2. April 2014

PREVENTING TERRORISM USING COSMIC RAY MUONS

PHASM201 - PHYSICS PROJECT MSCI

MICHELE PIERO BLAGO - 13084032
Supervised by Prof Jennifer Thomas and Dr Ryan Nichol
University College London

Abstract

In the past decades, the number of terrorist incidents such as bombings has grown in Western society, making techniques to detect and prevent explosive devices more important than ever before. Therefore, the CREAMTEA research group at University College London investigates cosmic ray muon scattering tomography (MST) using plastic scintillator detectors. These detectors are applicable in real world scenarios, such as in public areas or cargo scanning due to their low costs in comparison to other muon detection techniques.
In order to study the properties of the MST prototype created by the CREAMTEA group, an algorithm is needed to analyse the produced data. This algorithm was developed over the course of this Final Year Physic Project MSci. Its objective is to detect muons crossing the detector and allocate them to a position and time in the individual detector sections (so called modules).
Analysis of detector data files using this algorithm lead to the detection of muons, however, less than expected. Furthermore, the data analysis shows that the produced data needs to be enhanced in order to increase the detection performance and makes suggestions how to accomplish this goal.

Plagiarism Statement

I hereby confirm that this report was entirely written by me except for any quotations or periphrases which were clearly indicated and assigned to their individual authors, including electronic sources as the World Wide Web, under the rules of scientific citation. This also comprises all figures, pictures and tables in this work. I am conscious that an incorrect or untruthful declaration is an attempt of deception and will be treated as plagiarism in agreement with the University regulations for conducting examinations.

X 

Michele Piero Blago

Table of Contents

Introduction

Great Britain and in particular its capital London has been a target of a large number of terrorist incidents in the past decades. Since 2000, 13 out of 16 of those incidents involved explosive devices. The obstacle in preventing these attacks is the vast number of public areas such as train and underground stations as well as the unregulated shipping and trafficking of goods over the UK border.

Developing a technique to minimise this danger is the objective of the CREAMTEA research group. CREAMTEA (*Cosmic Ray Extensive Area Mapping for Terrorism Evasion Application*) is part of the High Energy Group at University College London (UCL). The goal is further investigation of large area scanning using cosmic ray muons. The CREAMTEA group in particular studies plastic scintillator detectors as a low-priced alternative to other muon detectors. They have successfully constructed a Muon Scattering Tomography (MST) prototype and are now evaluating its performance.

For this purpose an algorithm is needed which analyses the data produced by the detector prototype. This algorithm has to detect muons crossing the detector and allocate them in the detector planes. As part of the Physics Project MSci of UCL such an algorithm was developed and tested.

The CREAMTEA group

The Idea

The final goal of the CREAMTEA research group is to provide a method for quick scanning of large areas such as train and underground stations or truck cargos. This would reduce the risk of terrorist incidents directly by detecting explosive devices in public areas or prevent trafficking of special nuclear material and bombs.

The CREAMTEA research group uses cosmic ray muons as a natural source for large area scanning and cargo tomography. The reason why cosmic ray muons are the perfect candidates for this purpose is now briefly explained.

Cosmic Ray Muons

The major part of cosmic radiation consists out of protons which hit the upper atmosphere. Their reaction results in a cascade of secondary particles such as Pions. The Pions decay very quickly and produce more stable Muons. Due to their relativistic velocity their mean lifetime of around 2.197 μs [1] is sufficient to travel the distance to the earth's surface.

The muons belong to the family of leptons with spin ½ and an electric charge of -1e. Remarkably, their high mass of 105.66 MeV/c^2 [1] allows them to penetrate mass to a high extension. They are, however, deflected by material with high density and high proton number. This type of material is used to shield against gamma radiation such as lead or tungsten or special nuclear material itself. In addition, the natural cosmic radiation has two more advantages. First, no artificial source is needed. This decreases the costs, especially for large area screening. Secondly, the muons are part of our day-to-day natural background radiation. This means there is no radiation hazard for the people in screened areas.

The Experimental Setup

In order to make the MST technique applicable for terrorism evasion, the detection has to be economical as well. Hence, the CREAM TEA group studies the characteristics of plastic scintillator detectors which represent a low-priced alternative to other muon detection techniques.

The group succeeded in constructing a MST prototype which is using such plastic scintillators for detection.

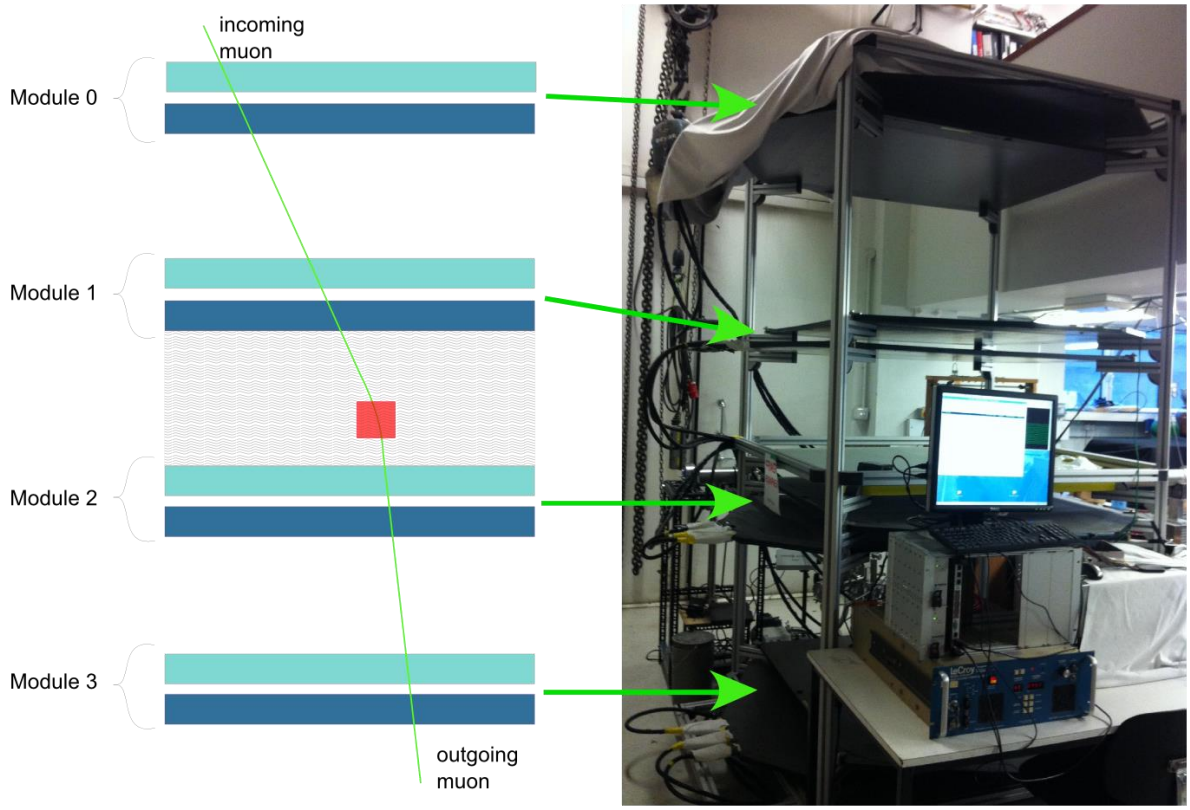


Figure 1 MST Prototype: a) On the left hand side the detector prototype is shown schematically. Each module consists of two detector scintillator planes. The scintillator stripes of the light blue elements are perpendicular to the stripes in the dark blue elements. The grey area is screened using the muon deflection, here a green line. The red box represents a high-Z object which deflects the incoming muon. **b)** The right hand side shows the real experimental setup. The individual scintillator planes can be seen.

The MST prototype consists of a total of four modules. This structure is illustrated in Figure 1. The upper two modules determine the trajectory of the incoming muon. The muon then crosses the volume, which is screened and is deflected if an object of high density or high Z number is present. The resulting trajectory of the muon is then measured by the two lower modules.
The cross section of a scintillator plane is shown in Figure 2. Each module contains two planes of parallel triangular scintillator stripes of 1 m length respectively. The planes have a surface area of 1 m^2.

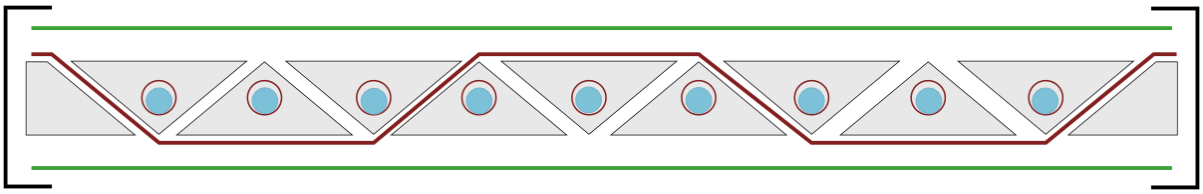


Figure 2 Scintillator Plane Cross Section: Each plane consists of parallel arranged triangular scintillator stripes of 1 m length. The total width of a plane also is 1 m resulting in a surface area of 1 m^2. When a particle crosses a scintillator stripe photons are emitted which are then guided to a photomultiplier by the fibres here shown in pictured as blue circles. The red line portrays the wrapping to provide stability and the green and black frame represent the photon shielding cover.

When a particle crosses one of the planes the luminescent material of the stripes partially absorbs its energy. This energy is then emitted in form of photons and guided by fibres, represented as blue circles in Figure 2, to a photomultiplier tube. There the photons are converted into an electric signal.

The two planes of a module are perpendicularly arranged. Consequently the two coordinates of the muons are measured by the two planes and thereby the position of the crossing muon is determined. The upper two modules determine the linear trajectory of the incoming muon. The muon then traverses the scanned volume where it may be deflected by an object of interest. This deflection is then measured by the lower two detectors which spot the outgoing muon's trajectory.

The Data

The Algorithm's most important property is to detect peaks which belong to particles crossing the modules. It therefore has to be distinguished from the electrical background noise and light leaks in the scintillator planes. Thus the properties of a pulse of the photomultiplier tube (PMT) have to be studied as well as the typical features of the noise.
The collected data is structured as follows. Each data file consists of a certain number of entries which is chosen when taking a measurement. The files usually include 1000 entries in the test runs of the detector. For each of the four modules 64 channels are distinguished representing the individual position on the module. One entry is composed of a 512 ns voltage measurement for all 64 channels in each module.
The data was obtained from the UCL plus1 server [2]

The Background Noise

Figure 3 shows the 64[1] channels of module 0 for an arbitrary entry number. This picture was produced by the runOfflineTargetDisplay.C script which is included in the RootMcp package[2] [3]. Each box displays the course of the voltage in millivolts on the y-axis over the time on the x-axis in 512 ns for every channel respectively. The units are the same for every graph in this paper unless stated otherwise.
In this example one can observe that the code is structured in a distinctive way. Each line of channels shows a repeating structure of voltage deflections. This feature will be used later in the data preparation since it impedes the detection of PMT pulses.

