Besondere Lernleistung

Auswertung von Messungen des Myonenflusses im Dresdner Felsenkeller

Maximilian Kotz

12. Dezember 2016

Inhaltsverzeichnis

1. Einleitung und Motivation

In der Physik sind Experimente von entscheidender Bedeutung, um neue Erkenntnisse zu erlangen.
Dabei werden die Experimente immer komplizierter, um immer genauere Messwerte zu erhalten. Bei der Low Background Physik untersucht man Ereignisse, welche nur sehr selten eintreten. Dies erfordert einen sehr geringen „Störuntergrund“, der durch radioaktive und kosmische Strahlung hervorgerufen wird, weshalb man bestimmte Experimente in Untergrundlaboren durchführt. [1]

Ein solches Niederniveaumesslabor befindet sich in den Stollen des Eiswurmlagers auf dem Gelände der ehemaligen Brauerei Felsenkeller, welches vom VKTA Rossendorf e.V.[1] betrieben wird. [2] In Abbildung 1 kann man es in Stollen IV eingezeichnet erkennen.

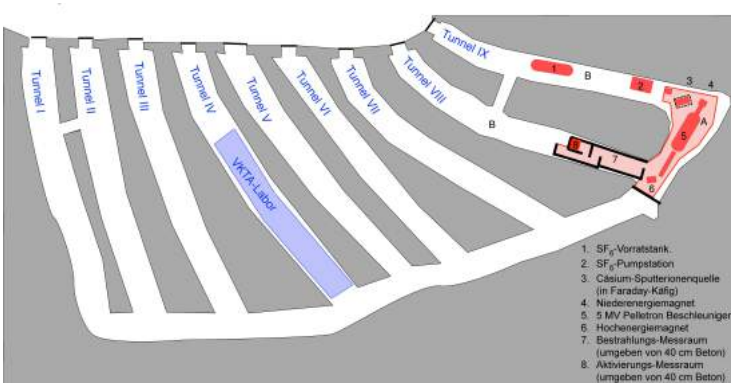


Abbildung 1: Lageplan des Felsenkellers

Diese Stollen dienten früher zur Lagerung von Eis, welches für die Bierproduktion benötigt wurde. Sie besitzen eine senkrechte Felsüberdeckung von ungefähr 50 m. Diese Stollen will die TU Dresden nutzen um dort einen Teilchenbeschleuniger aufzustellen und entsprechende Experimente, die eine geringe Hintergrundstrahlung benötigen, durchzuführen. Die geplanten Standorte sind die Stollen VIII und IX.

Da sich der Felsenkeller in der Nähe ehemaliger Uranabbaugebiete befindet, sind die Alpha-Strahler ^{218}Rn und ^{219}Rn, die bei der Uran-Radium- bzw. Uran-Actinium-Zerfallsreihe entstehen, ein wesentlicher Bestandteil der Hintergrundstrahlung. Um eine Ansammlung dieses Gases zu verhindern und so die Strahlung zu reduzieren, werden die Laborräume aktiv belüftet. [2]
Die meiste Strahlung ist somit auf kosmische Myonen zurückzuführen. Sie besitzen eine hohe Durchdringungsfähigkeit, sodass trotz der 50 m Gesteinsschicht über den Messgeräten noch viele in den Tunneln nachweisbar sind. Eine weitere Abschirmung durch Blei und Beton würde dies kaum ändern. Um die Hintergrundereignisse weiter zu reduzieren gibt es die Möglichkeit einer sogenannten aktiven Abschirmung. Dabei wird der Versuchsaufbau von Strahlungsdetektoren umgeben, die durchquerende Myonen registrieren und so Ereignisse von außen von denen im Experiment unterscheiden können. Für diese Detektoren verwendet man zumeist Szintillatoren.
Da großflächige Szintillatoren teuer sind, ist es wünschenswert die Richtung zu kennen,

[1] Strahlenschutz, Analytik und Entsorgung Rossendorf e. V.

aus der die meisten Myonen kommen, um die aktive Abschirmung auf diese Bereiche zu beschränken. Daraus ergab sich für diese Arbeit die Hauptmotivation: die Untersuchung der Winkelabhängigkeit des Myonenflusses im Felsenkeller Dresden.

Die Messungen, die im Rahmen dieser Arbeit durchgefüht wurden, konzentrierten sich somit auf die Bestimmung der Winkelabhängigkeit des Myonenflusses im Bereich des späteren Detektors. Zusätzlich wurden bei dieser Gelegenheit Messungen im benachbarten VKTA Labor durchgeführt.

2. Myonen

In den folgenden Abschnitten soll ein Überblick über das Myon (µ) gegeben werden.

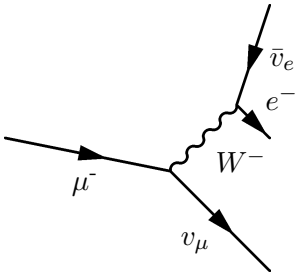
2.1. Eigenschaften

Bei einem Myon handelt es sich um das geladene Lepton der 2. Familie; es entspricht somit dem Elektron aus der 1. Familie. Es besitzt eine einfache negative elektrische Ladung, einen Spin von 1/2 und eine schwache Ladung von - 1/2. Im Gegensatz zum Elektron ist ein Myon um ein Vielfaches schwerer, außerdem zerfällt es rasch. Natürlich besitzt es auch ein dazugehörendes Neutrino, sowie entsprechende Antiteilchen.

2.2. Zerfall

Myonen sind instabil und zerfallen über die schwache Wechselwirkung in zwei Neutrinos und ein Elektron, beziehungsweise Positron.

Feynman-Diagramm des µ⁻ Zerfalls:



Der Zerfallsprozess ist exponentiell mit der Halbwertszeit T=1,523 µs, daraus folgt eine mittlere Lebensdauer von 2,197 µs (ohne Einfluss sonstiger Materie).
Zusätzlich können langsame µ⁻ vom elektromagnetischen Feld eines Atomkerns eingefangen und absorbiert werden, deshalb ist die mittlere Lebensdauer für µ⁻ etwas geringer als für μ^+.

$$\mu^- + p \rightarrow n + v_\mu \tag{1}$$

Dadurch verkürzt sich bei negativ geladenen Myonen die Lebensdauer abhängig vom umgebenden Stoff, dies ist u.a. der Grund dafür, dass Gestein den Myonenfluss mindert.

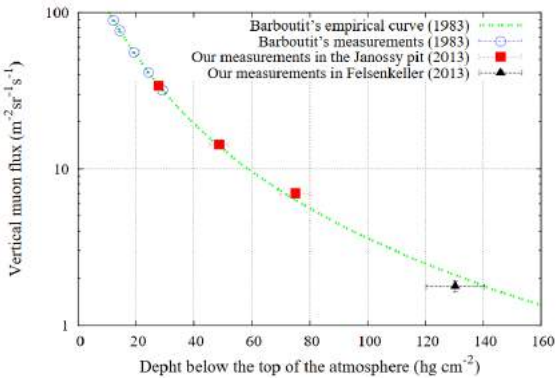


Abbildung 2: Abschirmung des Myonenflusses [3]

Man sieht, wie der Myonenfluss mit zunehmender Mächtigkeit der Gesteinsschicht abnimmt. So ist er im Felsenkeller bereits um fast 2 Größenordnungen kleiner, als oberirdisch, deshalb liegen einige Teilchenbeschleuniger unterirdich.

