logfilename, argv[3]);
        char threshold1char[10];
        strcpy(threshold1char, argv[4]);
        char threshold2char[10];
        strcpy(threshold2char, argv[5]);
        char threshold3char[10];
```

```
        strcpy(threshold3char, argv[6]);
        char threshold4char[10];
        strcpy(threshold4char, argv[7]);
        char anglechar[10];
        strcpy(anglechar,argv[8]);

        // Parse baudrate value
        long int baudrate;
        baudrate = atol(baudrate_arg);

        // Parse measurement angle and thresholds
        angle = atol(anglechar);

        threshold1 = atol(threshold1char);
        threshold2 = atol(threshold2char);
        threshold3 = atol(threshold3char);
        threshold4 = atol(threshold4char);

        // Get local time
        time_t rawtime;
    struct tm * timeinfo;
    time ( &rawtime );
    timeinfo = localtime ( &rawtime );

        // Init serial communication
        int tty;
        logfile = logopen(logfilename);
        tty = setup(filename,baudrate);
        int attempt = 0;

        while (prepare(tty) != 0 ) {
                printf("Retrying preparation...\n");
                attempt++;
                if (attempt > 5) {
                        fprintf(stderr, "Error: preparation of DAQ board failed after 5
attempts. Exiting. \n");
                        return 1;
                        }
                }

        printf ( "\n[START] DAQ starting at %s \n", asctime(timeinfo));

        // Log start timestamp
        char timelog[80];
        strftime(timelog,80,"START %Y %m %d %H %M %S \n",timeinfo);
        writelog(timelog);

        // Log threshold values
        char thresholdlog[80];
        sprintf(thresholdlog, "THRESHOLD %i %i %i %i \n", threshold1, threshold2,
threshold3, threshold4);
        writelog(thresholdlog);

        // Log angle value
        char anglelog[80];
        sprintf(anglelog, "ANGLE %i \n", angle);
        writelog(anglelog);



        signal(SIGINT,usrterminate); // handler for Ctrl+C

        // Read data and count lines saved
        char newline[1];
        strcpy(newline, "\n");
        printf("Resetting scalers [RB]... \n");
    strcpy(input, "RB");
    send(tty,input);
    // Enable data
        printf("Enabling card data output [CE]... \n");
    strcpy(input, "CE");
    send(tty,input);

        while (stop==false) {
                while (read(tty,&data,1) > 0) {
                        writelog(data);
                        if (strcmp(data, newline) == 0){
                                lines++;
                                }
```

```
                }
        }

        // Disable data
        printf("Disabling card data output [CD]... \n");
    strcpy(input, "CD");
    send(tty,input);

        while (read(tty,&data,1) > 0) {
         writelog(data);
                        if (strcmp(data, newline) == 0){
                                lines++;
                                }
                }

        printf("Reading scalers data... \n");
    strcpy(input, "DS");
    send(tty,input);
    sleep(1);

    process();

        // Log end timestamp
        time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    strftime(timelog,80,"\nSTOP %Y %m %d %H %M %S \n",timeinfo);
        writelog(timelog);

        // Print data statistics
        struct stat filestat;
        stat(logfilename, &filestat);
        printf ( "\n[STOP] DAQ ended at %s", asctime(timeinfo));
        printf("[STOP] %i lines, %lu bytes data recorded. \n", lines,
filestat.st_size);

        // Close file
    fclose(logfile);

        return 0;
}
```

Exemple de l'execution du programme:

```
CosmicDAQ v0.1-alpha1
Using device /dev/ttyUSB0 with baudrate 115200
Preparing DAQ board for data acquisition...
Disabling data output [CD]...
Disabling status messages [ST 0]...
Setting 4-fold coincidence count and trigger to all channels [WC 0 3F]...
Setting gate width to 100ns [WC 2 A]...
Setting threshold for channel 0 to 300 mV ...
Setting threshold for channel 1 to 300 mV ...
Setting threshold for channel 2 to 280 mV ...
Setting threshold for channel 3 to 150 mV ...
Test [DS]...
[START] DAQ starting at Fri Dec 19 23:06:58 2014
Enabling card data output [CE]...
^C
Received interruption signal. Terminating DAQ...
Disabling card data output [CD]...
[STOP] DAQ ended at Fri Dec 19 23:07:30 2014
[STOP] 330 lines, 20480 bytes data recorded.
```

Fichier avec de donnes « raw data » obtenu:

```
START 2014 12 19 23 06 58
THRESHOLD 300 300 280 150
ANGLE 50
01C8AD42 B2 00 36 00 33 00 35 00 004CA816 000000.000 000000 V 00 8 0000
01C8AD43 00 26 00 00 00 00 00 00 004CA816 000000.000 000000 V 00 8 0000
01C8AD43 2E 00 00 00 00 2A 00 2F 004CA816 000000.000 000000 V 00 0 0000
01C8AD43 00 34 00 30 00 00 00 00 004CA816 000000.000 000000 V 00 0 0000
025B4731 BA 00 3E 00 3C 00 3E 00 01CA2056 000000.000 000000 V 00 0 0000
025B4732 00 00 00 00 00 00 00 26 01CA2056 000000.000 000000 V 00 0 0000
025B4732 00 2F 00 00 00 00 00 00 01CA2056 000000.000 000000 V 00 0 0000
025B4732 00 00 00 36 00 30 00 00 01CA2056 000000.000 000000 V 00 0 0000
02C65899 AE 00 00 00 00 00 00 00 01CA2056 000000.000 000000 V 00 0 0000
```

```
02C65899 00 00 30 00 36 00 37 00 01CA2056 000000.000 000000 V 00 0 0000
02C6589A 00 00 00 00 00 27 00 24 01CA2056 000000.000 000000 V 00 0 0000
02C6589A 00 2F 00 00 00 00 00 00 01CA2056 000000.000 000000 V 00 0 0000
02C6589A 00 00 00 34 00 00 00 00 01CA2056 000000.000 000000 V 00 0 0000
 [...]
10081B8A 00 23 00 00 00 00 00 00 0F335A96 000000.000 000000 V 00 0 0000
10081B8A 00 00 00 29 00 00 00 00 0F335A96 000000.000 000000 V 00 0 0000
10CFB5F9 A3 00 25 00 22 00 25 00 10B0D2D6 000000.000 000000 V 00 0 0000
10CFB5F9 00 37 00 00 00 37 00 00 10B0D2D6 000000.000 000000 V 00 0 0000
10CFB5F9 00 00 00 3D 00 00 00 39 10B0D2D6 000000.000 000000 V 00 0 0000
10DA56E6 A5 00 26 00 24 00 00 00 10B0D2D6 000000.000 000000 V 00 0 0000
10DA56E6 00 2F 00 00 00 00 2C 00 10B0D2D6 000000.000 000000 V 00 0 0000
1F330D4C BD 00 00 00 00 00 00 00 1E1A0D16 000000.000 000000 V 00 0 0000
1F330D4D 00 00 00 00 22 00 25 00 1E1A0D16 000000.000 000000 V 00 0 0000
1F330D4D 00 00 2F 00 00 00 00 00 1E1A0D16 000000.000 000000 V 00 0 0000
1F330D4D 00 34 00 31 00 00 00 00 1
STOP 2014 12 19 23 07 30
```

3)“CosmicInterpreter”, programme d’analyse de “raw data”, transformations vers fichier ROOT:

```
/*
 * cosmicinterpreter.cxx
 *
 * CE PROGRAMME N'EST PAS FINI AU MOMENT DE L'IMPRESSION DU DOCUMENT
 *
 * Programme ecrit pour un projet de Travaux Pratiques Encadres en classe de
 * 1ere Scientifique (1S2 du lycee de Ferney-Voltaire) par Tomasz BIALAS
 *
 * Groupe: Donatella AVONI, Tomasz BIALAS, Nils VAN WEELDEREN
 *
 * Copyright 2014 Tomasz Bialas <tbx1024 at gmail dot com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 *
 *
 */

#include <stdio.h>                    // provides basic I/O features
#include <iostream>                   // provides I/O C++ streans
#include <string.h>                   // provides strings
#include <stdlib.h>                   // provides basic features
#include <time.h>                     // provides local time
#include <sstream>                    // provides C++ streams
#include <TFile.h>                    // CERN ROOT TFile
#include <TNtuple.h>          // CERN ROOT NTuple
#include <TH2.h>                      // CERN ROOT 2D graphing functions
#include <TString.h>          // CERN ROOT TString
#include <Riostream.h>        // C++ iostreams
#include <TTree.h>
#include <TSystem.h>          // CERN ROOT TSystem for accessing underlying operating
system
#include <TApplication.h>     // CERN ROOT TApplication for standalone out-of-
interpreter running
#include "MuonEvent.h"


#define FLAG_RISINGEDGE 32
#define FLAG_FALLINGEDGE 32
#define FLAG_NEWEVENT 128

time_t starttime;
time_t stoptime;
```

```
int angle;
char threshold[3];
int totalcounts = 0;
int errorcount = 0;
int eventcount = 0;
int clk10ns = 0;
int clk40ns = 0;
int event_clk40ns = 0;
TFile *rootfile;
TNtuple *ntuple;
MuonEvent *event;
int debug = 0;
int zenit_angle = 0;