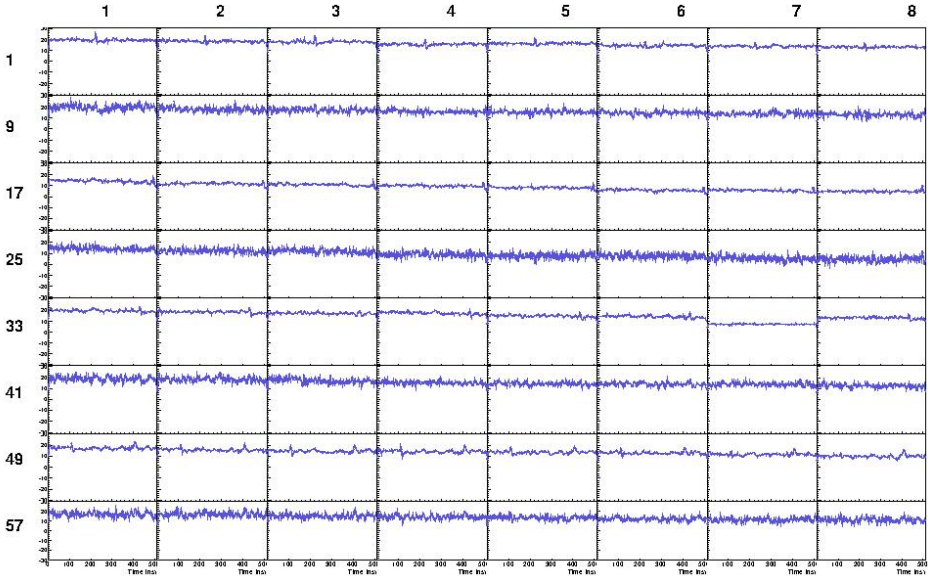


Figure 3 Exemplary 64 Channels of Module 0 for Arbitrary Entry Number: In this picture a typical example of the data which only contains background noise is shown. It features the 64 channels of module 0. Including the same amount of channels for the other three modules this makes up one entry of the data file. In the individual lines a repeating structure can be observed which will be relevant for the reduction of background noise later.

[1] Although the channels are labelled from 1 to 64 they are assigned the number 0 to 63 in the code (see appendix).
[2] For further information on the programming language ROOT and the package RootMCP please consider my *Progress Report PHASM426* [10] or visit *Download RootMCP* [3] by the High-Energy Physics Group of UCL.

A PMT Pulse

Figure 4 shows an example of an ideal PMT pulse. The major feature which can be taken from the picture is that a particle crossing the scintillator planes results in a negative deflection of voltage. The characteristic course of the pulse is a sharp decline of the voltage and a long relaxation time.

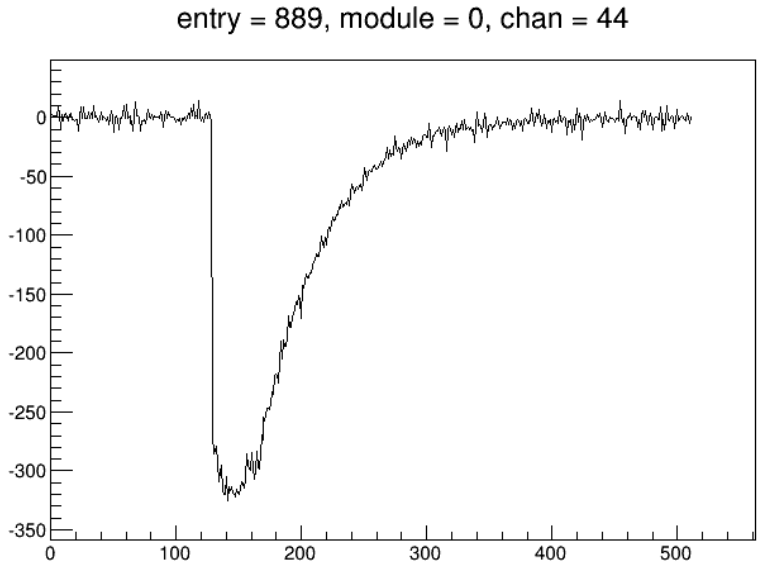


Figure 4 Course of Ideal PMT Pulse: The picture shows an example of an ideal pulse produced by the photomultiplier. The ratio of peak size and noise is immense which makes its detection straightforward. The minimum value of around -330mV is the most common peak size in module 0.

However, this is not applicable for all pulses and makes the fitting of a function impossible as demonstrated in Figure 5. It shows a PMT pulse which will be harder to detect because of its uneven structure.

Moreover, its minimum size is of only around -50mV. For that reason it can get confused with electrical noise which reaches up to -60 or even -70mV.

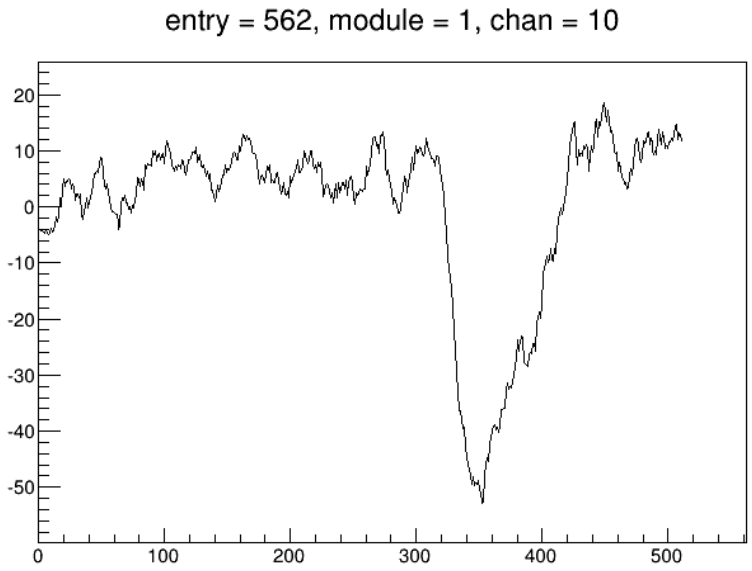


Figure 5 Course of Uncharacteristic PMT Pulse: An example of a PMT pulse of low minimum and uneven structure. This type of peaks has to be taken into account, too, and may be confused with background noise which can reach similar heights.

In conclusion, one can say that the main features of the PMT pulses are their negative deflection and the width of their peaks. In the data are many noise deflections of only a small width that can be distinguished due to this particular feature.

The Algorithm

In order to process the data the code is split into two main parts. The first three functions which will be described in the following prepare the data by reducing the noise and offset and filtering high frequency noise. The second part of the code then analyses the data by collecting the features of each individual peak, deciding if the peak comes from the photomultiplier tube and finally by finding muons which cross all four detector modules.

Data Preparation

As mentioned the data is first processed to facilitate the analysis and peak finding and to reduce the occurrence of false positives after the detection of PMT pulses.

Noise Reduction

The Noise Reduction function uses the repeating structure of the noise which was described in *The Background Noise* paragraph. As an example how the code works the lines marked in green of Figure 6 are taken. The third and fourth lines of channels show a repeating peak with a negative deflection of around -1000mV. The structure of the peak would make it a valid candidate for a PMT pulse. Nevertheless, the fact that the peak repeats in every single channel proves that it is false positive. This kind of signal is very common in the data and would cause a large number of false positives in the PMT pulse finding function later.

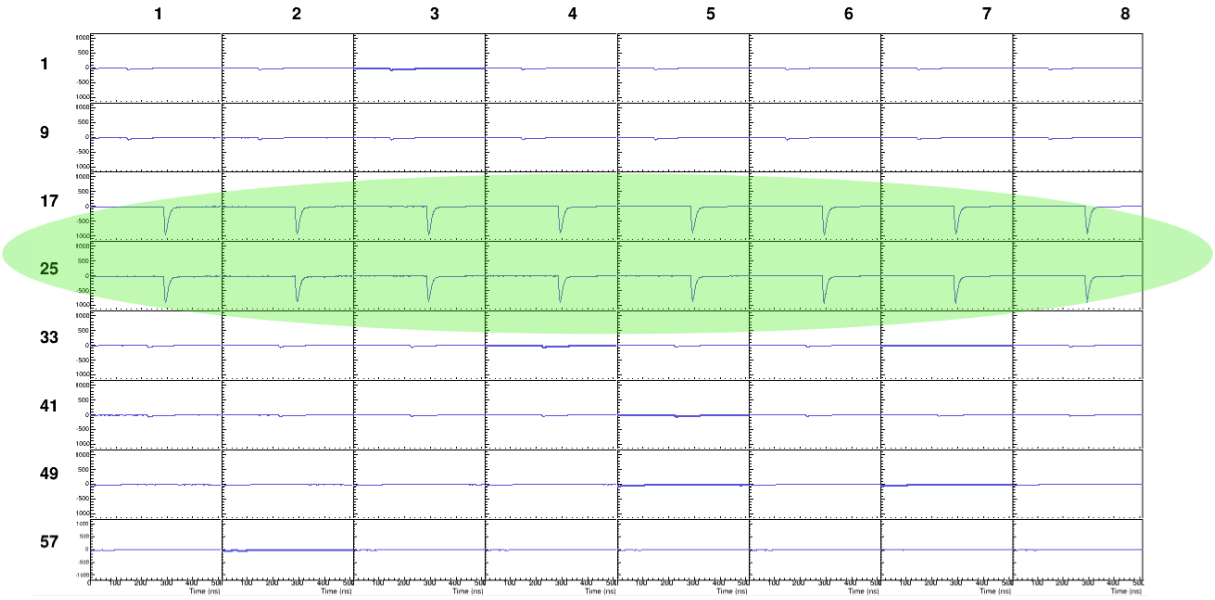


Figure 6 Repetitive Noise and False Positive Peaks: This arbitrary entry displays two lines (marked in green) which contain peaks of around -1000mV each. Every single peak would make a valid PMT pulse; the repeating structure however hints that it is a false signal by the photomultiplier.

The Noise Reduction function sums up all eight graphs of one channel line and constructs an average graph. In the next step the average graph is subtracted from each individual channel of the line. The results for the first two channels are depicted in Figure 7. The deflection of -1000mV was highly reduced. As a result a much smaller peak of 40mV is generated. This peak is no longer a PMT pulse candidate due to its positive deflection which is now well in the range of the background noise.