2.3. Entstehung

Myonen entstehen aufgrund der Wechselwirkung der kosmischen Strahlung mit der Atmosphäre. Diese, hauptsächlich aus Protonen (87%) und α-Teilchen (12%) bestehende Strahlung aus dem Weltall, kollidiert in der Atmosphäre mit Atomkernen, dabei entstehen extrem kurzlebige Teilchen, die Sekundärstrahlung.

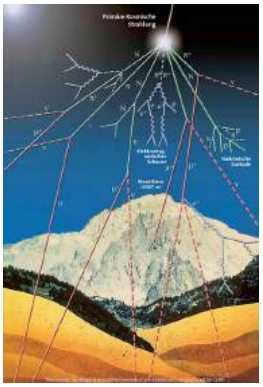


Abbildung 3: Teilchenschauer [4]

Pionen (π) sind die Hauptprodukte dieser Reaktionen.

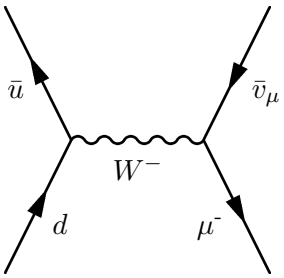
$$p + p \rightarrow p + n + \pi^{+} \tag{2}$$

$$p + n \rightarrow p + p + \pi^{-} \tag{3}$$

Mesonen[2] besitzen Lebensdauern von teilweise weniger als 10^{-8} s, bei ihrem Zerfall werden Myonen frei.

[2]Pionen sind eine Art der Mesonen

Feynman-Diagramm des π^- Zerfalls:



Diese Myonen entstehen typischerweise in einer Höhe von 10 bis 20 km und bewegen sich mit rund 99,8% der Lichtgeschwindigkeit (in Bezug zur Erde). Dadurch erhöht sich die mittlere Lebensdauer der Myonen für einen Beobachter um ca. den Faktor 16, sodass ihre mittlere zurückgelegte Wegstrecke ungefähr 9,6 km beträgt und viele von ihnen auf der Erdoberfläche nachgewiesen werden können. [5, 6]

Typischerweise treffen 200 Myonen pro Quadratmeter und Sekunde auf Meereshöhe auf [6], wobei der Myonenfluss winkelabhängig ist. Es gilt dafür näherungsweise für $-90° < \alpha < 90°$:

$$I(\alpha) = I_0 * \cos(\alpha)^n \tag{4}$$

Hierbei ist der Exponent n von der Energie der Myonen abhängig. Dieser Zusammenhang wird in Abbildung 4 dargestellt.

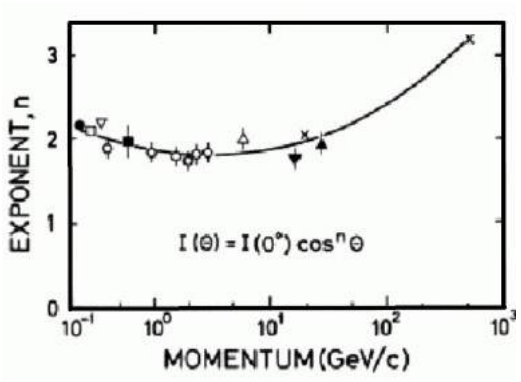


Abbildung 4: Abhängigkeit des Exponenten vom Impuls der Myonen [7]

Unser Teleskop misst im Durchschnitt eine Verteilung für n = 2,1.
Diese Verteilung gilt allerdings nur unter freiem Himmel, da Myonen von Gestein teilweise abgeschirmt werden.

2.4. Wechselwirkungen mit Materie

Myonen wechselwirken auf unterschiedliche Weise mit umgebender Materie, wobei bei verschiedenen Energien entsprechend andere Komponenten wirken. So dominieren bei geringen Energien der Energieverlust durch unelastische Stöße mit der Elektronenhülle der Atome des Stoffes, während bei hohen Energien die Bremsstrahlung einen erheblichen Effekt besitzt. [8]

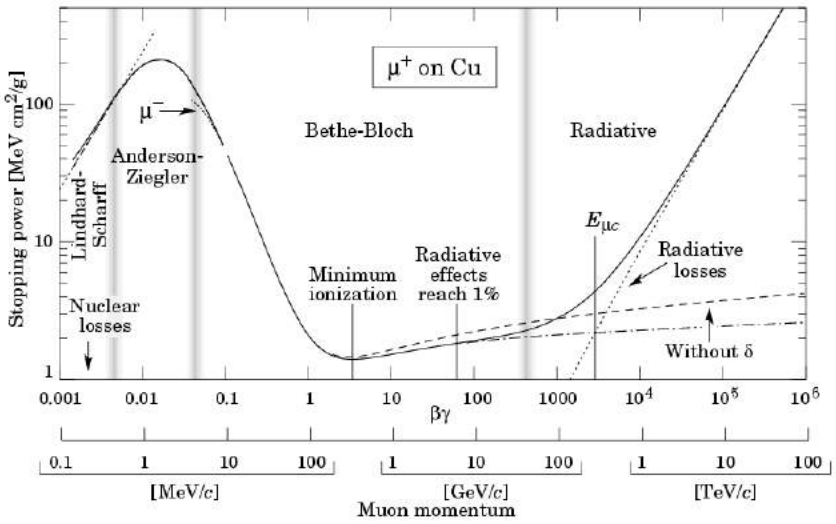


Abbildung 5: Energieverlust eines μ^+ [9]

Die in der Abbildung 5 angedeuteten Energieverluste bei niedrigen Energien spielen bei kosmischen Myonen zunächst keine Rolle. Diese kommen durch elastische Stöße mit dem Target [3] zustande.

2.4.1. Energieverlust durch Kollisionen

Wenn ein geladenes Teilchen einen Stoff durchquert, so wirkt zwischen ihm und den Elektronen der Atomhülle die Coulombkraft. Diese Kraft sorgt für eine Impulsänderung, welche eine Energieabgabe des Teilchens nach sich zieht. Die Atome des Target werden dabei angeregt bzw. ionisiert. Berechnet werden kann dieser Verlust durch die Bethe-Bloch-Formel: [11, 12]

$$-\frac{dE}{dx} = \frac{4\pi n z^2}{m_e v^2} \cdot \left(\frac{e^2}{4\pi\varepsilon_0}\right)^2 \cdot \left[\ln\left(\frac{2m_e v^2}{I \cdot \left(1-\frac{v^2}{c^2}\right)}\right) - \frac{v^2}{c^2}\right] \tag{5}$$

Dabei ist z die Ladungszahl des Teilchens, v seine Geschwindigkeit und n die Elektronendichte, für die gilt:

$$n = \frac{Z\rho}{Au} \tag{6}$$

Z ist die Ordnungszahl, A die Massenzahl, ϱ die Dichte und u die atomare Masseneinheit. [11] I ist das mittlere Anregungspotenzial und ist stoffspezifisch, für die etwa gilt: $I = 10eV \cdot Z$

In Formel 5 ist zu erkennen, dass bei geringen Energien der Energieverlust indirekt propotional zu v^2 ist, während er bei großen Energien, aufgrund der relativistischen Massezunahme, logarithmisch steigt. Daraus ergibt sich, dass Teilchen, kurz bevor sie ihre gesamte Energie verloren haben, am meisten Energie verlieren, diesen Punkt nennt man Bragg-Peak. Dies nutzt man unter anderem in der Medizin zur Tumorbehandlung

[3] *engl. Zielscheibe*, Probenmaterial, das beim Beschuss mit Strahlung mit ihr wechselwirkt [10]

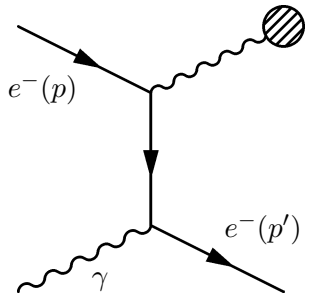
durch Protonenbestrahlung. Bei diesem Verfahren wird der größte Teil der Strahlung an den Tumor abgegeben und nicht an das umliegende Gewebe.