Bool_t inEventTask = false ;

void decode(char input[],TTree& tree) {
       /* Read line, parse string and output human-readable test to STDOUT */

       char* data[16];
    long int processeddata[16];
       data[0] = strtok(input," ");
       if (strcmp(data[0],"DS") == 0){
              if (debug) printf("Found scaler status line [DS] \n");
       }
       else if (strcmp(data[0],"START") == 0){
              if (debug) printf("Found start time tag \n");
       }
       else if (strcmp(data[0],"ANGLE") == 0){
              if (debug) printf("Found start angle tag \n");
              }

       else if (strcmp(data[0],"ST") == 0){
              if (debug) printf("Found status line [ST] \n");
       }
       else if (strcmp(data[0],"THRESHOLD") == 0){
              if (debug) printf("Found start threshold tag \n");
       }
       else if (strcmp(data[0],"SCALERS") == 0){
              if (debug) printf("Found stop scaler status tag\n");
       }
       else if (strcmp(data[0],"STOP") == 0){
              if (debug) printf("Found stop time tag \n");
       }
       else if (strlen(data[0]) == 8) {
              //eventcount++;
              if (debug) printf("Found data line \n");
              data[1] = strtok(NULL, " ");
              data[2] = strtok(NULL, " ");
              data[3] = strtok(NULL, " ");
              data[4] = strtok(NULL, " ");
              data[5] = strtok(NULL, " ");
              data[6] = strtok(NULL, " ");
              data[7] = strtok(NULL, " ");
              data[8] = strtok(NULL, " ");
              data[9] = strtok(NULL, " ");
              data[10] = strtok(NULL, " ");
              data[11] = strtok(NULL, " ");
              data[12] = strtok(NULL, " ");
              data[13] = strtok(NULL, " ");
              data[14] = strtok(NULL, " ");
              data[15] = strtok(NULL, "\0");

              /* Debugging output */
              /*
              printf("\n NEWLINE \n");
              printf("data 0 = %s \n",data[0]);
              printf("data 1 = %s \n",data[1]);
              printf("data 2 = %s \n",data[2]);
              printf("data 3 = %s \n",data[3]);
              printf("data 4 = %s \n",data[4]);
              printf("data 5 = %s \n",data[5]);
              printf("data 6 = %s \n",data[6]);
              printf("data 7 = %s \n",data[7]);
              printf("data 8 = %s \n",data[8]);
              printf("data 9 = %s \n",data[9]);
              printf("data 10 = %s \n",data[10]);
              printf("data 11 = %s \n",data[11]);
              printf("data 12 = %s \n",data[12]);
```

```
                printf("data 13 = %s \n",data[13]);
                printf("data 14 = %s \n",data[14]);
                printf("data 15 = %s \n",data[15]);
                */

                processeddata[0] = strtol(data[0],0,16); // Word 1
                processeddata[1] = strtol(data[1],0,16); // Word 2
                processeddata[2] = strtol(data[2],0,16); // Word 3
                processeddata[3] = strtol(data[3],0,16); // Word 4
                processeddata[4] = strtol(data[4],0,16); // Word 5
                processeddata[5] = strtol(data[5],0,16); // Word 6
                processeddata[6] = strtol(data[6],0,16); // Word 7
                processeddata[7] = strtol(data[7],0,16); // Word 8
                processeddata[8] = strtol(data[8],0,16); // Word 9
                processeddata[9] = strtol(data[9],0,16); // Word 10

                // processeddata[10] =
        if (debug) {
                printf("data 0 = %ld\n",processeddata[0]);
                printf("data 1 = %ld\n",processeddata[1]);
                printf("data 2 = %ld\n",processeddata[2]);
                printf("data 3 = %ld\n",processeddata[3]);
                printf("data 4 = %ld\n",processeddata[4]);
                printf("data 5 = %ld\n",processeddata[5]);
                printf("data 6 = %ld\n",processeddata[6]);
                printf("data 7 = %ld\n",processeddata[7]);
                printf("data 8 = %ld\n",processeddata[8]);
                printf("data 9 = %ld\n",processeddata[9]);
                printf("data 10 = %s \n",data[10]);
                printf("data 11 = %s \n",data[11]);
                printf("data 12 = %s \n",data[12]);
                printf("data 13 = %s \n",data[13]);
                printf("data 14 = %s \n",data[14]);
                printf("data 15 = %s \n",data[15]);
                }
                //clk10ns++; // increment 10ns clock time keeper

                        if (processeddata[1] & FLAG_NEWEVENT) { // this data line is a
start of new event
                        if (inEventTask) { // end of processing of previous event:
                                if (event->isValid()) { // fill only event that was
considered as OK
                                if (debug) printf("Filling TTree with event.\n");
                                tree.Fill(); // filling TTree with the event
                                }
                                event->Clear(); // preparing event placeholder for new
processing ie reseting values
                                inEventTask = false; // mark that prcessing is finished
                        }
                        inEventTask = true; // flaging that we start event processing
                          eventcount++; // trigger count
                        clk10ns = 0; // reset 10 ns clock time keeper - this is first
word of event data
                        clk40ns = 0 ; // reset 40ns clock time keeper
                        event_clk40ns = processeddata[0];
                        if (debug) printf("FOUND EVENT %d START FLAG \n",eventcount);
                        if (data[10] && data[11]) { // assure that inputs fields are
there
                        event->setTime(data[10],data[11]); // process GPS date and time
                        // for time being validate event just with time and date inside
                        event->setValid(true);
                        }
                 }

                if (inEventTask) { //  line inside event - processing data
                        clk40ns = processeddata[0] - event_clk40ns ;
                        event->processRiseFallData(clk40ns,

        processeddata[1],processeddata[2],processeddata[3],processeddata[4],

        processeddata[5],processeddata[6],processeddata[7],processeddata[8]);
                }