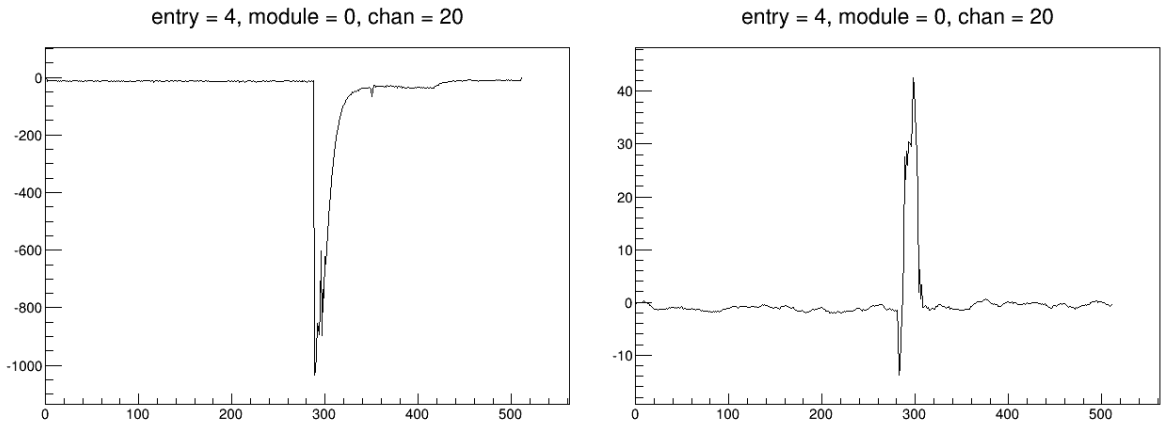


Figure 7 Effect of Noise Reduction on False Positive Peak: a) The picture on the left shows one of the original channels before the Noise Reduction function was applied. The structure makes it a valid candidate for a PMT pulse. **b)** The result of the reduction is portrayed in the picture on the right. One can see that the deflection -1000mV was clearly reduced. The result is a much smaller positive peak of 40mV which is in the range of the typical noise.

This procedure is repeated for every single channel line in the entire data file. It demonstrates an ideal example of the functionality of the Noise Reduction method. In this case the deflection maintained after reduction is comparably high, which is due to the large original deviation of -1000mV. For smaller repeating peaks the resulting maximum is substantially smaller. In conclusion, the overall noise is significantly reduced which illustrates the necessity of the function.

Offset Correction

In contrast to the Noise Reduction method which reduces the noise according to the average of an entire channel line, the Offset Correction function works for individual channels. Its working principle is straightforward: each individual graph is subtracted by its arithmetic mean of the single voltage values.

This has the obvious disadvantage that graphs with big deflections receive a new offset in the opposite direction of the peak after the correction. However, this always occurs in direct relation to the size and width of the peak. For an ordinary PMT pulse the y-axis shift is negligibly low and does not affect its detection. Furthermore, other deflections such as voltage fluctuations which last significantly longer than PMT pulses are shifted towards zero and will therefore be less likely mistaken for a pulse. These can occur in single graphs and are thus not corrected by the Noise Reduction function. Such an example is shown in Figure 8. The long and steady plateau unambiguously indicates a voltage fluctuation which is here marked with a green circle.

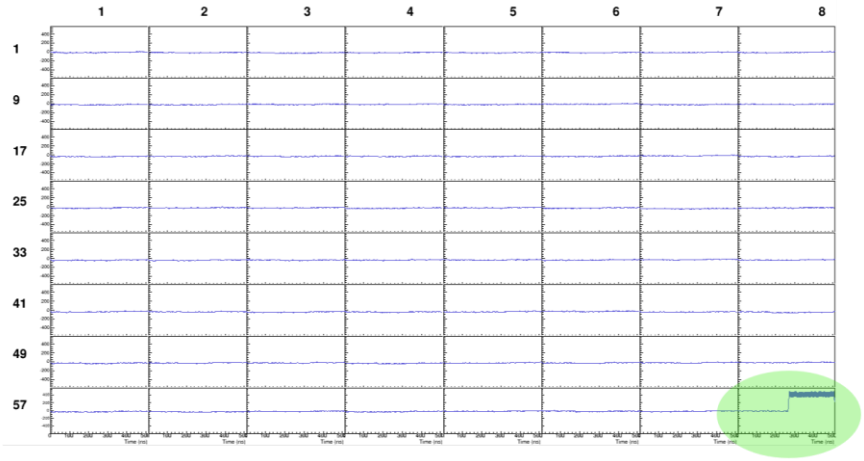


Figure 8 Example of Single Voltage Fluctuation: This is not corrected by the Noise Reduction function which only handles repeating voltage fluctuations. The Offset Correction function however covers this type of irregularities.

In direct comparison the reduction of a fluctuation is demonstrated in Figure 9. Here the offset of -20mV is rather small but illustrates the working principle very well. The result of the correction is shown on the right hand side of the figure.

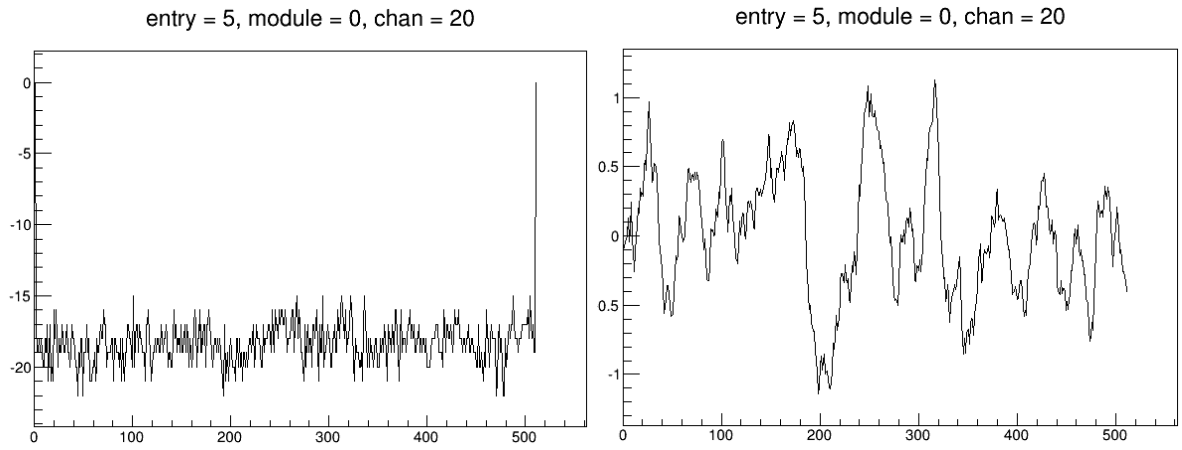


Figure 9 Effects of Offset Correction on Data Graph: A direct comparison of before and after offset correction. **a)** The graph on the left hand side has an offset of -20mV which is reduced to zero by the Offset Correction function. **b)** The result of the correction is presented on the right hand side; the offset is reduced to zero.

The small offset of -20mV is completely reduced to zero in the example of Figure 9. Clearly the correction also works similarly for bigger offsets.

Low-Pass Filter

The graphs of module 1, 2 and 3 especially imply a high frequency voltage fluctuation. These very small peaks with duration of only a few nanoseconds can reach values of around 80mV and are therefore in the range of PMT pulses. Such an example is shown in Figure 10.

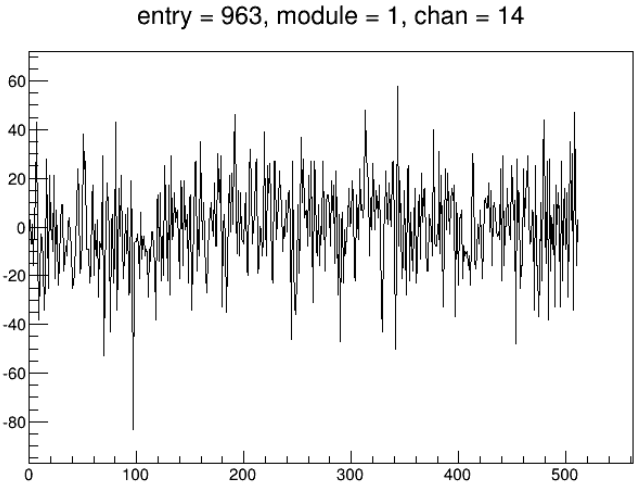


Figure 10 High Frequency Noise: The picture shows the high frequency noise with very thin peaks which in this case reach 0--80mV and are thus in the range of PMT pulses.

Moreover, the exact position of the minimum will become crucial later in assigning a time to PMT events. Hence, a flattening of the peak due to high frequency fluctuations needs to be adjusted. For this purpose the Low-Pass Filter function is used which reduces a peak if its half width is below a defined threshold. In this case a threshold of 6ns has proven to filter most of the noise fluctuations. An example is shown in Figure 11.

The peak is taken from module 3 which contains considerably more noise than module 0. The ratio of peak size to noise in Figure 11.a is very low and the voltage fluctuates at high frequency. This diminishes the true value of the minimum position and the full width at half minimum (FWHM). The FWHM will be a key criterion in detecting PMT pulses as will be discussed later.

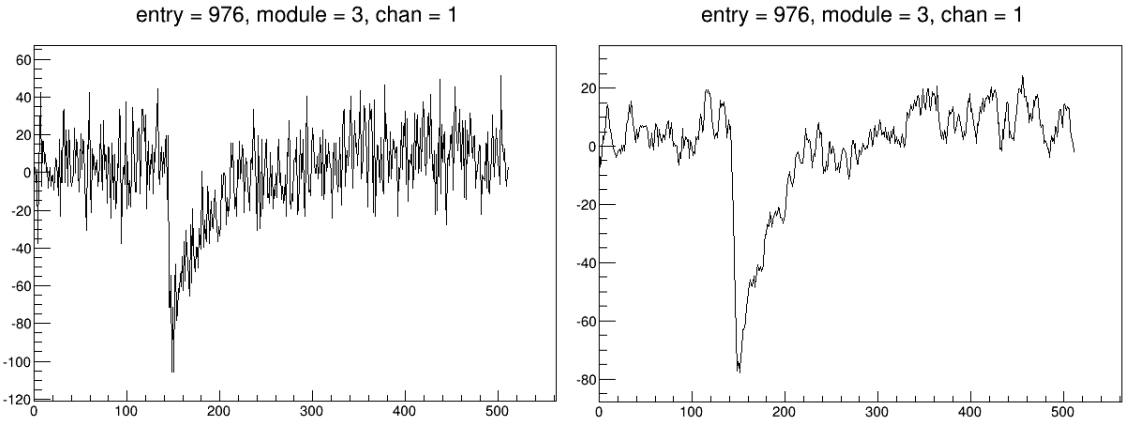


Figure 11 Effects of the Low-Pass Filter: a) The graph on the left is not filtered. This affects the positioning of the minimum as well as the full width at half minimum which is one of the key features to determine whether a peak is a PMT pulse or not. **b)** On the right hand side the same graph is shown after filtering. The position of the minimum as well as the FWHM can now be determined with much higher precision.

After the low-pass filtering the precision of the minimum location and the FWHM is no longer biased by the noise fluctuations as is observable in Figure 11.b.

Results of Data Preparation

The changes due to the data preparation process are emphasised in Figures 12 and 13. Each figure presents a histogram for one of the four modules respectively. The histograms are filled with the absolute minimum value of each single graph of the whole data file. The interval shown is from 0mV to 150mV with a bin size of 1. Consequently, the PMT pulse minima in this range are invisible among the immense amount of noise peaks. Therefore the histograms only demonstrate the peak size of the background noise for the individual modules.

Figure 12 illustrates the noise values before data preparation. One can see that the peaks of the background noise are for all modules in the range of the first PMT pulses. The extensive number of these occurrences would result in many false positives in the PMT pulse detection.