Die Bethe-Bloch-Formel verliert bei sehr hohen Energien ihre Aussagekraft, deshalb wurde eine sogenannte kritische Energie E_c eingeführt. Bis zum Erreichen dieses Wertes von E_c ist der Energieverlust durch Ionisation größer, als der durch Bremsstrahlung. Diese kritische Energie ist materialabhängig. Bei geringen Energien, also geringen Geschwindigkeiten verliert sie ebenfalls ihre Gültigkeit, wenn die durchquerenden Teilchen beginnen Hüllenelektronen mit sich zu führen, da sich so ihre effektive Ladung reduziert.

2.4.2. Energieverlust durch Bremsstrahlung

Bei sehr hochenergetischen Myonen gewinnt der Energieverlust durch Bremsstrahlung immer mehr an Bedeutung, da dieser Verlust direktproportional zur Energie des Teilchens ist.
Zu Bremsstrahlung kommt es, wenn ein Teilchen im Target durch Coulombkräfte abgelenkt wird, dabei ändert sich der Geschwindigkeitsvektor des Teilchens und es erfolgt die Emission eines Photons, da es sich um ein geladenes Teilchen handelt.

Beispiel eines Feynman-Diagramm für die Entstehung von Bremsstrahlung:[12]



Bremsstrahlung wird in Röntgenröhren zur Erzeugung von Röntgenstrahlung genutzt, dazu reichen bei Elektronen bereits Energien von 25 keV, während bei Myonen die Bremsstrahlung erst bei höheren Energien entscheidend wird. Dies hängt natürlich vom umgebenden Material ab, befindet sich bei Myonen aber in Größenordnungen von einigen 100 GeV. [12]

2.4.3. Tscherenkow-Strahlung

Grundsätzlich ist der Energieverlust durch diesen Effekt untergeordnet, doch auf ihm fußen bestimmte Detektionsverfahren, die dieses Licht detektieren können, deshalb soll es nicht unerwähnt bleiben.

Zu dieser Abstrahlung kommt es, wenn sich ein geladenes Teilchen mit einer größeren Geschwindigkeit bewegt als die Lichtgeschwindigkeit im Target. Grund dafür ist, dass durch das elektrische Feld des Teilchens die Atome im Target ionisiert werden und die so entstandenen elektromagnetischen Dipole Photonen aussenden. Bei Unterlichtgeschwindigkeit interferieren die Dipole, sodass kein Licht sichtbar wird; bei Überlichtgeschwindigkeit geschieht dies nicht und es kommt zur Tscherenkow-Strahlung.

Diese Strahlung kann von sehr empfindlichen Photomultiplern wahrgenommen werden. Diese nutzen den photoelektrischen Effekt, um selbst geringste Mengen Licht zu

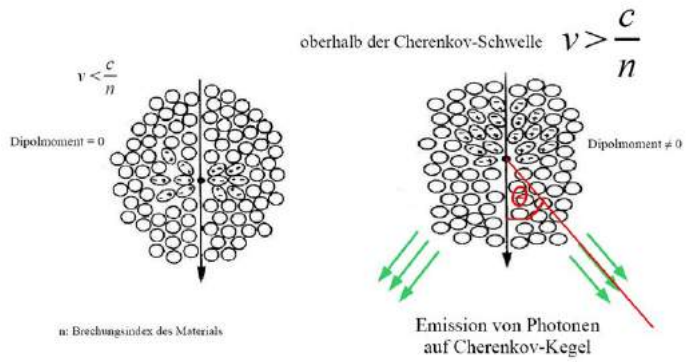


Abbildung 6: Durchgang eines geladenen Teilchens bei verschiedenen Geschwindigkeiten [13]

messen. Durch die Detektion eines solchen Lichts kann ein Myon nachgewiesen werden. Mit bloßem Auge kann man dieses Licht in Kernreaktoren beobachten. Es ist der Grund dafür, dass das Kühlwasser, welches die Brennstäbe umgibt, blau leuchtet. Dieses Licht wird dabei von der entstehenden radioaktiven Strahlung erzeugt.

2.5. Detektion

Beim Nachweis von Myonen bedient man sich der Wechselwirkung dieses Teilchens mit der umgebenden Materie.

Eine Variante besteht in der Detektion des Tscherenkow-Lichtes. Bei diesem Verfahren ist eine Winkelauflösung allerdings nur schwer möglich. Außerdem sind diese Detektoren groß und schwer, aufgrund des benötigten Mediums (meist Wasser) in dem das Licht entsteht.
Ein anderes Verfahren ist die Verwendung von Szintillatoren. Dabei bedient man sich der Fähigkeit von Myonen Stoffe zu ionisieren. Der Szintillator besteht aus einem besonderen Material, welches sichtbares Licht bei Ionisation aussendet, das anschließend detektiert werden kann. Aber auch bei diesem Verfahren ist eine Winkelauflösung sehr aufwendig.

Bei dem verwendeten Myonenteleskop handelt es sich um einen Gasdetektor. In solchen Detektoren werden Elektroden, zwischen welchen eine Hochspannung anliegt, durch ein Gas getrennt. In Abbildung 7 ist der Aufbau, sowie des elektrische Feld in einem Gasdetektor zu sehen. Das Myon ionisiert die Gaskammer und ein Stromfluss kann gemessen werden. Durch die Verwendung mehrerer Kanäle und Kammern ist auch eine Winkelauflösung möglich.

Bei allen genannten Detektionsverfahren muss man Myonen von anderer ionisierender Strahlung unterscheiden. Dafür nutzt man die, im Gegensatz zu anderen Teilchen, lange Reichweite von Myonen aus. Man stellt also zwei oder mehr Detektorschichten hintereinander und wenn beide kurz hintereinander ein ionisierdendes Teilchen detektieren, handelt es sich mit hoher Wahrscheinlichkeit um ein Myon.

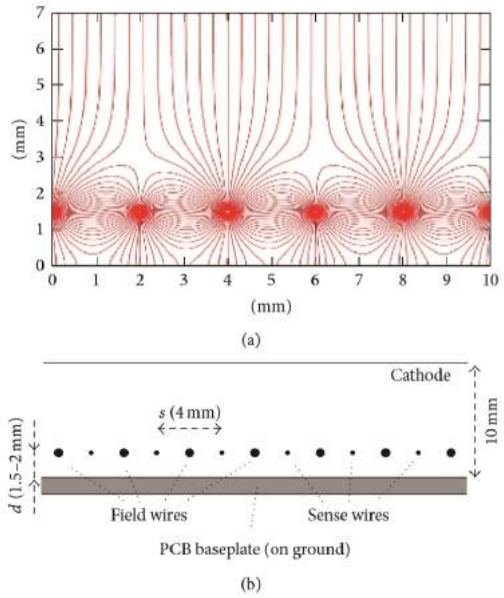


Abbildung 7: a) Simulation des elektrischen Feldes in einem Gasdetektor
b) Schematischer Aufbau einer Gasdetektorkammer [14]

Abbildung 8: Tscherenkow-Detektor für Myonen [15]

Ein Beispiel für eine solche Anordnung sieht man in Abbildung 8. Es wurden zwei Tscherenkow-Detektoren hintereinander angeordnet. Wenn nun ein geladenes Teilchen beide durchquert, dann wird es als Myon registriert, wenn nur ein Detektor einen Lichtblitz wahrnimmt, dann wird es ignoriert. An diesem Versuchsaubau sieht man, dass nur ein kleiner Raumwinkel von Myonen überhaupt beide Detektoren durchqueren kann. Dies sorgt für lange Messzeiten, außerdem wird deutlich, dass man für eine Winkelauflösung extrem viele dieser Detektoren zusammenschließen müsste.