        }
        else if(strlen(data[0]) == 1){
                if (debug) printf("Found empty line \n");
                }
        else {
```

```
                if (debug) printf("Error: line could not be identified! \n");
                errorcount++;
                }

}

FILE *openfile(char *input) {
        /* Open file */
        FILE *localdatafile;
        printf("Reading from file %s\n",input);
        localdatafile = fopen(input,"r");
        if (localdatafile == NULL) {
                fprintf(stderr,"Error: error occurred while reading file. Exiting.
\n");
                exit(1);
                }
        return localdatafile;

        }

void process(FILE *input,TTree& tree) {
        /* Read a line from the file */
        char currentline[256];
        int i;
        i=0;
        while(fgets(currentline, sizeof(currentline), input)) {
         /* printf(currentline); */
         decode(currentline,tree);


         i++;
                }

        // finish of event under treatment:
        if (inEventTask) { // end of processing of previous event:
                if (event->isValid()) { // fill only event that were considered as OK
                        if (debug) printf("Filling TTree with event.\n");
                        tree.Fill(); // filling TTree with the event
                }
         event->Clear(); // overkill
        inEventTask = false ;
         }


        printf("Processed %i lines \n", i);
        printf("%i error(s) occured \n",errorcount);
        //*printf("\n");
        }


int main(int argc, char **argv) {
        printf("CosmicInterpreter v0.0-alpha3 \n");

        /* Check for number of arguments (1+itself) */
        if (argc != 3) {
                printf("Usage: cosmicinterpreter <data filename> <ROOT output filename>
\n");
                fprintf(stderr,"Error: wrong number of arguments specified. Exiting.
\n");
                exit(1);
                }

        /* Parse filename from argument */
        char filename[512];
        strcpy(filename,argv[1]);


        char rootfilename[512];
        strcpy(rootfilename,argv[2]);

        /* Open file */
        FILE *datafile;
        datafile = openfile(filename);

        printf("Preparing the ROOT subsystem... \n");
        TApplication* rootapp = new TApplication("CosmicInterpreter",&argc, argv);
        rootapp->SetReturnFromRun(true);

        printf("Creating ROOT file... \n");
```

```
        rootfile = new TFile(rootfilename,"RECREATE");
        printf("ROOT file name is: %s\n", rootfilename);

        event = new MuonEvent();

        TTree MuonTree("MuonTree","Tree containing MuonEvent muon detection event
data");
        MuonTree.Branch("event", "MuonEvent", &event,16000);

        // Get local time
        time_t rawtime;
    struct tm * timeinfo;
    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
        printf ( "\n [START] Data analysis starting at %s \n", asctime(timeinfo));
        /* Read file line by line, process, output, store and analyse the data*/


        process(datafile,MuonTree);
        MuonTree.Write();
        MuonTree.Print();

        rootfile->Close();

        //rootapp->Run();

        return 0;
}
```

Example de execution:

```
CosmicInterpreter v0.0-alpha3
Reading from file test1.data
Preparing the ROOT subsystem...
Creating ROOT file...
ROOT file name is: test1sp.root

 [START] Data analysis starting at Thu Jan 15 13:20:02 2015

Processed 5336 lines
1 error(s) occured
******************************************************************************
*Tree    :MuonTree  : Tree containing MuonEvent muon detection event data    *
*Entries :     1288 : Total =          258810 bytes  File  Size =      48131 *
*        :          : Tree compression factor =   5.37                       *
******************************************************************************
*Branch  :event                                                               *
*Entries :     1288 : BranchElement (see below)                               *
*.............................................................................*
*Br    0 :fUniqueID : UInt_t                                                  *
*Entries :     1288 : Total  Size=       5734 bytes  File Size  =        134 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.04     *
*.............................................................................*
*Br    1 :fBits     : UInt_t                                                  *
*Entries :     1288 : Total  Size=      10874 bytes  File Size  =       1752 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=   5.93     *
*.............................................................................*
*Br    2 :EventTime :                                                         *
*Entries :     1288 : Total  Size=       1599 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      16000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    3 :EventTime.fSec : Int_t                                              *
*Entries :     1288 : Total  Size=       5759 bytes  File Size  =        989 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=   5.30     *
*.............................................................................*
*Br    4 :EventTime.fNanoSec : Int_t                                          *
*Entries :     1288 : Total  Size=       5779 bytes  File Size  =        143 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  36.65     *
*.............................................................................*
*Br    5 :ch0rise   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23958 bytes  File Size  =       5238 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.48     *
*.............................................................................*
*Br    6 :ch0fall   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23922 bytes  File Size  =       5275 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.44     *
*.............................................................................*
*Br    7 :ch1rise   : vector<Int_t>                                           *
```