In comparison the histograms in Figure 13 show the exact same setting but after the data preparation. The absolute minimum values of the noise now do not exceed 30mV. Hence they do not influence the detection of PMT pulses and the probability of detecting a false positive is highly reduced.

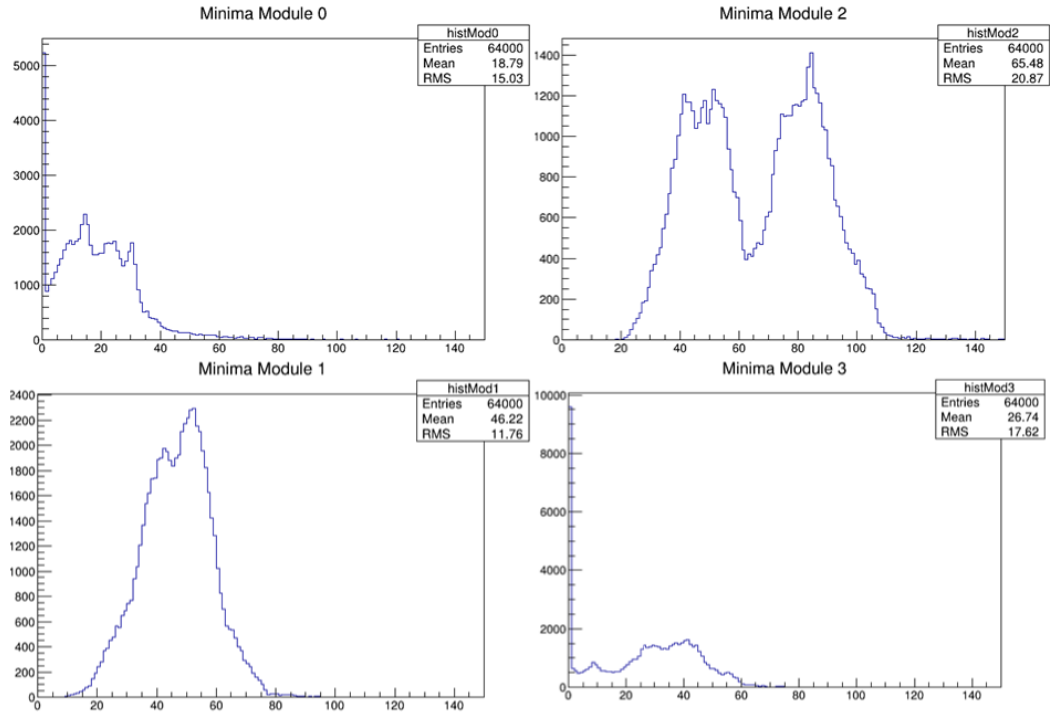


Figure 12 High Background Noise before Data Preparation: The histograms show the number of occurrences on the y-axis for the absolute minimum values in range of 0mV to 150mV of all channels and for each individual module of run number 365. This is before the data preparation and shows the range of the negative deflections of the background noise. **a)** The noise peaks in module 0 in the top left corner is majorly distributed around 20mV. There is a big amount of peaks up to 70mV and thus in the range of PMT pulses. **b)** Module 1 in the top right corner has its noise peaks exactly in the range of the first PMT pulses and up to 110mV. **c)** Module 2 in the bottom left corner contains negative noise peaks mainly around 45mV but also up to 80mV **d)** In module 3 in the bottom right corner again noise peaks are up to 60mV and covering the PMT pulse peaks.

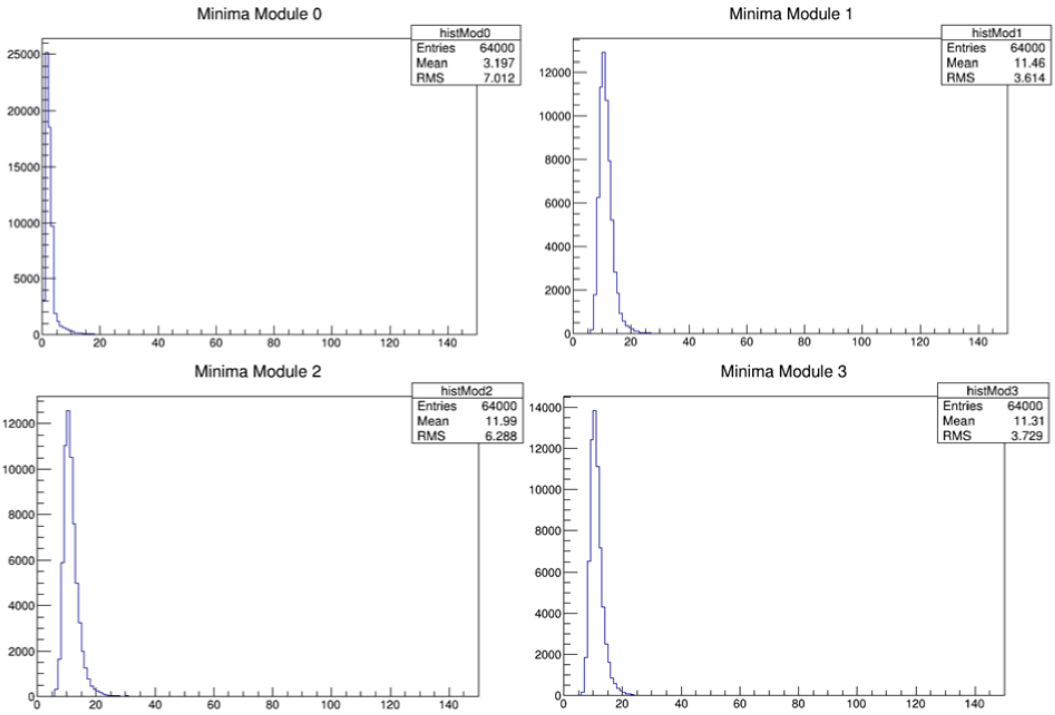


Figure 13 Low Background Noise after Data Preparation: The four histograms show the distribution of the absolute minima for every channel in each module after the data preparation for the same run 365 as seen in Figure 12. In comparison to Figure 12 the peaks do not exceed 30mV in all modules and are therefore not in the range of the PMT pulses.

Data Analysis

After preparing the data the risk of detecting false positives in PMT pulses is highly decreased. Moreover, the high frequency noise has cleared the shape and structure of the pulses and the offset was corrected. Therefore, the characteristic features of the peaks can now be determined.

Peak Analysis

The Peak Analysis function determines the following values for each single graph: minimum, full width at half minimum and surface area of peak. To store these major criteria of each peak, a *PmtPulse* class was created. In addition, this class is able to hold the module-, channel-, and event number as well as the event time. The event time is calculated by the minimum position on the x-axis which is given in nanoseconds plus the event number times 512ns, the duration of one event.

As already mentioned, the deflection caused by the photomultiplier is always negative. Therefore only the minimum values need to be identified. The height of the minimum is dependent on the voltage settings of the photomultiplier. During the course of the experiment different settings were tested.

To verify whether the peaks are a result of the photomultiplier and not caused by electrical noise a zero measurement was taken. In this measurement the voltage adjustment, called *High Voltage* (HV), was set to zero Volt. Here, no deflections above the electrical noise of about 40 mV were detected (after the data preparation). Hence, it was assured that all pulses are due to the photomultiplier and that an increase in HV does result in higher absolute minimum values. However, the output differs depending on the module which produces it. For instance, the pulses of the uppermost module 0 were on average much higher than the pulses of other modules. Moreover, the minimum height varies with the number of photons emitted by the scintillators. Consequently, no reasonable peak interval exists to identify particles or especially muons crossing the planes.

The FWHM is essential in deciding whether the peak is a PMT pulse or not. Noise peaks as seen in Figure 10 provide an extremely short width, whereas voltage fluctuations as shown in Figure 9.a have a disproportionally large FWHM if not corrected by the data preparation part of the algorithm. These peak values are then stored in a *PmtPulse* object and returned to the main function.

PMT Check

The *PMT Check* function then takes each single of the *pmtPulse* objects as an argument and decides whether it is a PMT pulse or not. The first criterion to be analysed is the size of the minimum. As described in the Peak Analysis paragraph the peak size of PMT pulses varies between the different modules as well as within the individual modules. Since a muon can result in the emission of multiple photons depending on its energy, an upper limit for the absolute minimum value is not advisable. Therefore, a threshold has to be chosen for each module rather than an allowed interval.

On the other hand, a single photon which is emitted by the scintillators needs to be considered as a particle crossing the module, too. Consequently, the lowest allowed minimum value needs to be determined. This specific value is dependent on the module and its individual settings.

For the full width at half minimum value a large interval was chosen to consider pulses of different shapes as demonstrated in Figure 5. With that said, the FWHM did not exceed a value of 150mV.

The distribution of the absolute values for the minima of detected PMT pulses can be seen in Figure 14. Evidently, the average peak heights as well as the total amount of pulses decrease over the modules.