3. Das Myonenteleskop

Das Myonenteleskop wurde in Budapest gebaut und entwickelt vom Wigner Forschungszentrum für Physik und wurde vom HZDR[4] angeschafft, um den kosmischen Mynenfluss im Felsenkeller zu untersuchen. Zusätzlich zum Teleskop existiert auch ein Programm inklusive Quellcode.

3.1. Aufbau und Funktionsweise

Bei dem verwendeten Myonenteleskop handelt es sich um einen Gasdetektor. Es befindet sich in Abbildung 9 rechts auf einem Stativ, welches eine Neigung des Teleskops im Winkel von 0°, 45° und 90° zur Horizontalen ermöglicht.

Abbildung 9: Aufgebautes Myonenteleskop mit Gasflasche

Im Wesentlichen besteht es aus sechs Close-Cathode-Chambers (CCC), die übereinander angeordnet sind und sensibel auf Myonen reagieren. Diese müssen ständig von Gas durchflossen werden; es wird eine Argon-Kohlenstoffdioxidmischung (82%-18%) verwendet, bei einem Gasfluss von ungefähr 0,5 $^{l}/_{h}$.[16]

3.1.1. Close-Cathode-Chamber

Die einzelnen Close-Cathode-Chamber sind wie in Abbildung 7 aufgebaut; es handelt sich dabei um eine asymmetrische Anordnung der Drähte, bei der sich die untere Kathode (0 V) deutlich näher an den Drähten befindet als die obere (-550 V). Es gibt dabei zwei Arten von Drähten, die Senswires und die Fieldwires. Diese sind abwechselnd, mit einem Abstand von 2 mm, angeordnet. Die Sw[5] (+1060 V) stellen die Anode dar, während die Fieldwires (-600 V) zur Detektion dienen. Die untere Kathode ist in 4 mm breite Streifen geteilt, die Pads, welche senkrecht zu den Fw [6] angeordnet sind und zur Detektion in der anderen Koordinatenachse dienen. Insgesamt gibt es jeweils 64 Fw und Pad pro CCC. [16, 17]

[4] Helmholtz-Zentrum Dresden-Rossendorf
[5] im folgenden als Abkürzung für Senswires
[6] im folgenden als Abkürzung für Fieldwires

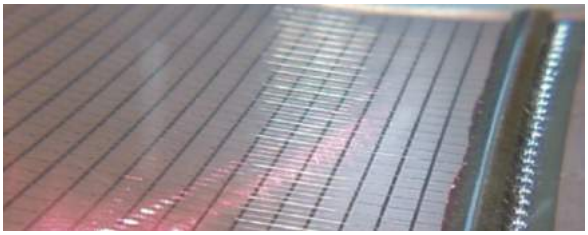


Abbildung 10: Wires und Pads eines Layers [17]

Durchquert nun ein Myon den CCC, so ionisiert es das darin befindliche Gas (Primärionisation), dadurch entstehen freie Elektronen. Diese freien Elektronen werden nun durch das elektrische Feld im oberen Bereich des CCC zu den Sw gezogen. Auf Höhe des Sw erhöht sich die elektrische Feldstärke nun durch die, abwechselnd mit den Sw verbauten, Fw. Dadurch werden die sich zu den Sw bewegenden Elektronen stark beschleunigt und lösen so durch weitere Ionisationsstöße eine Ladungslawine aus. Dieser Effekt bewirkt eine Vergrößerung des zu messenden Stromimpulses bei den Fw. Zu beachten ist, dass die Pads ähnlich weit von den Sw gelegen sind, wie die Fw, sodass auch auf sie Ionen gelangen und diese auch eine Stromfluss registrien.

Auf Grund von baulichen Ungenauigkeiten und physikalischen Einflüssen auf das Teleskop kann sich der Abstand der Drähte zur unteren Kathode ändern. Dies würde zu einer unterschiedlich starken Verstärkung von Myonen führen, welche die CCC an unterschiedlichen Positionen durchqueren. Um dies zu verhindern, wurde im Vorfeld, beim Bau eines Prototyps der CCC, untersucht, wie sich dieser unterschiedliche Abstand bei verschiedenen Spannungen auf die Verstärkung auswirkt.

Bei diesem Prototyp wurden die Fw und Sw schräg zur Horizontalen gespannt. Sie befanden sich in einem Abstand von rund 10 mm von der oberen Kathode und zwischen 1,5 bis 2 mm entfernt von den Pads, wie in Abbildung 11 schematisch dargestellt ist.

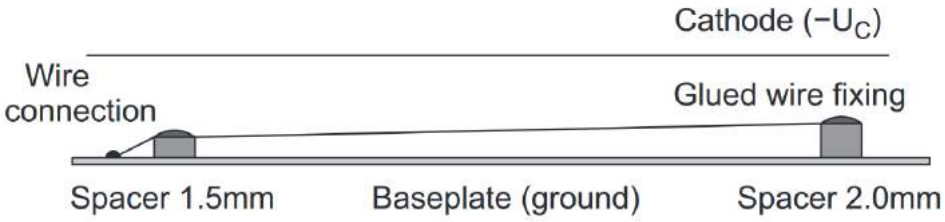


Abbildung 11: Abstand der Wires von den Kathoden [17]

Um nun die Verstärkung in Abhängigkeit vom Abstand der Wires zur unteren Kathode zu erhalten, wurde eine Betaquelle an verschiedene Positionen des CCC gebracht und die Verstärkung gemessen. Dies wurde für verschiedene Spannungsquotienten $^{\text{Fw-Spannung}}/_{\text{Sw-Spannung}}$ wiederholt. Das Ergebnis sieht man in Abbildung 12.

Um nun eine möglichst abstandsunabhängige Verstärkung zu erhalten, wurde für unser Teleskop ein Spannungsquotient von ungefähr -0,6 gewählt. Dies gewährleistet eine konstante Verstärkung, auch wenn der Abstand der Drähte von 2 mm zur unteren Kathode abweichen sollte.

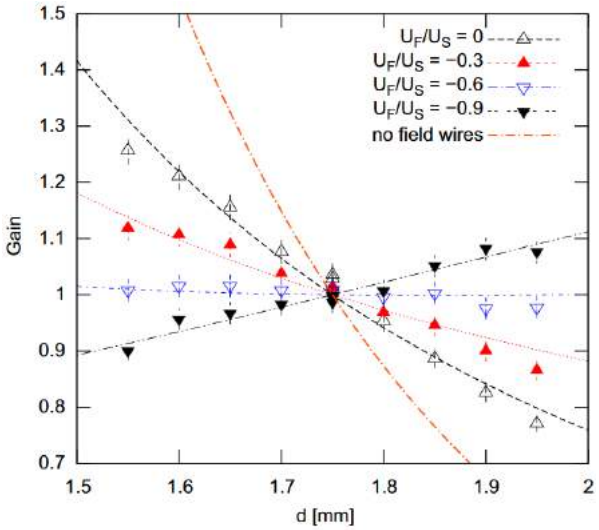


Abbildung 12: Verstärkung (Gain) in Abhängigkeit des Abstandes (d) zur unteren Cathode bei verschiedenen Spannungen [17]

3.1.2. Triggersystem

Das Messsystem benötigt ein sogenanntes Triggersystem, welches Alarm gibt, wenn es zu einem möglichen Myonendurchgang gekommen ist. Dazu dienen die vorhandenen CCC, da so Material, Gewicht und Stromverbrauch gespart werden kann. Alle Sw einer Schicht werden dazu genutzt, sodass es zu einem Signal kommt, wenn ein Myon irgendwo diesen Layer durchquert. Dieses Signal wird durch eine Trigger-FE-[7]Card verstärkt und diskriminiert. Dadurch erhält man ein Triggersignal, welches weiter gegeben werden kann. Diese Triggersignale der einzelnen Layer werden zusammengefasst. Wenn dabei mindestens 2 Layer getriggert wurden, so kommt es zum Triggersignal und ein Event wird ausgelöst.