```
*Entries :     1288 : Total  Size=      23962 bytes  File Size  =       5264 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.46     *
*.............................................................................*
*Br    8 :ch1fall   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23878 bytes  File Size  =       5263 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.44     *
*.............................................................................*
*Br    9 :ch2rise   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23946 bytes  File Size  =       5204 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.51     *
*.............................................................................*
*Br   10 :ch2fall   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23930 bytes  File Size  =       5254 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.46     *
*.............................................................................*
*Br   11 :ch3rise   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23950 bytes  File Size  =       5226 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.49     *
*.............................................................................*
*Br   12 :ch3fall   : vector<Int_t>                                           *
*Entries :     1288 : Total  Size=      23926 bytes  File Size  =       5228 *
*Baskets :        2 : Basket Size=      16000 bytes  Compression=   4.48     *
*.............................................................................*
*Br   13 :tot_ch0   : Int_t                                                   *
*Entries :     1288 : Total  Size=       5724 bytes  File Size  =        134 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.03     *
*.............................................................................*
*Br   14 :tot_ch1   : Int_t                                                   *
*Entries :     1288 : Total  Size=       5724 bytes  File Size  =        134 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.03     *
*.............................................................................*
*Br   15 :tot_ch2   : Int_t                                                   *
*Entries :     1288 : Total  Size=       5724 bytes  File Size  =        133 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.32     *
*.............................................................................*
*Br   16 :tot_ch3   : Int_t                                                   *
*Entries :     1288 : Total  Size=       5724 bytes  File Size  =        133 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.32     *
*.............................................................................*
*Br   17 :tot_event : Int_t                                                   *
*Entries :     1288 : Total  Size=       5734 bytes  File Size  =        136 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  38.47     *
*.............................................................................*
*Br   18 :energy    : Int_t                                                   *
*Entries :     1288 : Total  Size=       5719 bytes  File Size  =        133 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  39.32     *
*.............................................................................*
*Br   19 :valid     : Bool_t                                                  *
*Entries :     1288 : Total  Size=       1850 bytes  File Size  =        107 *
*Baskets :        1 : Basket Size=      16000 bytes  Compression=  12.75     *
*.............................................................................*
```

3)“CosmicInterpreter”, definition class MuonEvent (C++):

MuonEvent.h :

```
/*
 * Composante du programme CosmicInterpreter
 *
 * MuonEvent, objet definissant les informations d'une evenement
 *
 *  Programme ecrit pour un projet de Travaux Pratiques Encadres en classe de
 * 1ere Scientifique (1S2 du lycee de Ferney-Voltaire) par Tomasz BIALAS
 *
 * Groupe: Donatella AVONI, Tomasz BIALAS, Nils VAN WEELDEREN
 *
 * Copyright 2014 Tomasz Bialas <tbx1024 at gmail dot com>
 * TPE 2014, Lycee de Ferney Voltaire
 *
 */

#ifndef _MuonEvent_h
#define _MuonEvent_h

#include "Rtypes.h"
#include "TObject.h"
#include "TTimeStamp.h"
#include "TVector.h"
#include "stdlib.h"
```

```
//class RunInfo : public TObject {
        //private:
                //time_t start;
                //time_t end;
                //int angle;
                //int threshold_ch0;
                //int threshold_ch1;
                //int threshold_ch2;
                //int threshold_ch3;
                //int scaler_ch0;
                //int scaler_ch1;
                //int scaler_ch2;
                //int scaler_ch3;
                //int EventCount;
//};

class MuonEvent : public TObject {
        private:
                TTimeStamp              EventTime;
                std::vector<Int_t>              ch0rise;
                std::vector<Int_t>      ch0fall;
         std::vector<Int_t>      ch1rise;
         std::vector<Int_t>      ch1fall;
         std::vector<Int_t>      ch2rise;
         std::vector<Int_t>      ch2fall;
         std::vector<Int_t>      ch3rise;
         std::vector<Int_t>      ch3fall;
                Int_t                   tot_ch0;
                Int_t                   tot_ch1;
                Int_t                   tot_ch2;
                Int_t                   tot_ch3;
                Int_t                   tot_event; // ns
                Int_t                   energy;
                Bool_t                  valid;
                Int_t   debug; //!