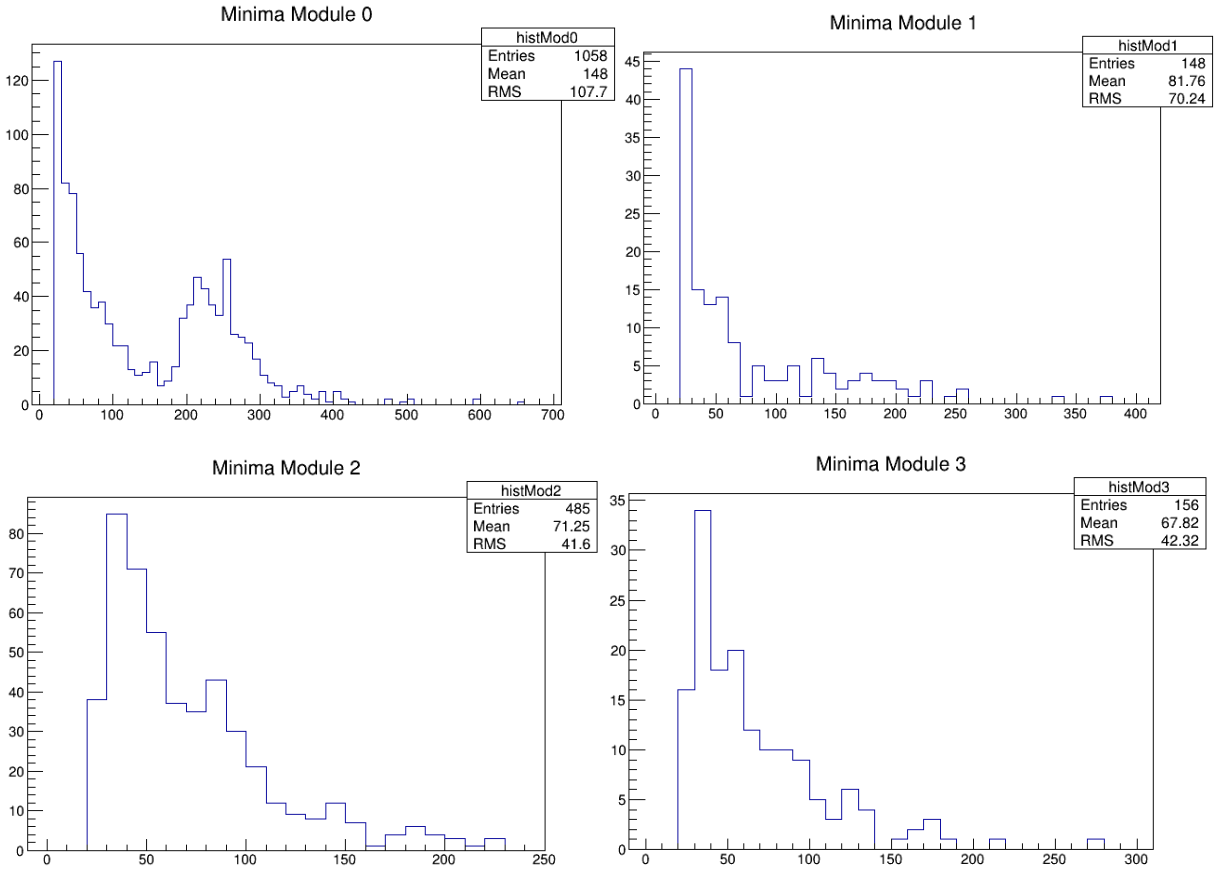


Figure 14 Size of PMT Pulse Minima: The histograms show for each individual module the absolute size of the minima for the detected PMT pulses of run number 365. The x-axis goes up to the value of the highest detected minima for each module. **a)** Module 0 contains the most pulses in addition to the highest values around 350mV. **b)** A lot less pulses are detected in module 1, especially for voltage values above 80mV. The highest values are around 250mV with two pulses around 360mV. **c)** For module 2 again are more peaks found then in module 1 with a similar range of voltages. **d)** The last module 3 images the least amount of pulses. The distribution is similar as in modules 1 and 2.

The largest source of false positives is the interval of minimum values where the minima of PMT pulses are in the range of the noise. Hence, this overlap needs to be held as small as possible. Figure 13 illustrates the range of the noise for the test run 365 which indicates a threshold of 25mV for the lowest allowed minima.
By visual inspection of all PMT pulse graphs 5 to 10% false positives were detected, dependent on the run number. This is a comparatively high amount which is, however, justified by the next step of the algorithm, the *Muon Check.* Provided a peak passes the criteria given by the *PMT Check* it is apprehended to the end of a list of pulses, namely the *pulseList.*

Muon Check

The *Muon Check* function detects particles crossing all four modules. Those particles are then most likely to be muons since all other particles should have been absorbed by the modules themselves or by the walls of the building until it reaches the basement where the detector is located.
The functions operation principle is based on the event time of the individual PMT pulses and checks the occurrence of a PMT pulse.
A muon traveling at a velocity of 0.92±0.01c [4] takes circa 3.63 ns to travel the distance between the different modules. However, a much larger error in the positioning of the muon event is assumed. Ideally the steep fall in the beginning of the peak shape would be used, as seen in Figure 4. The resulting error would be near 4ns. Taking PMT pulses of the shape of Figure 5 into account, one can only locate the particle crossing event by taking the position of the minimum. The different shapes of the peaks and fluctuations result in an additional error of around 10 ns.

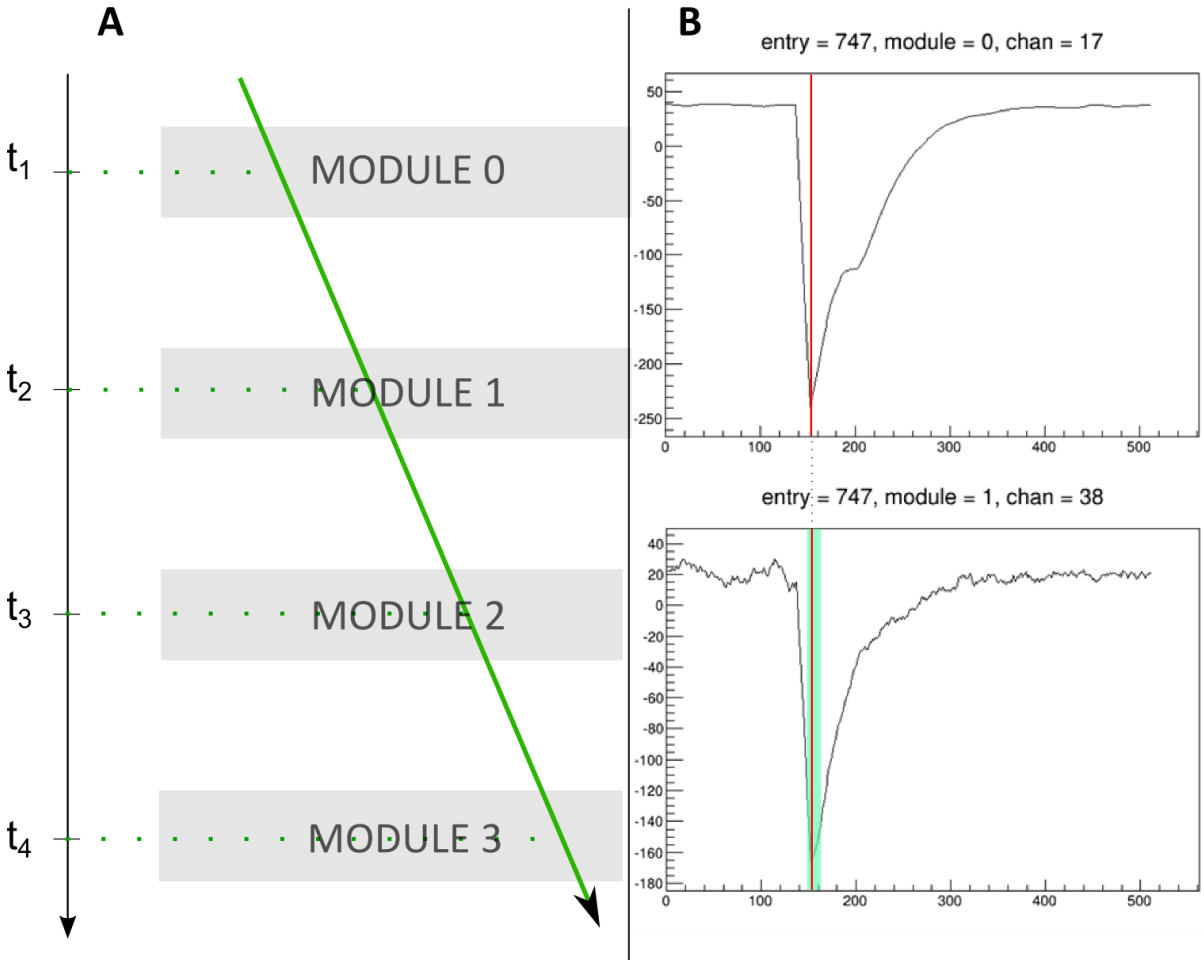


Figure 15 Working Principle of *Muon Check* function: a) The cartoon on the left hand side illustrates the working principle of the Muon Check function. For each PMT pulse in module 0 is checked if a pulse occurred in module 1 in the allowed time interval $t_2 - t_1$. This interval is drawn as a blue box in the bottom graph on the right hand side of the picture. It is then checked for module 2 and last for module 3. If a match occurs in all four modules a muon is detected. **b)** The two graphs on the right hand side show a match of module 0 and 1 of run number 361. The red line indicates the time t_1 of the first PMT pulse and the blue box the allowed interval for the next one. A similar structure in the shape of the two peaks can be observed.

Figure 15.a illustrates the working principle of the Muon Check function. For every PMT pulse time, t1 in the uppermost module 0 is checked if a pulse occurred in next module 1 in the allowed time interval

$$\Delta t = t_2 - t_1 = t_2 \pm (10ns)_{sys} + (3.5ns)_{travel} ,$$

which implies the time the muon takes to travel from one module to the next as well as the systematic error in the timing of the PMT event. In the code the travel time is rounded to 4 ns.
If a match is found, the step is repeated for the next module 2 and in case of another match then for module 3. If a match in all four modules occurs a muon is found. Its properties which include the entry, module and channel number as well as the graph and event time for each module are then stored in a *Muon* object which is appended to the *muonList*.
This proceeding highly reduces the risk of false positives in the *muonList*. As an example, a short calculation is given for the probability to detect a match for randomly distributed false pulses in 15ns intervals in a total of 1000 entries of 512ns. For a worst case of 10% falsely detected PMT pulses out of a total of 2500 pulses per module the probability for a match in two modules is

$$P_{mod0,1} = \frac{2500*10\%}{1000*\frac{512}{15}} = 0.73\% \quad . \tag{1}$$

Therefore, the probability of receiving a match in all four modules for randomly distributed false positive pulses is $\left(P_{mod0,1}\right)^3 = 4 * 10^{-5}\%$. Reasons for false matches are voltage fluctuations which especially appear in the first few entries of each data file. As a result I suggest starting the data analysis at entry number 5 or higher to minimise influences caused by the initial fluctuations.

Results

When the code is executed in its original form no match for all four modules is found. However, various matches for the first two modules are detected. One of these is shown in Figure15.b. Another example is displayed in Figure 16 below.