Die Triggereffizienz ist ein wichtiger Parameter, da alle nicht getriggerten Myonen auch nicht weiter in der Offlineanalyse untersucht werden können. Dieser muss daher experimentell bestimmt werden. Dazu wurde das Triggersystem mit allen Layern und mit dem Verzicht auf einzelne Layer getestet, daraus wurde die Effizienz jeden Layers errechnet. Es zeigte sich, dass die Triggereffizienz proportional zur Trackingeffizienz ist. Dazu wird ein Track ohne den entsprechenden Layer gefittet und überprüft, ob der auf dem Layer extrapolierte Punkt dort den getroffenen Punkt treffen würde. Als getroffen gilt dabei, wenn der Punkt einen maximalen Abstand von 2 Wires hat. [18]

3.1.3. Messsystem

Die einzelnen Fw und Pad sind an FE-Karten angeschlossen, welche die Signale verstärken und diskriminieren. Wenn die FE-Karten das Triggersignal erhalten, so speichern sie das Signal in ihren Schieberegistern. Anschließend werden die Signale über SPI ausgelesen und schließlich gespeichert. Dazu dient das sogenannte DAQ[8], welches aus einem Raspberry Pi und dem DAQ-Board besteht. Dabei werden die Messdaten eines Runs in Textdatei auf einer SD-Card aufgezeichnet. Die Textdatei trägt den Namen Mts4, gefolgt

[7] Abkürzung für Front-End Elektronik
[8] Abkürzung für Data AcQuisition

von der Runnumber und der Endung .ebe (Event-by-Event). Jede Zeile der Datei repräsentiert dabei ein getriggertes Event. Die Datenstruktur ist beispielhaft in Abbildung 13 dargestellt.

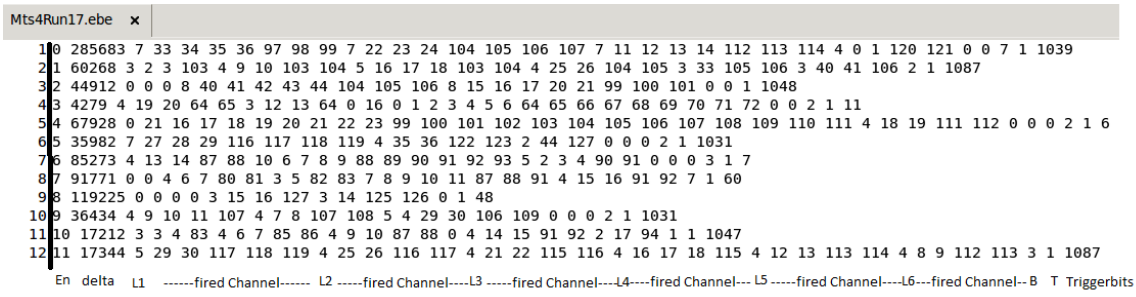


Abbildung 13: Beispiel aus Mts4Run17.ebe für Datenformat

Zunächst wird die Eventnumber (En) angegeben, gefolgt von der Zeit, die seit dem letzten Event vergangen ist (delta t). Nun kommen die Informationen über die Layer: die erste Zahl gibt dabei immer an, wie viele Channel in einem Layer getroffen wurden (L), die folgenden Zahlen geben den Zahlencode der getroffenen Channel wieder, dabei stehen Zahlen von 0 bis 63 für getroffene Fw und Zahlen von 64 bis 127 für Pad. Nun kommen drei Zahlen, die keine weitere Bedeutung für die Analyse haben, die Anzahl der verpassten Events während des Auslesevorgangs (B), sowie Informationen über das Triggering.

3.1.4. Interface

Der RasPi besitzt einen WiFi Stick. Nachdem er hochgefahren ist, kann er somit als Hotspot arbeiten und man kann sich mit ihm verbinden und anschließend über ssh [9] auf ihn zugreifen, dabei werden die üblichen Kommandozeilenbefehle verwendet. In Tabelle 1 sind die wichtigsten Befehle dazu aufgelistet. Diese benutzt auch das Programm.

[9] Secure Shell (SSH) ist ein Netzwerkprotokoll zur verschlüsselten Netzwerkverbindung mit einem entfernten Gerät [11]

Befehl	Beschreibung
ssh pi@*IPAdresse*	Verbindung mit Teleskop
ssh pi@*IPAdresse* 'head -n 1 *FileName RunNumber*.ebe & > /dev/null'	prüft ob Runnumber vergeben ist
scp ini/*.ini pi@*IPAdresse*:MtRpiDaq/. > /dev/null; ssh pi@*IPAdresse* 'cd MtRpiDaq; nohup sudo ./MtRpiDaq.run -f &> StdDump &'	startet Messung
ssh pi@*IPAdresse* 'sudo rm -f MtRpiDaq/Running 2> /dev/null; sudo killall MtRpiDaq.run 2> /dev/null	beendet Messung
ssh pi@*IPAdresse* 'cat MtRpiDaq/Running & > /dev/null'	prüft ob Messung läuft
scp pi@*IPAdresse*:*Verzeichniss** Measurements/	Kopieren von Messungen aus dem Verzeichniss auf Rechner
ssh pi@*IPAdresse* 'rm -f *Verzeichniss**'	löscht Daten vom Paspberry Pi

Tabelle 1: Übersicht wichtiger Befehle für das Teleskop

Informationen zur Hochspannung, Anodenstrom und Gasfluss müssen direkt am Detektor abgelesen und eingestellt werden, da der Raspberry Pi nicht darauf zugreift.

3.2. Offlineanalyse

Die Rohdaten müssen nach ihrer Messung analysiert werden, um aus ihnen den richtungsabhängigen Myonenfluss sowie einige Statistiken zur Messung zu erhalten. Ein solches Analyseprogramm war bereits vorhanden, musste aber aus mehreren Gründen überarbeitet werden:

1. Der Myonenfluss konnte nur aus einer einzigen Messung bestimmt werden, was eine umfassende Winkelabdeckung nicht erreichen konnte.
2. Der Myonenfluss wurde in Anstiegen dargestellt, was sich als höchst unintuitiv erwies.
3. Wie sich zeigte, konnte der Code an einigen Stellen erheblich vereinfacht werden.
4. Das existierende Programm nutzte nur Tracks, die in allen 6 Chambern getroffen wurden.
5. Außerdem konnte die Programmlaufzeit halbiert werden.

Wir entschieden uns für Root als Programmiersprache, diese Sprache ist an C++ angelehnt und auf die Analyse großer Datenmengen optimiert. Außerdem können mit ihr einfach GUI[10] realisiert werden.
Im Folgenden sind die wichtigsten Programmteile dargestellt.

3.2.1. Pre- und Hauptanalyse

Im Wesentlichen habe ich an diesem Programmabschnitt gearbeitet. Der Funktionsumfang entspricht zu großen Teilen dem existierenden Programm, allerdings wurden einige

[10]Graphical User Interface (GUI)

Funktionen verbessert und die Programmiersprache von C++ zu Root verändert. Das Ziel dieses Teils ist es, aus den Rohdaten die Myonentracks zu extrahieren.