        public:
                MuonEvent();
                virtual ~MuonEvent();

                void Clear() {
                tot_ch0 = 0; tot_ch1 = 0; tot_ch1 = 0; tot_ch1 = 0;
                tot_event = 0; energy = 0; EventTime.Set(0,0,0,1,0);
                valid = false;
                ch0rise.clear(); ch1rise.clear(); ch2rise.clear(); ch3rise.clear();
                ch0fall.clear(); ch1fall.clear(); ch2fall.clear(); ch3fall.clear();
                }

                void setTime(Char_t *input, Char_t *date) {

                Char_t year[5];
                Char_t month[3];
                Char_t day[3];
                Char_t hour[3];
                Char_t min[3];
                Char_t sec[3];
                Char_t ms[4];

                strncpy(year,date+4,2); year[2] = 0;
                strncpy(month,date+2,2); month[2] = 0;
                strncpy(day,date,2);   day[2] = 0;
                strncpy(hour,input,2); hour[2] = 0;
                strncpy(min,input+2,2);        min[2] = 0;
                strncpy(sec,input+4,2);        sec[2] = 0;
                strncpy(ms,input+7,3); ms[3] = 0;
                if (debug) {
                printf("%s\n",year);
                printf("%s\n",month);
                printf("%s\n",day);
                printf("%s\n",hour);
                printf("%s\n",min);
                printf("%s\n",sec);
                }
                Int_t year_int = atol(year)+2000;
                Int_t month_int = atol(month);
                Int_t day_int = atol(day);
                Int_t hour_int = atol(hour);
```

```
                Int_t min_int = atol(min);
                Int_t sec_int = atol(sec);
                if (debug) {
                printf("%i\n",year_int);
                printf("%i\n",month_int);
                printf("%i\n",day_int);
                printf("%i\n",hour_int);
                printf("%i\n",min_int);
                printf("%i\n",sec_int);
                }
                EventTime =
TTimeStamp(year_int,month_int,day_int,hour_int,min_int,sec_int);
                if (debug) printf("EventTime.fSec: %ld\n",EventTime.GetSec());

                }


                void setTOT(Int_t ch0,Int_t ch1, Int_t ch2, Int_t ch3) {

                }

                Int_t getEnergy() {
                return 0;
                }

                Bool_t getValid() {
                return valid ;
                }

                void setValid(Bool_t val) {
                valid = val ;
                }

                Bool_t isValid() {
                 return valid ;
                 }

                TTimeStamp& getEventTime() {
                        return EventTime;
                }

                void processRiseFallData(int clk10ns,
                        int ch0r, int ch0f,
                        int ch1r, int ch1f,
                        int ch2r, int ch2f,
                        int ch3r, int ch3f);

                ClassDef(MuonEvent,1);


        };

#endif
```

Fichier MuonEvent.cxx:

```
#include "RVersion.h"
#include "MuonEvent.h"

#include <iostream>

ClassImp(MuonEvent)

#define FLAG_RISINGEDGE 0x020
#define FLAG_FALLINGEDGE 0x020
#define TOT_BITS_MASK 0x01F
//#define TOT_UNITS_PER_CLK_NS 8
#define TOT_UNITS_PER_CLK_NS 32

MuonEvent::MuonEvent(){debug = 0;}
MuonEvent::~MuonEvent(){ Clear();}

void MuonEvent::processRiseFallData(int clk_ns,
                        int ch0r, int ch0f,
                        int ch1r, int ch1f,
                        int ch2r, int ch2f,
                        int ch3r, int ch3f) {
```

```
int clkTotUnits = clk_ns*TOT_UNITS_PER_CLK_NS;

if (debug) {
std::cout << "MuonEvent::processRiseFallData => " << clk_ns << " " ;
std::cout << ch0r << " " << ch0f << " " << ch1r << " " << ch1f << " ";
std::cout << ch2r << " " << ch2f << " " << ch3r << " " << ch3f << std::endl;
}


if (ch0r & FLAG_RISINGEDGE ) { ch0rise.push_back(clkTotUnits + (ch0r &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed rise ch0 bits: " << (ch0r & TOT_BITS_MASK)<<
std::endl ;};
if (ch1r & FLAG_RISINGEDGE ) { ch1rise.push_back(clkTotUnits + (ch1r &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed rise ch1 bits: " << (ch1r & TOT_BITS_MASK)<<
std::endl ;}
if (ch2r & FLAG_RISINGEDGE ) { ch2rise.push_back(clkTotUnits + (ch2r &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed rise ch2 bits: " << (ch2r & TOT_BITS_MASK)<<
std::endl ;}
if (ch3r & FLAG_RISINGEDGE ) { ch3rise.push_back(clkTotUnits + (ch3r &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed rise ch3 bits: " << (ch3r & TOT_BITS_MASK)<<
std::endl ;}