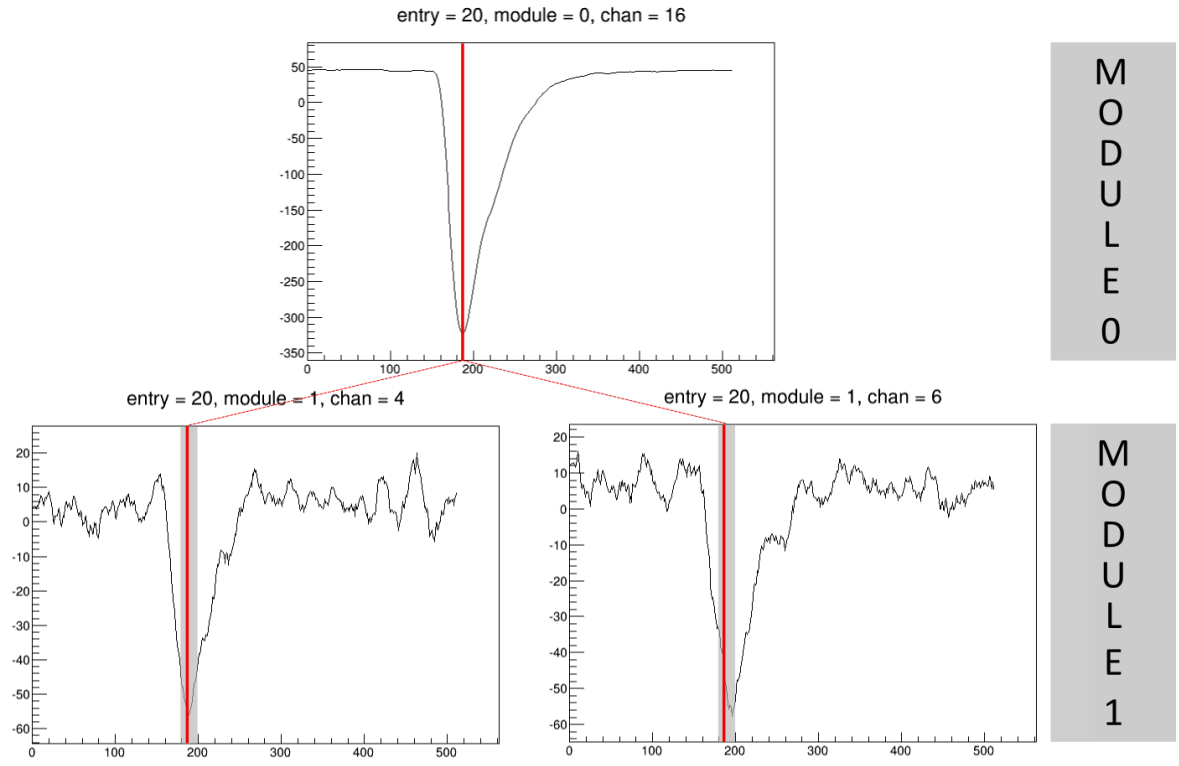
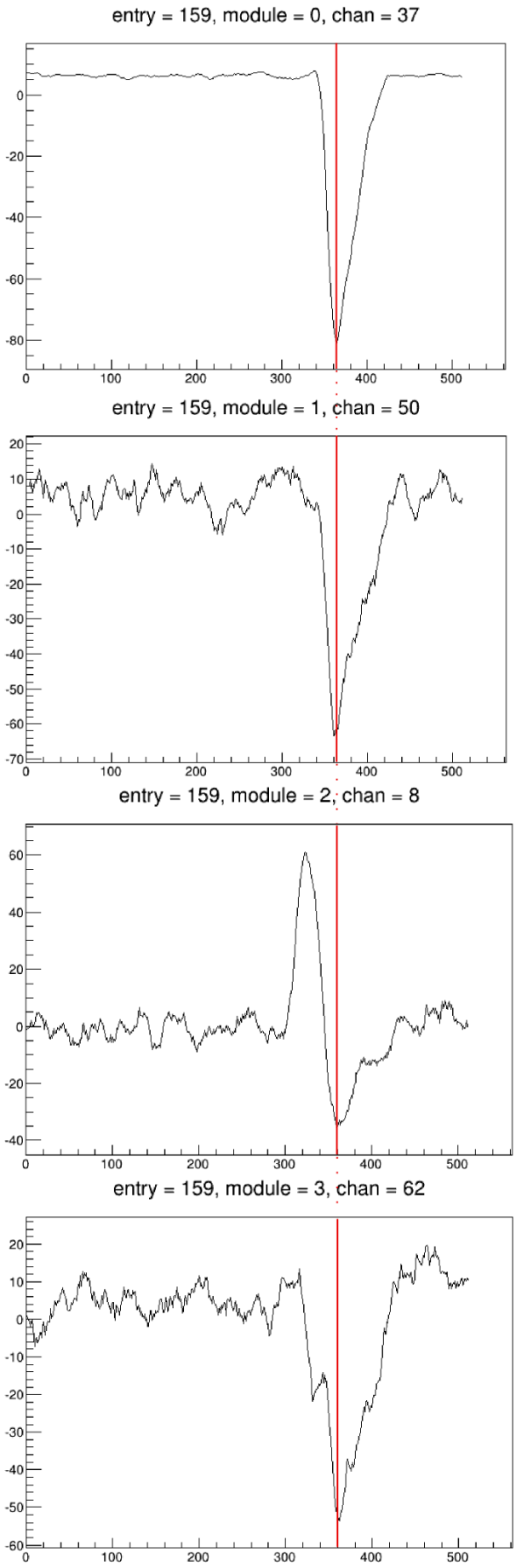


Figure 16 Exemplary Double Hit: Example of a match of PMT pulses in the first two modules of run number 361. The particle causing the peaks crossed two scintillator stripes in module 1 causing a double hit. The red stripes indicate the event time of the particle crossing module 0 and the blue grey boxes illustrate the allowed interval for the minimum position in module 1. One can observe that the peak in module in the uppermost graph has a much bigger minimum value than the peaks in module 1 underneath. Moreover the shape of the peaks is very similar, which was observed before in Figure 15.b.

Since no match in the first three modules could be found in any of the tested data files an *Inverse Muon Check* method was added to the script. This method works identically to the *Muon Check* function; however, checking gradually the modules beginning from the lowermost module 3 and then to the next module in upwardly. In addition, the HV has been increased for the module 1, 2 and 3 which provided a higher background noise and smaller peak sizes caused by the photomultiplier. As a result matches of two modules were found again, in this case in the two lowest modules 2 and 3.



Another change in the code structure and the criteria in the *PMT Check* function finally lead to two matches in the *Muon Check* function which is displayed in Figure 17.

The red line indicates the position of the minimum in the uppermost module 0. Consequently, the short time difference well below the estimated interval can be seen.

As one can see the peak voltage is still considerably low. However, further increase is not possible since the photomultiplier tubes are likely to be damaged when working constantly above 1000V.

The Tables 1 and 2 on the next page display the counted PMT pulses and PMT pulse matches for the individual modules. The run number 361 has a standard HV setting of 975V for each module whereas the other run 363 to 365 have increased HV settings for modules 1, 2 and 3.

One can see a tendency of higher PMT pulse counts in these modules for increased HV values in Table 1. In contrast, the numbers of PMT pulse matches in Table 2 do not show this tendency.

One can draw the conclusion that a higher HV setting does increase the chance of detecting PMT pulses; however, the increase was not high enough to affect the number of identified matches. A higher HV setting could not be tested since it would most likely damage the photomultiplier tubes.

Figure 17 Muon Match: The figure shows a match in all four modules. The red line illustrates the time in the uppermost module 0. This matches easily the allowed intervals in the other modules. Therefore this indicates a muon event.

Run Number	HV [V]	PMT pulses mod 0	PMT pulses mod 1	PMT pulses mod 2	PMT pulses mod 3	PMT pulses total
361	975/975	1037	125	95	53	1310
363	950/1000	1091	182	248	107	1628
364	950/1025	1051	214	526	178	1969
365	975/1050	1036	144	463	149	1792

Table 1 PMT Pulse Counts: In this table the number of PMT pulses is pointed out for the last three runs with increased HV of the photomultiplier tube in contrast to the standard HV setting in run 361. Here, the first number for the HV entry names the setting for module 0, the second one the HV for the other modules 1, 2 and 3. Particularly the disproportional amount of pulses in module 0 as well as a higher number of pulses in module 2 can be observed.

Run Number	HV [V]	Matches mod 0,1	Matches mod 1,2	Matches mod 2,3	Matches 3 modules	Muons detected
361	975/975	21	1	4	1	1
363	950/1000	26	26	9	1	2
364	950/1025	8	25	8	2	—
365	975/1050	10	2	8	—	—

Table 2 PMT Match Counts: The table shows the number of matches in the different modules for the last test runs of the detector with increased HV in contrast to the standard HV setting in run 361. The next columns display the number of matches for PMT pulses in two, three or four modules. All runs had a threshold of 1830 and a total number of 1000 entries of 512 ns each.

Conclusion

The conclusion which can be drawn from the data analysis is twofold. Firstly, one can say that the individual modules indeed measure the crossing of particles. This can be seen from the characteristic shape of the pulses, the occurrence of pulses in two or more modules at nearly the same time and finally the zero HV test run which was mentioned in the last paragraph and did not contain any pulses. That said, the quality of the pulses in terms of shape and low background noise of module 0 did not hold for the other three modules. Higher noise voltage values and lower peaks resulted in a decrease in measured PMT pulse events. Hence, the collected data does not contain as many muon events as expected.

To evaluate the number of expected muon events for a data file of 1000 entries of 512ns each is briefly estimated. For this approximate calculation, a flux of 10 000 muons per square metre and minute is assumed [5]. This is valid for muons of an average energy of 1GeV. As a result, around 85 muons are expected for each square metre in the measured time span at sea level. Two factors now drastically decrease this estimation. First of all, the location of the detector in the basement of UCL Physics Department of more than 5 stories does not absorb all the heavy particles but certainly causes a noticeable decrease of particle numbers, here called reduction factor α. The second factor β is due to the fact that a muon has to cross all four modules to be detected. Taking their spacing of 1m for each module into consideration, the muon has to cross a cuboid of 1x1x4m. This again

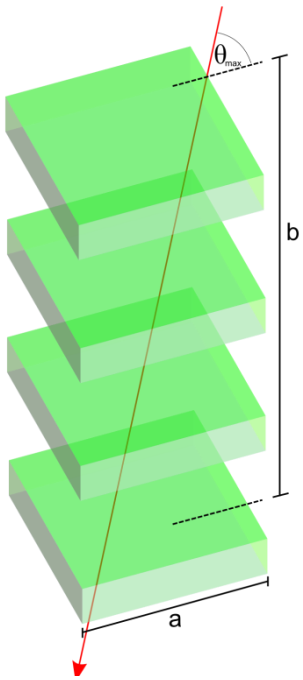


Figure 18 Illustration of Detector Properties: The picture is illustrating the four modules of the detector, the maximum inclination angle θ_{max}, the length a and height b of the detector.

reduces the number by a reduction factor β to the muons of sufficient steep inclination angles. This is demonstrated by a short calculation.
Assuming the inclination angle θ is normally distributed (Equation 2).

$$f(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2)$$

Then the Gaussian function for μ=0 as the expected vertical incoming muon describes the distribution sufficiently well with a value of $\sigma = \frac{\pi}{5}$ (Equation 3).

$$f(x) = \frac{1}{\sqrt{2\pi(\frac{\pi}{5})^2}}\, e^{-\frac{x^2}{2\left(\frac{\pi}{5}\right)^2}} \quad (3)$$

Integration over the interval $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$ results in approximately 1. Hence, integrating over the interval of allowed inclination angles $[-\theta_{max}, \theta_{max}]$ will give an adequate estimate for the factor how the number of incoming muons is reduced. This two dimensional calculation of course does not consider the three dimensional structure of the detector. Nonetheless, it is sufficient for the purpose of estimation.
The angle θ_{max} can simply be determined using the length a and height b of the detector (Figure 18) by calculating

$$\theta_{max} = \arctan(\frac{b}{a}) \quad (4)$$

This gives a result of θ_{max} = 0,245 rad. The reduction factor α is then calculated by

$$\beta = \int_{-\theta_{max}}^{\theta_{max}} f(x)dx \quad (5)$$

Resulting in α = 0.30, hence the number of muons which actually cross the plane is reduced by around 30%. The reduction factor α of the building is difficult to determine but for this purpose estimated around 50%[3]. Hence, the number of muons which should be crossing all 4 modules in the given time is around 13. Consequently, there are a significantly fewer muons detected than expected.