Zunächst werden die Rohdaten aus einer Textdatei in einen TTree geschrieben, dabei handelt es sich um eine Klasse von Root, die zur Verwaltung großer Daten gedacht ist. Einen solchen TTree kann man sich als 2D-Array vorstellen. In der einen Dimension ist er offen, sodass beliebig viele Events untereinander angeordnet werden können. In der anderen Dimension ist der TTree in TBranches geordnet, welche den Zugriff auf Variablen, Strukturen und Objekte ermöglichen.

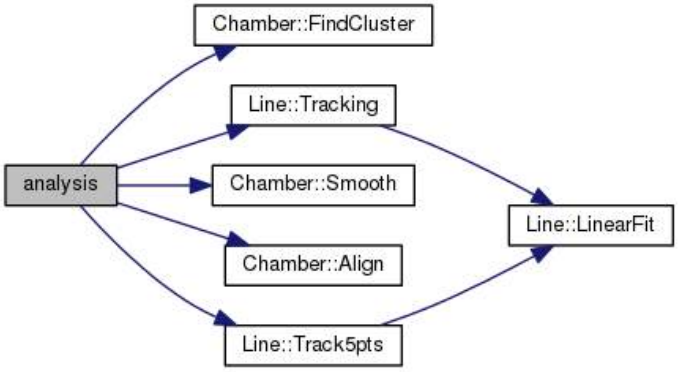


Abbildung 14: Funktionen der Analyse

Grundsätzlich werden nun die Myonen-Tracks gesucht und ihre Richtung bestimmt. In Abbildung 14 sind schematisch die einzelnen Funktionen der Analyse dargestellt. Zunächst werden durch die Funktion „FindCluster“ die Koordinaten der Punkte bestimmt, die durch den Myonen-Durchgang in jedem Chamber erzeugt wurden. Die Koordinate entspricht dem Schwerpunkt der getroffenen Channels, welcher allerdings durch das Smoothing und Alignment noch entsprechend korrigiert wird. Des Weiteren wird dieser Punkt noch durch einen zufälligen Wert verschmiert, da sonst auf Grund des Aufbaus nur bestimmte Positionen gemessen werden, obwohl eine kontinuierliche Verteilung vorliegt.

Diese Punkte werden nun im Tracking zu einer Linie gefittet, dabei wird nur die jeweils beste Linie verwendet (mehrere Möglichkeiten, wenn mehrere Cluster pro Chamber). Für einen Track müssen mindestens 5 Chamber getroffen werden und das mittlere Abweichungsquadrat darf nicht größer als 2 [11] sein.

Dies wird in 2 Schritten getan, zunächst werden in einer Preanalyse 2000 Tracks untersucht um statistische Aussagen über den Aufbau des Teleskops treffen zu können und anschließend kann in der Hauptanalyse die eigentliche Auswertung erfolgen.

Um einfach Statistiken zu verwalten, hilft erneut ein Root Objekt: THist. Dabei handelt es sich um ein Objekt, das als Histogramm dient und viele Funktionen, unter anderem auch eine grafische Ausgabe, unterstützt, die für diese Zwecke sehr hilfreich sind.
Es treten im Wesentlichen zwei, durch den Aufbau bedingte Fehler auf, die statistisch erfasst werden müssen. Zum einen weisen die einzelnen Wires aus baulichen Gründen eine

[11] dieser Wert nennt sich CHI_2_THRESHOLD

unterschiedliche Akzeptanz auf, was zu einer scheinbaren Verschiebung der Koordinaten hin zu diesen Wires führt. Dieser Fehler tritt mit einer Periodizität von vier Wires auf, wie man auch in Abbildung 15 sieht. Um diesen Fehler zu bestimmen geht man davon aus, dass jede der 4 Wires grundsätzlich mit derselben Wahrscheinlichkeit getroffen werden müsste, durch statistische Überprüfung kann die abweichende Affinität für Myonen ausgeglichen werden, dieser Wert wird Smoothingprobability genannt.

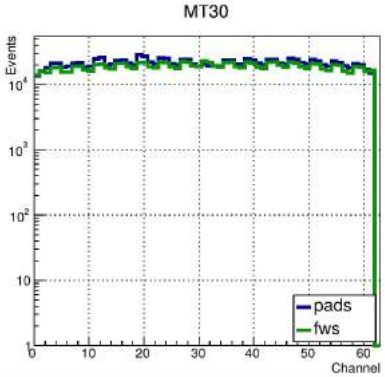


Abbildung 15: Hits der einzelnen Channels der MT30 Chamber bei Run 39

Der zweite Fehler, der korrigiert werden soll, entsteht durch die baulich bedingte Verschiebung der Chamber relativ zueinander (Abbildung 16). Durch die Differenz der Messwerte von der extrapolierten Position des Treffers kann eingeschätzt werden, wie die Chamber zueinander verschoben sind, dieser Effekt wird „Alignment“ [12] genannt. Die durchschnittliche Abweichung der Messwerte entspricht dabei dem Alignment des Chambers.

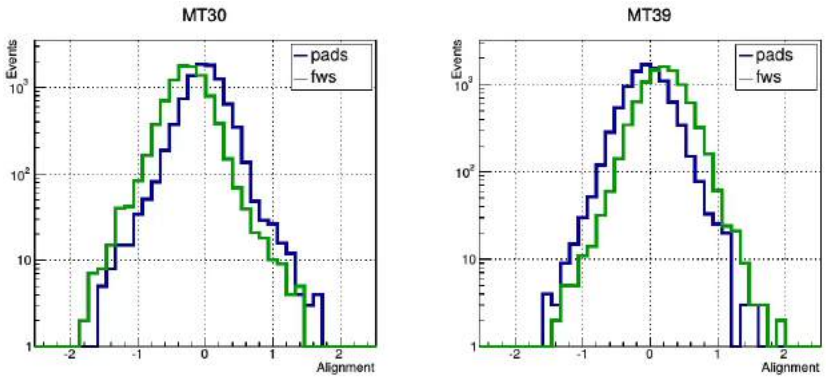


Abbildung 16: Abweichung der Punkte vom Linearfit des MT30 und MT39 Chambers bei Run 39

In der Hauptanalyse werden nun die Myonen-Tracks gesucht und ihre Richtung bestimmt. Die Tracks werden anschließend in ein Histogramm abhängig von ihrer Richtung gefüllt. Um später den tatsächlichen Myonenfluss zu erhalten, muss allerdings noch die

[12] engl.: Ausrichtung, Flucht

Akzeptanz und Effizienz berücksichtigt werden. Um diese Effizienz der Chamber zu berechnen, werden nun zu jedem Track zusätzlich Linien mit nur fünf Punkten gefittet. Wenn der sechste Punkt getroffen wurde gilt diese Linie als getrackt.

Die Ergebnisse werden in einem weiteren TTree für die nächsten Programmschritte gespeichert und einige Statistiken werden ausgegeben. Das Abspeichern dieser Zwischenergebnisse hat den Vorteil, dass man die Analyse, die sehr zeitintensiv ist, nicht bei jeder veränderten Einstellung in den nachfolgenden Programmabschnitten erneut durchführen muss.

Im Vergleich zu dem bereits existierenden Programm verwendet unser Programm auch Myonentracks, die nur von fünf Chambern registriert wurden. Dies erhöht bei gleicher Messzeit die Anzahl der Tracks, sodass man in kürzerer Zeit mehr Statistik errreicht. Außerdem läuft dieses Programm etwa nur halb so lange, was sich gerade bei großen Messdaten bemerkbar macht.