//if ((ch0f & FLAG_FALLINGEDGE ) && (!ch0rise->size()%2)) ch0fall-
>push_back(clkTotUnits + (ch0r & TOT_BITS_MASK));

if (ch0f & FLAG_FALLINGEDGE ) { ch0fall.push_back(clkTotUnits + (ch0f &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed fall ch0 bits: " << (ch0f & TOT_BITS_MASK)<<
std::endl ;}
if (ch1f & FLAG_FALLINGEDGE ) { ch1fall.push_back(clkTotUnits + (ch1f &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed fall ch1 bits: " << (ch1f & TOT_BITS_MASK)<<
std::endl ;}
if (ch2f & FLAG_FALLINGEDGE ) { ch2fall.push_back(clkTotUnits + (ch2f &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed fall ch2 bits: " << (ch2f & TOT_BITS_MASK)<<
std::endl ;}
if (ch3f & FLAG_FALLINGEDGE ) { ch3fall.push_back(clkTotUnits + (ch3f &
TOT_BITS_MASK));
       if (debug) std::cout << "pushed fall ch3 bits: " << (ch3f & TOT_BITS_MASK)<<
std::endl ;}

}
```

4)Les macro du ROOT (C++) , creation des histogrammes et graphs:

fichier printTimeLine.C:

```
TH1F* printTimeLine(Int_t timeStep,char *fName = "test1.root") {
// T.Bialas 2014
//   TDatime da(2003,02,28,12,00,00);
//   gStyle->SetTimeOffset(da.Convert());
// ct = new TCanvas("ct","Time on axis",0,0,600,600);

TFile *f = TFile::Open(fName,"READ");
if (!f) return 0;
TTree *myTree = (TTree*)f->Get("MuonTree");
if (!myTree) return 0;
//MuonEvent *event = new MuonEvent();
Int_t  EventTime_fSec;
myTree->SetMakeClass(1);
TBranch  *b_event_EventTime_fSec;
myTree->SetBranchAddress("EventTime.fSec", &EventTime_fSec, &b_event_EventTime_fSec);
//myTree->SetBranchAddress("EventTime.fSec",&EventTime_fSec);
int nentries = myTree->GetEntries();
cout << "number of events: " << nentries << endl ;
// calculate time span of measurements:
myTree->GetEntry(0);
Int_t startTime = EventTime_fSec;
cout << "Start time: " << startTime << endl ;
myTree->GetEntry(nentries-1);
Int_t stopTime = EventTime_fSec;
cout << "Stop time: " << stopTime << endl ;
```

```
Int_t timeSpan = stopTime - startTime ; // in seconds
cout << "Time span [sec]: " << timeSpan << endl ;
//Float_t nFBins = (Float_t) timeSpan / (Float_t) timeStep ;
//cout << "Number of periods: " << nFBins << endl ;
Int_t nBins = (timeSpan / timeStep) + 1;
cout << "nBins: " << nBins << endl ;
TH1F *hTimeLine = 0;

if (nBins>2)
        hTimeLine = new TH1F("hTimeLine","Measurements rate vs.
time",nBins,startTime,stopTime);
else
        return hTimeLine;

//f->Close();
//delete f;
// // delete event;
//return hTimeLine;

hTimeLine->GetXaxis()->SetTimeDisplay(1);
hTimeLine->GetXaxis()->SetTimeFormat("%d%b-%Hh");


int i = 0;
for ( i=0; i < nentries ; i++ ) {
        myTree->GetEntry(i);
        //cout << i << " " ;
        //cout << (int) event->getEventTime().GetSec() << endl ;
        //int timeStamp = event->getEventTime().GetSec();
        hTimeLine->Fill(EventTime_fSec);
        }

hTimeLine->Scale(1.0/timeStep);
hTimeLine->SetAxisRange(0., 10.0,"Y");
hTimeLine->GetXaxis()->SetTitle("Time");
hTimeLine->GetYaxis()->SetTitle("Muon rate [imp/sec]");
hTimeLine->SetStats(0);

/*f->Close();
delete f;*/
return (TH1F*) hTimeLine ;

}
```

fichier printTotDistro.C:

```
TH1F* printTotDistro(Int_t chan,Int_t zenith_angle, char *fName) {
// T.Bialas 2014


TFile *f = TFile::Open(fName,"READ");
if (!f) return 0;
TTree *myTree = (TTree*)f->Get("MuonTree");
if (!myTree) return 0;
//MuonEvent *event = new MuonEvent();
Int_t  EventTime_fSec;
   vector<Int_t>   ch0rise;
   vector<Int_t>   ch0fall;
   vector<Int_t>   ch1rise;
   vector<Int_t>   ch1fall;
   vector<Int_t>   ch2rise;
   vector<Int_t>   ch2fall;
   vector<Int_t>   ch3rise;
   vector<Int_t>   ch3fall;
myTree->SetMakeClass(1);
TBranch  *b_event_EventTime_fSec;
TBranch        *b_event_ch0rise;   //!
TBranch        *b_event_ch0fall;   //!
TBranch        *b_event_ch1rise;   //!
TBranch        *b_event_ch1fall;   //!
TBranch        *b_event_ch2rise;   //!
TBranch        *b_event_ch2fall;   //!
TBranch        *b_event_ch3rise;   //!
TBranch        *b_event_ch3fall;   //!
myTree->SetBranchAddress("EventTime.fSec", &EventTime_fSec, &b_event_EventTime_fSec);
myTree->SetBranchAddress("ch0rise", &ch0rise, &b_event_ch0rise);
myTree->SetBranchAddress("ch0fall", &ch0fall, &b_event_ch0fall);
myTree->SetBranchAddress("ch1rise", &ch1rise, &b_event_ch1rise);
```