Nevertheless, the example of the analysed data demonstrates the working principle of the algorithm and shows its functionality. The results and most likely the number of detected muons could be highly increased by enhanced data. As mentioned, the aspired level is module 0. Since all modules are equally constructed the only difference is in the triggering and voltage settings of the other modules. Optimising these will be the last step to a working Muon Scattering Tomography detector. To accomplish that a number of test runs of short entry number needs to be carried out. The effect of the small setting changes between each test run can be observed and analysed using the *runTargetDisplay.C* script of in the RootMCP package.
The MST prototype can then be used as a model for large scale applications such as scanning of truck cargo or underground stations and thus reducing the risk of terrorist incidents.

[3] This estimation is based on former student's projects [8] [9].

List of Figures

Figure 2: Adapted from: Minerva Document Database, http://minerva-docdb.fnal.gov/; http://minerva-docdb.fnal.gov/cgi-bin/RetrieveFile?docid=218;filename=cdr-detector-MODIFIED.pdf;version=4, 17.10.2013 Fig. 56

Figure 3, 6, and 8: Created with *runOfflineTargetDisplay.C* from the RootMCP package [3]

Figure 12, 13, and 14: Created using the *Draw ()* member function of Root

All graphs were created using the *drawGraph(TGraph* channel, Int_t i, Int_t module, Int_t chan)* function in the code (see Appendix). All other figures were created using *Inkscape (version 0.48.4)* freeware [6].

Literature and Sources

[1] J. B. e. a. (. D. Group), “pdg.lbl.gov,” 2012. [Online]. Available: http://pdg.lbl.gov/2012/tables/rpp2012-sum-leptons.pdf. [Accessed 10 03 2014].

[2] H. E. G. UCL, “Plus1 server,” [Online]. Available: plus1.hep.ucl.ac.uk.

[3] “Download RootMcp,” [Online]. Available: http://cvs.hep.ucl.ac.uk/trac/creamtea/wiki/RootMcp. [Accessed 21 10 2013].

[4] N. Romero and V. Mukund, “Speed and Decay of Cosmic Ray Muons,” Massachusetts Insitute of Technology, 1998.

[5] R. Clay and B. Dawson, “Pierre Auger Observatory,” [Online]. Available: http://www.auger.org/cosmic_rays/faq.html#how_many. [Accessed 14 03 2014].

[6] “Inkscape,” [Online]. Available: http://www.inkscape.org/en/. [Accessed 24 01 2013].

[7] “Download ROOT,” [Online]. Available: http://root.cern.ch/drupal/content/downloading-root. [Accessed 2013 09 27].

[8] Y. Meiron and C. Tradonsky, “Measurement of the Flux of Cosmic Ray Muons,” 2009.

[9] S. Kliewer, “docstoc.com,” 08 11 2010. [Online]. Available: http://www.docstoc.com/docs/138383019/Muon-Count-Rate-Estimator. [Accessed 2014 03 16].

[10] M. Blago, “Preventing Terrorism Using Cosmic Ray Muons - Progress Report,” University College London, 2014.

Appendix

The appendix includes the algorithm script which was discussed in this report. Consisting of two parts, the first one, *runMuonAnalysis.C,* will run the analysis code when executed. The second one, *muonAnalysis.C,* contains the analysis code of the data file. The file path of the data file has to be stated in *runMuonAnalysis.C.* Before running the code ROOT [7] has to be installed on the computer.

runMuonAnalysis.C

```
void runMuonAnalysis(){
 gSystem->AddIncludePath("-I/home/creamtea/rootMcp/branches/multiusb/");

 gSystem->Load("libGraf.so");
 gSystem->Load("libPhysics.so");
 gSystem->Load("../libMcpTargetRoot.so");

 bool success = gSystem->CompileMacro("muonAnalysis.C","k");
 printf("Success = %d\n", success);
 muonAnalysis("~/CREAM_TEA/data/outputFile363.root"); //path of analysed data file as argument
}
```