3.2.2. Polargauß

Dieser Teil wandelt die Ergebnisse der Analyse in Polarkoordinaten um und berechnet den Fehler. Der Myonenfluss und dessen Fehler werden als Histogramme dargestellt. Dies bedeutet, dass sie eine bestimmte Bingröße aufweisen müssen. Diese Größe ist variabel und wichtig sinnvoll zu wählen; natürlich bieten mehr Bins [13] eine höhere Auflösung, jedoch steigt der relative statistische Fehler, da es so weniger Treffer pro Bin gibt. Dazu muss man die Rotation des Runs angeben. Das Teleskop kann dabei entlang der drei Raumachsen gedreht werden. Das Programm unterstützt lediglich eine Rotation um zwei von ihnen, da eine gleichzeitige Rotation um drei Achsen einerseits unnötig und andererseits unpraktisch ist. In Abbildung 17 ist ein Beispiel zu sehen.

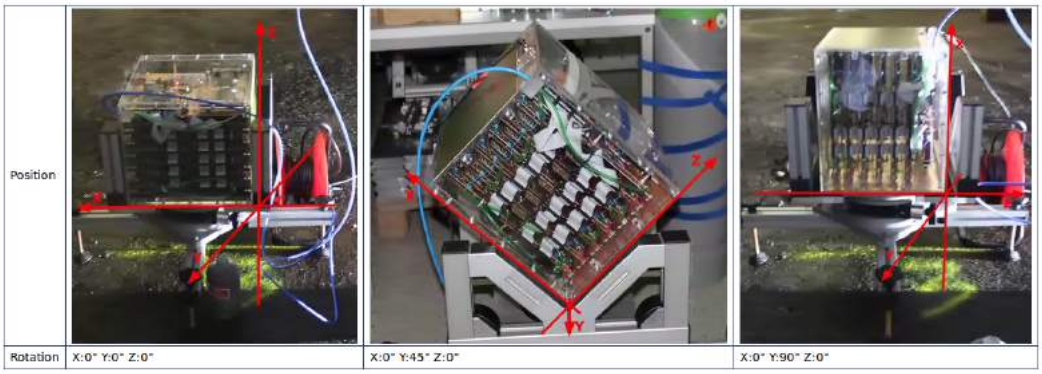


Abbildung 17: Beispiele für Rotationen des Myonenteleskop

Das Programm greift nun auf die Ergebnisse der Hauptanalyse zu und rotiert zunächst die Anstiege der Myonentracks um den entsprechenden Winkel. Da diese Anstiege sehr unintuitiv sind, werden nun daraus Polarkoordinaten berechnet. Es gilt folgender Zusammenhang:

[13] engl. Behälter; Unterteiung der Werteskale eines Histogramms Bereicher einer breite, die Bingröße genannt wird

$$mPad = \cos(\phi) \cdot \tan(\theta) \tag{7}$$
$$mFw = \sin(\phi) \cdot \tan(\theta) \tag{8}$$

Dabei ist mPad der Anstieg in Pad Richtung und mFw der Anstieg in Fieldwires Richtung. φ und ϑ sind die Polarkoordinaten, wobei ϑ der Abstand zum Zenit und φ der Winkel von Norden aus ist. Zu beachten ist, dass Myonen hauptsächlich von oben kommen und das Myonenteleskop den Richtungssinn der Myonentracks nicht erfassen kann, deshalb werden alle Tracks in positive theta Richtung interpretiert.

Anschließend wird die Akzeptanz der Tracks bestimmt, diese ist hauptsächlich von der Fläche abhängig, bei der diese Tracks registriert werden können und einem zusätzlichen winkelabhängigen Faktor. Diese Akzeptanz dient bei der Fluxberechnung zur Wichtung der Tracks. In Abbildung 18 sieht man den Graph der Akzeptanz, man sieht ein spitzes Maximum bei senkrechten Tracks.

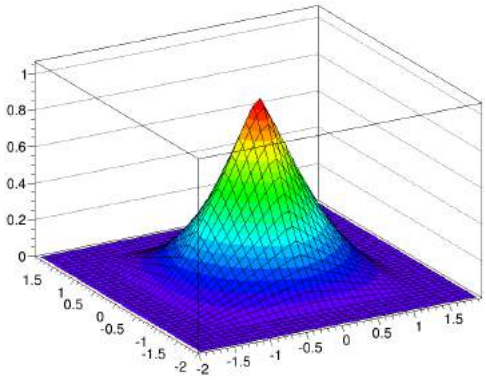


Abbildung 18: Acceptanz des Myonenteleskop

Um nun die Effizienz der Chamber zu bestimmen konnten wir wieder ein Root Objekt nutzen: TEfficiency. Dieses Objekt besteht im Wesentlichen aus zwei Histogrammen. In einem werden alle Tracks aufgezeichnet, in dem anderen nur die guten, also getrackten Ereignisse. Aus dem Verhältnis beider wird die Trackingeffizienz berechnet, welche bei der Berechnung des Flux berücksichtigt werden muss.

Diese Trackingeffizienz der Chamber (ChamberEff) gibt an, welcher Anteil der durchquerten Myonen auch als Myon von einem Chamber getrackt wird. Um nun die Trackingeffizienz für das gesamte Teleskop ε (11) zu erhalten, müssen die Effzienzen für 6 Punkte (9) und 5 Punkte Tracks (10) bestimmt und addiert werden.

$$trackingEff6 = \prod_{i=1}^{n} ChamberEff_i \tag{9}$$

$$trackingEff5 = \sum_{i=1}^{n} \frac{1 - ChamberEff_i}{ChamberEff_i} trackingEff6 \tag{10}$$

$$trackingEff = trackingEff6 + trackingEff5 \tag{11}$$

Die Effizienz des gesamten Teleskops ist in Abbildung 19 dargestellt. Man sieht deutlich, dass die Effizienz nur schwach winkelabhängig ist (bis zu einem ϑ von 45° [14]), danach sinkt sie auf 0. Somit kann man sagen, dass das Teleskop einen Öffnungswinkel von rund 90° aufweist.

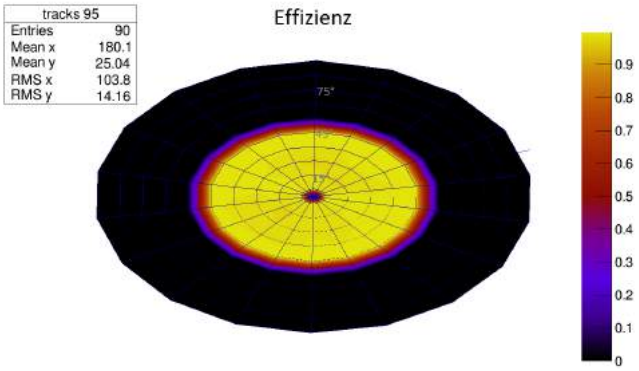


Abbildung 19: Effizienz des Myonenteleskops bei Run 95

Der Fluss Φ berechnet sich nun für jedes Bin, wie in Formel 12 zu sehen, aus der Anzahl der getrackten Myonen unter Berücksichtigung der Akzeptanz N, dem Raumwinkel des Bins Ω, der Detektorfläche A, der Zeit t und der Effizienz ε.

$$\Phi = \frac{N}{\Omega \cdot A \cdot t \cdot \epsilon} \tag{12}$$

Der Flux nützt alleine wenig, wenn man nicht weiß, wie groß der Messfehler ist. Die Hauptfehlerquelle liegt in der Endlichkeit unserer Stichprobe von Myonentracks begründet, was zu statistischen Fehlern führt. Dabei spielen zwei Arten von statistischen Fehlern eine Rolle, der statistische Fehler der Anzahl der getrackten Myonen ΔN und der statistische Fehler bei der Berechnung der Effizienz $\Delta\varepsilon$. Dabei sind beide Fehler unabhängig voneinander.