```
myTree->SetBranchAddress("ch1fall", &ch1fall, &b_event_ch1fall);
myTree->SetBranchAddress("ch2rise", &ch2rise, &b_event_ch2rise);
myTree->SetBranchAddress("ch2fall", &ch2fall, &b_event_ch2fall);
myTree->SetBranchAddress("ch3rise", &ch3rise, &b_event_ch3rise);
myTree->SetBranchAddress("ch3fall", &ch3fall, &b_event_ch3fall);

// pobierz liczbe elementow zapisanych w TTree
int nentries = myTree->GetEntries();

TString hName, hDescr ;
hName += "hCh";  hName += chan;  hName += "Distrib";
hName += zenith_angle; hName += "deg";
hDescr += "Channel "; hDescr += chan ; hDescr += " TOT distribution ";
hDescr += zenith_angle ; hDescr += " deg";

TH1F *hChDistr = new TH1F(hName.Data(),hDescr.Data(),26,-0.5,100.5);
hChDistr->GetXaxis()->SetTitle("TOT [ns]");
hChDistr->GetYaxis()->SetTitle("Entries");


// vector<Int_t>& rise = ch0rise;
// vector<Int_t>& fall = ch0fall;
//
// cout << "selected chan 0. " << endl ;
//
// if (chan==1) { cout << "selecting chan 1." << endl ;rise = ch1rise; fall =
ch1fall;}
// if (chan==2) { cout << "selecting chan 2." << endl ;rise = ch2rise; fall =
ch2fall;}
// if (chan==3) { cout << "selecting chan 3." << endl ;rise = ch3rise; fall =
ch3fall;}


int i = 0;
for ( i=0; i < (nentries-1) ; i++ ) {
       myTree->GetEntry(i);
       //cout << i << " " ;
       //cout << (int) event->getEventTime().GetSec() << endl ;
       //int timeStamp = event->getEventTime().GetSec();
//     cout << "event: " << (i+1) <<  " vect sizes: "
//             <<  ch0rise.size() << " " << ch0fall.size() << " "
//             <<  ch1rise.size() << " " << ch1fall.size() << " "
//             <<  ch2rise.size() << " " << ch2fall.size() << " "
//             <<  ch3rise.size() << " " << ch3fall.size() << endl ;
       if (chan==0) {
       if (ch0rise.size() && ch0fall.size()) {
              if (ch0rise.size() == ch0fall.size()) {
                     int k=0;
                     int vsum = 0 ;
                     for(k=0; k<ch0rise.size();k++) vsum += (ch0fall[k] -
ch0rise[k]);
                     hChDistr->Fill(vsum*1.25); // precyzja TMC DAQ jest 10ns/8 =
1.25 ns
              }

       }
       }
       if (chan==1) {
       if (ch1rise.size() && ch1fall.size()) {
              if (ch1rise.size() == ch1fall.size()) {
                     int k=0;
                     int vsum = 0 ;
                     for(k=0; k<ch1rise.size();k++) vsum += (ch1fall[k] -
ch1rise[k]);
                     hChDistr->Fill(vsum*1.25);
              }

       }
       }
       if (chan==2) {
       if (ch2rise.size() && ch2fall.size()) {
              if (ch2rise.size() == ch2fall.size()) {
                     int k=0;
                     int vsum = 0 ;
                     for(k=0; k<ch2rise.size();k++) vsum += (ch2fall[k] -
ch2rise[k]);
                     hChDistr->Fill(vsum*1.25);
              }
```

```
        }
        }
        if (chan==3) {
        if (ch3rise.size() && ch3fall.size()) {
                if (ch3rise.size() == ch3fall.size()) {
                        int k=0;
                        int vsum = 0 ;
                        for(k=0; k<ch3rise.size();k++) {
                                vsum += (ch3fall[k] - ch3rise[k]);
                                //cout << "vsum: " << vsum << " fall: " << ch3fall[k] <<
" rise: " << ch3rise[k] << endl ;
                        }
                        hChDistr->Fill(vsum*1.25);

                }

        }
        }

}



/*f->Close();
delete f;*/
return (TH1F*) hChDistr ;

}
```