muonAnalysis.C

```
#include <iostream>
#include <fstream>
#include <map>
#include <list>
#include "../Defs.h"
#include "../McpTarget.h"
#include "../MultiRawTargetModules.h"
#include "../MultiTargetModules.h"
#include "TH1.h"
#include "TFile.h"
#include "TTree.h"
#include "TCanvas.h"
#include "TMath.h"
#include "TMultiGraph.h"


//holding values for a pmt pulse
class PmtPulse
{
public:
 Double_t minAbs; //absolut value of pulse minimum
 Double_t fwhm; //full width at half minimum
 Double_t area; //surface area of peak
 Double_t minPos; //position of minimum in ns

 Int_t entry;
 Int_t module;
 Int_t channel;

 Double_t eventTime; //time of event in ns
 TGraph *pulseGraph;
};
```

```
//holding values for a detected muon
class Muon
{
public:
 Int_t entry;

 //module0
 Double_t timeMod0; //time of muon event in module, (entry number * 512ns + minimum position)
 Int_t chanMod0;
 TGraph *pulseMod0;

 //module1
 Double_t timeMod1; //all timeMod variables similar calculated as timeMod0
 Int_t chanMod1;
 TGraph *pulseMod1;

 //module2
 Double_t timeMod2;
 Int_t chanMod2;
 TGraph *pulseMod2;

 //module3
 Double_t timeMod3;
 Int_t chanMod3;
 TGraph *pulseMod3;
};


TGraph *getBoxCar(TGraph *grWave, Int_t halfWidth);
TObjArray *reduceNoise(TObjArray* channelLine);
TObjArray *correctOffset(TObjArray* channelLine);
PmtPulse *peakAnalysis(TGraph *channel);
void drawGraph(TGraph* channel, Int_t i, Int_t module, Int_t chan);
Bool_t pmtCheck(PmtPulse *peak);
list<Muon*> muonAnalysis(list<PmtPulse*> pulseList);
void displayMuons(list<Muon*> muonList);
class PmtPulse;
class Muon;


void muonAnalysis(char *fileName) {
 McpTarget *myTarget = new McpTarget(1);
 //myTarget->loadPedestal();
 MultiRawTargetModules *multiRawTargetDataPtr=0;
 MultiTargetModules *multiTargetDataPtr=0;
 TFile *offlineFile = new TFile(fileName);
 TTree *mcpTree = (TTree*) offlineFile->Get("mcpTree");
 if(!mcpTree) { //checks if a tree exists
  std::cerr << "No input tree -- giving up\n";
  exit(0);
 }
```

```
 mcpTree->SetBranchAddress("target",&multiRawTargetDataPtr);
 Int_t numEntries=mcpTree->GetEntries();
 std::cout << "Looping over " << numEntries << " entries.\n";

 list<PmtPulse*> pulseList;
 /*
 //Histograms to analyse minimum distribution in individual modules
 TH1D *histMod0 = new TH1D("histMod0", "Minimum Module 0",102,0,512);
 TH1D *histMod1 = new TH1D("histMod1", "Minimum Module 1",102,0,512);
 TH1D *histMod2 = new TH1D("histMod2", "Minimum Module 2",102,0,512);
 TH1D *histMod3 = new TH1D("histMod3", "Minimum Module 3",102,0,512);
 */
 //initialising variables to count PMT pulses in individual modules
 int pmtPulseCounter = 0; int pmtPulseCounter0 = 0; int pmtPulseCounter1 = 0; int
pmtPulseCounter2 = 0; int pmtPulseCounter3 = 0;


 for(int i=5;i<numEntries;i++) { //start from 5th entry because first entries contain voltage
fluctuations
   mcpTree->GetEntry(i);
   if(i%100==0) std::cerr << "*";
   if(multiTargetDataPtr) delete multiTargetDataPtr;
   multiTargetDataPtr=new MultiTargetModules(multiRawTargetDataPtr);
   myTarget->fillVoltageArray(multiTargetDataPtr);
   const int n = multiTargetDataPtr->getChannel(0,0)->GetN(); //Number of points in TGraph
   if (n!=512) std::cerr << "number of points in graph is not 512!\n";
   for(int module=0;module<4;module++) {
    for(int line=0;line<8;line++) { //looping over individual channel lines to reduce repeating noise in
channels


      TObjArray *channelLine = new TObjArray(); //creating collection of channels of this line
      for(int chaninline=0;chaninline<8;chaninline++) {
       TGraph *chanTemp = multiTargetDataPtr->getChannel(module,chaninline + line*8);
           channelLine->Add(chanTemp); //adding channel to channel line collection
      }

      //Data preparation for whole channel line
          TObjArray *offsetCorrLine = (TObjArray*)correctOffset(channelLine); //offset correction of
channelLine
          TObjArray *reducedLine = (TObjArray*)reduceNoise(offsetCorrLine); //reduction of
channelLine

      TGraph *channel;
      TGraph *channelUnfilt;
      PmtPulse *peak;

      //now handling individual channels of each line
          for(int chanNo=0;chanNo<8;chanNo++) {
           Int_t chan = (line*8 + chanNo);
           channelUnfilt = (TGraph*)reducedLine->At(chanNo);
     channel = getBoxCar(channelUnfilt,7); //Filtering high frequency noise
```

```
        peak = peakAnalysis(channel); //collecting properties of graph

        //Double_t minAbsVal = peak->minAbs; //only needed for histogram analysis
        peak->module = module; //is needed because pmtCheck distinguishes between the modules
        Bool_t peakIsPulse = pmtCheck(peak); //checking if pulse fulfill criteria to be a PMT pulse
candidate

        //further processing if graph contains PMT pulse candidate
        if (peakIsPulse) {
          pmtPulseCounter++;
          peak->entry = i;
          peak->channel = chan;
          peak->eventTime = i*512 + peak->minPos; //entry * 512ns of each entry + position of
minimum in ns
          peak->pulseGraph = channel;
          pulseList.push_back(peak); //adding PMT pulse object to list

          //counting PMT pulses in modules and adding to histograms if needed
          if(module==0) {
            //histMod0->Fill(minAbsVal);
            pmtPulseCounter0++;
          } else if(module==1) {
            //histMod1->Fill(minAbsVal);
            pmtPulseCounter1++;
          }else if(module==2) {
            //histMod2->Fill(minAbsVal);
            pmtPulseCounter2++;
          }else if(module==3) {
            //histMod3->Fill(minAbsVal);
            pmtPulseCounter3++;
          }
        }
           }
           delete channelLine;
      delete channel;
      delete channelUnfilt;
      delete peak;
    }
   }
 }


 cout << "Total number of pulses: " << pmtPulseCounter << endl;
 cout << "Number of pulses in mod0: " << pmtPulseCounter0 << endl;
 cout << "Number of pulses in mod1: " << pmtPulseCounter1 << endl;
 cout << "Number of pulses in mod2: " << pmtPulseCounter2 << endl;
 cout << "Number of pulses in mod3: " << pmtPulseCounter3 << endl;

 //check for muons in data
 list<Muon*> muonList = muonAnalysis(pulseList);
 displayMuons(muonList); //print muon properties to screen
```

```
 /*
 //draw histograms for minimum analysis if needed
 new TCanvas();
 histMod0->Draw();
 new TCanvas();
 histMod1->Draw();
 new TCanvas();
 histMod2->Draw();
 new TCanvas();
 histMod3->Draw();
 */
}

//_____________________________________________________________________________
//function to reduce the repeating noise structure in each channel line
TObjArray *reduceNoise(TObjArray* channelLine) {
 TObjArray *reducedLine = new TObjArray();
 TGraph *firstChan = (TGraph*)channelLine->At(0);
 Double_t *x = firstChan->GetX();
 const int n = firstChan->GetN();
 Double_t average[512]={0}; //create array for average voltage values of channel line
 for(int chanID=0; chanID<8; chanID++) { //loop over channels to create average
  TGraph *chanFirst = (TGraph*)channelLine->At(chanID);
  for(int point=0; point<n; point++) {
   average[point]+= (chanFirst->GetY()[point])/8;
  }
  //delete chanFirst;
 }
 TGraph *avGraph = new TGraph(n,x,average);
 for(int chanID=0; chanID<8; chanID++) { //loop over channels to reduce channels
  TGraph *chanSecond = (TGraph*)channelLine->At(chanID);
  Double_t reduced[512]={0};
  for(int point=0; point<n; point++) {
   reduced[point] = chanSecond->GetY()[point] - avGraph->GetY()[point];
  }
  delete chanSecond;
  x = chanSecond->GetX();
  TGraph *redChannel = new TGraph(n, x, reduced);
  reducedLine->Add(redChannel);
 }
 return reducedLine;
}

//_____________________________________________________________________________
//function to correct offset of each individual graph
TObjArray *correctOffset(TObjArray* channelLine) {
 TObjArray *correctedLine = new TObjArray();
 TGraph *firstChan = (TGraph*)channelLine->At(0);
 Double_t *x = firstChan->GetX();
 const int n = firstChan->GetN();
 for(int chanID=0; chanID<8; chanID++) { //loop over channels to correct channel offset
```

```
   TGraph *channel = (TGraph*)channelLine->At(chanID);
   Double_t mean = channel->GetMean(2);
   Double_t corrected[512]={0};
   for(int point=0; point<n; point++) {
    corrected[point] = channel->GetY()[point] - mean;
   }
   // delete channel;
   TGraph *corChannel = new TGraph(n, x, corrected);
   correctedLine->Add(corChannel);
  }
  return correctedLine;
}


//_____________________________________________________________________________
//method to determine characteristics of graphs and return them as a PmtPulse object
PmtPulse *peakAnalysis(TGraph* channel) {
  PmtPulse *peak = new PmtPulse();
  Int_t numPoints = channel->GetN();

  //Minimum
  Double_t mean = channel->GetMean(2);
  Double_t min = mean;
  Double_t *yVal = channel->GetY();
  Int_t minPos = -1;
  for(int i=0;i<numPoints;i++) {
   if(yVal[i]<min) {
    min=yVal[i];
    minPos=i;
   }
  }
  Double_t minAbs = abs(min);
  peak->minAbs = minAbs;
  peak->minPos = minPos;

  //FWHM
  Int_t fwhm = -1;
  Int_t bin1 = -1;
  Int_t bin2 =numPoints;
  for(int i=minPos;i>0;i--) { //finding first bin
   if(yVal[i]<=min/2) {
    bin1 = i;
    break;
   }
  }
  for(int i=minPos; i<numPoints; i++) { //finding second bin
   if(yVal[i]>=min/2 && i>minPos) {
    bin2 = i;
    break;
   }
  }
  fwhm = bin2-bin1;
```

```
 peak->fwhm = fwhm;

 //Area
 Double_t area = -1;
 for(int i=minPos; i>10; i--) {
  if(yVal[i]<mean) {
   area += yVal[i];
  } else {
    break;
  }
 }
 for(int i=minPos; i<(numPoints-10); i++) {
  if(yVal[i]<mean) {
    area += yVal[i];
  } else {
    break;
  }
 }
 area = abs(area);
          peak->area = area;

 return peak;
}


//_____________________________________________________________________________
//method to determine whether peak is PMT pulse candidate or not
Bool_t pmtCheck(PmtPulse *peak) {
 Bool_t isPMTPulse = false;
 Bool_t minTrue = false;

 //getting peak values
 Double_t minAbsVal = peak->minAbs;
 Double_t fwhm = peak->fwhm;
 //Double_t area = peak->area; //a further area criterion should be included after detector data is
enhanced
 Int_t module = peak->module;

 //threshold values can be changed depending on the detector settings
 if (module==0 && minAbsVal>20 ) {minTrue = true;}
 else if (module==1 && minAbsVal>25) {minTrue = true;}
 else if (module==2 && minAbsVal>25) {minTrue = true;}
 else if (module==3 && minAbsVal>25) {minTrue = true;}

 //defining threshold for fwhm to exclude voltage flucutation peaks
 if(minTrue && fwhm<150 && fwhm>20) {isPMTPulse=true;}

 return isPMTPulse;
}

//_____________________________________________________________________________
//method to display properties of individual muons
```

```
void displayMuons(list<Muon*> muonList) {
  Int_t muonCounter = 1;
  cout << "There was a total number of " << muonList.size() << " muons detected\n";
  for (list<Muon*>::iterator muonNumber=muonList.begin(), end=muonList.end();
muonNumber!=end; ++muonNumber) {
    cout << "Muon number " << muonCounter << ":\n";
    muonCounter++;

    //collecting muon properties
    Int_t entry = (*muonNumber)->entry;

    //printing muon properties
    cout << "Module 0 was crossed in channel " << (*muonNumber)->chanMod0 << " at t = " <<
(*muonNumber)->timeMod0 << "ns\n";
    cout << "Module 1 was crossed in channel " << (*muonNumber)->chanMod1 << " at t = " <<
(*muonNumber)->timeMod1 << "ns\n";
    cout << "Module 2 was crossed in channel " << (*muonNumber)->chanMod2 << " at t = " <<
(*muonNumber)->timeMod2 << "ns\n";
    cout << "Module 3 was crossed in channel " << (*muonNumber)->chanMod3 << " at t = " <<
(*muonNumber)->timeMod3 << "ns\n";

    //drawing muon graphs if not too many muons found
    if(muonList.size()<4) {
      drawGraph((*muonNumber)->pulseMod0, entry, 0, (*muonNumber)->chanMod0);
      drawGraph((*muonNumber)->pulseMod1, entry, 1, (*muonNumber)->chanMod1);
      drawGraph((*muonNumber)->pulseMod2, entry, 2, (*muonNumber)->chanMod2);
      drawGraph((*muonNumber)->pulseMod3, entry, 3, (*muonNumber)->chanMod3);
    }
  }
}

//_____________________________________________________________________________
//checks if PMT pulse occurs in all modules at same time plus travel time and position error
list<Muon*> muonAnalysis(list<PmtPulse*> pulseList) {
  list<Muon*> muonList;
  Muon *muonCandidate = new Muon;
  Double_t travelTimePos = 14; //time of muon to travel from one to the next module plus error in
minimum position
  Double_t travelTimeNeg = -7; //position error minus travel time

  //loop over module 0
  for (list<PmtPulse*>::iterator pulseIt0=pulseList.begin(), end0=pulseList.end(); pulseIt0!=end0;
++pulseIt0) {
     if ((*pulseIt0)->module == 0) {
     Double_t timeMod0 = (*pulseIt0)->eventTime;

     //module 1 check
     for (list<PmtPulse*>::iterator pulseIt1=pulseList.begin(), end1=pulseList.end(); pulseIt1!=end1;
++pulseIt1) {
       Double_t timeMod1 = (*pulseIt1)->eventTime;
       Double_t deltaTime01 = timeMod1 - timeMod0;
```

```
      if ((*pulseIt1)->module == 1 && deltaTime01<=travelTimePos && deltaTime01>=travelTimeNeg)
{

       //module 2 check
       for (list<PmtPulse*>::iterator pulseIt2=pulseList.begin(), end2=pulseList.end(); pulseIt2!=end2;
++pulseIt2) {
         Double_t timeMod2 = (*pulseIt2)->eventTime;
         Double_t deltaTime12 = timeMod2 - timeMod1;
         if ((*pulseIt2)->module == 2 &&  deltaTime12<=travelTimePos &&
deltaTime12>=travelTimeNeg) {
          cout << "Match in first three modules detected" << endl;

          //module 3 check
          for (list<PmtPulse*>::iterator pulseIt3=pulseList.begin(), end3=pulseList.end();
pulseIt3!=end3; ++pulseIt3) {
            Double_t timeMod3 = (*pulseIt3)->eventTime;
            Double_t deltaTime23 = timeMod3 - timeMod2;
            if ((*pulseIt3)->module == 3 &&  deltaTime23<=travelTimePos &&
deltaTime23>=travelTimeNeg) {
             cout << "Muon detected!" << endl;

             //adding properties of muon
             muonCandidate->entry = (*pulseIt0)->entry;
             //module 0 properties
             muonCandidate->timeMod0 = timeMod0;
             muonCandidate->chanMod0 = (*pulseIt0)->channel;
             muonCandidate->pulseMod0 = (*pulseIt0)->pulseGraph;
             //module 1 properties
             muonCandidate->timeMod1 = timeMod1;
             muonCandidate->chanMod1 = (*pulseIt1)->channel;
             muonCandidate->pulseMod1 = (*pulseIt1)->pulseGraph;
             //module 2 properties
             muonCandidate->timeMod2 = timeMod2;
             muonCandidate->chanMod2 = (*pulseIt2)->channel;
             muonCandidate->pulseMod2 = (*pulseIt2)->pulseGraph;
             //module 3 properties
             muonCandidate->timeMod3 = timeMod3;
             muonCandidate->chanMod3 = (*pulseIt3)->channel;
             muonCandidate->pulseMod3 = (*pulseIt3)->pulseGraph;

             muonList.push_back(muonCandidate); //adding muon to list
            }
           }
          }
         }
        }
       }
      }
     }
 return muonList;
}
```

```
//_____________________________________________________________________________
//method to display graph
void drawGraph(TGraph* channel, Int_t i, Int_t module, Int_t chan) {
 new TCanvas();
 char title[100];
 sprintf(title, "entry = %d, module = %d, chan = %d", i, module, chan);
 channel->SetTitle(title);
 channel->Draw("alp");
}

//_____________________________________________________________________________
//method to filter high frequency noise (adapted from pulseFinder.C script)
TGraph *getBoxCar(TGraph *grWave, Int_t halfWidth)
{
 //Just do this the lazy way for now
 Double_t *inY = grWave->GetY();
 Double_t *inX = grWave->GetX();
 Int_t length=grWave->GetN();
 Double_t *smoothY = new Double_t[length];
 for(int i=0;i<length;i++) {
  smoothY[i]=0;
  if(i<halfWidth || length-i<=halfWidth) {
   int countVals=0;
   for(int j=i-halfWidth;j<=i+halfWidth;j++) {
        if(j>=0 && j<length) {
         smoothY[i]+=inY[j];
         countVals++;
        }
   }
   //    cout << i << "\t" << countVals << endl;
   smoothY[i]/=countVals;
  }
  else {
   for(int j=i-halfWidth;j<=i+halfWidth;j++) {
        smoothY[i]+=inY[j];
   }
   smoothY[i]/=1+2*halfWidth;
  }
 }
 TGraph *grSmooth = new TGraph(length,inX,smoothY);
 delete [] smoothY;

 return grSmooth;
}
```