Für ΔN gilt:

$$\Delta N = \sqrt{N} \tag{13}$$

Somit ist der relative statistische Fehler von N monoton fallend.

Der statistische Fehler der Effizienz $\Delta\varepsilon$ entsteht durch die Fehlerfortpflanzung der statistischen Fehler der Effizienzberechnungen der einzelnen Chamber. Diese Fehler berechnet das TEfficiency Object selbstständig. Den gesamten Fehler $\Delta\varepsilon$ berechnet man, indem man die einzelnen Fehler der Chamber mithilfe des Gauß´schen Fehlerfortpflanzungsgesetzes auf die Formeln 10 bis 11 anwendet.

Der Fehler des Flusses $\Delta\Phi$ berechnet sich ebenfalls über das Gauß´sche Fehlerfortpflanzungsgesetz, wie in Formel 14 zu sehen.

$$\Delta\Phi = \sqrt{\left(\frac{\Delta N}{\Omega t A \epsilon}\right)^2 + \left(\frac{N \Delta\epsilon}{\Omega t A \epsilon^2}\right)^2} \tag{14}$$

[14] die scheinbar geringere Effizienz bei ϑ=0 °ist in diesem Fall nur ein Darstellungsfehler

Bins die leer sind, bekommen einen extrem großen Fehler, sodass sie bei der Histogrammaddition nicht berücksichtigt werden und man sieht, wo der Messbereich endet. Auf Grund des statistischen Fehlers ist es zu empfehlen, möglichst lange zu messen und die Binanzahl nicht unnötig groß zu gestalten, da sich sonst weniger Tracks in einem Bin befinden.

3.2.3. Histogrammaddition

Um eine vollständige Winkelabdeckung zu erhalten, müssen die einzelnen Messungen zusammengefasst werden.

Aus den Ergebnissen der einzelnen Runs wird das gewichtete Mittel errechnet. Dafür gelten folgende Formeln:

$$\bar{x} = \frac{\sum_{i=1}^{n} \frac{x_i}{\delta x_i^2}}{\sum_{i=1}^{n} \frac{1}{\delta x_i^2}} \tag{15}$$

$$\Delta\bar{x} = \sqrt{\frac{1}{\sum_{i=1}^{n} \frac{1}{\Delta x_i^2}}} \tag{16}$$

Durch diese Rechnung fallen Messwerte mit kleineren Fehlern mehr ins Gewicht, als welche mit großen. Im Programm werden dazu alle Bins der einzelnen Läufe erfasst und in einem neuen zusammengefasst. Zu beachten ist dabei, dass alle Runs mit derselben Bingröße analysiert werden müssen.

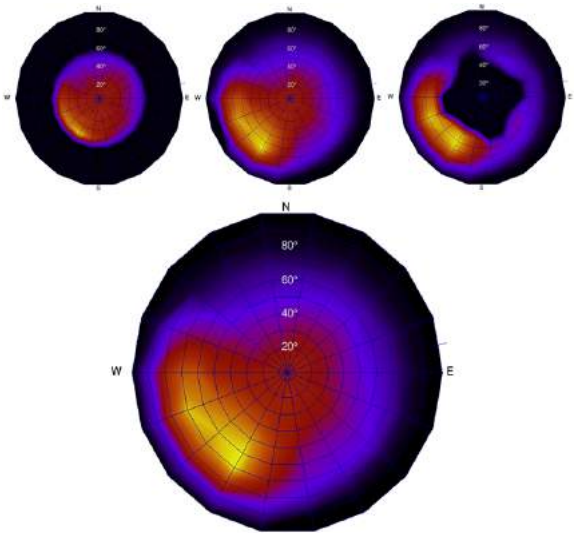


Abbildung 20: Flux aus verschiedenen Messungen zusammengefasst; oben von links nach rechts 0°, 45° und 90° Messungen; unten zusammengefasst

Wie man in Abbildung 20 sieht, kann durch das Zusammenfassen mehrerer Runs eine sehr große Winkelabdeckung realisiert werden.

Zusätzlich kann noch der gesamte Myonenfluss über Integration ermittelt werden.

3.2.4. GUI

Root erlaubt es einfach grafische Userinterfaces zu realisieren. Dieses ist notwendig, da das Programm auch von anderen genutzt werden soll. In Abbildung 21 sieht man das Formular des GUI.

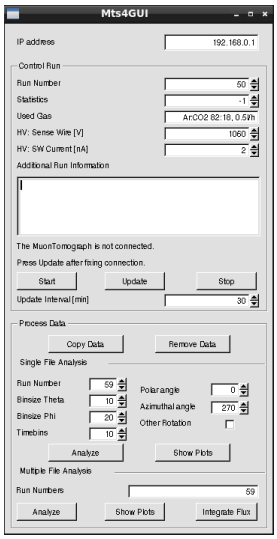


Abbildung 21: GUI

Im oberen Teil befinden sich die Eingabemöglichkeiten für die Interaktion mit dem Myonenteleskop. Man kann Messungen starten, beenden, sowie kommentieren. Die Kommunikation mit dem Teleskop erfolgt durch die in Tabelle 1 aufgelisteten Befehle.

Im mittleren Teil findet der Nutzer alle Einstellungsmöglichkeiten für die Offlineanalyse der einzelnen Runs. Dabei kann die Pre- und Hauptanalyse separat von der Polaranalyse und der Histogrammaddition ausgeführt werden. So spart man deutlich Zeit, da die weitere Analyse sehr viel schneller vonstatten geht.

Im unteren Teil findet man die Eingabemöglichkeiten für die Analyse mehrerer Runs. Die einzelnen Runs können zusammengefasst werden und es kann der gesamte Fluss durch Integration bestimmt werden. Dabei können maximal 10 Runs gleichzeitig analysiert werden.

3.2.5. Dokumentation

Damit das Programm nachträglich und auch von anderen Personen verstanden werden kann, ist es nötig eine übersichtliche Dokumentation zu erstellen. Dazu verwendeten wir das Programm Doxygen. Dieses durchsucht den Quellcode automatisch nach Klassen und ihren Methoden und Variablen und erstellt daraus eine Dokumentation, welche sowohl aus Latex als auch als HTML-Dateien bestehen kann.
Die Dokumentation besteht aus einem Verzeichnis der Quellcode-Dateien und Klassen. Alle Elemente sind miteinander verlinkt. Es gibt ein Suchfenster, sodass man schnell alle benötigten Informationen findet. Des Weiteren werden die Abhängigkeiten der Klassen, Methoden und Dateien grafisch dargestellt.

4. Messungen

Im Folgenden geht es um die Messungen, die mit dem Teleskop durchgeführt wurden sowie ihre eigentliche Auswertung.

4.1. Grundsätzliche Vorgehensweise

Ziel der Messungen ist es einen möglichst großen Winkel abzudecken. Da das Teleskop allerdings nur ungefähr mit einem Öffnungswinkel von rund 90° gut abdeckt, werden für die Flussbestimmung an einem Standort Messungen in verschiedene Richtungen gemacht: eine horizontal und jeweils vier Messungen im Winkel von 45° und 90° zur Senkrechten in alle Richtungen. Da das Myonenteleskop nicht den Richtungssinn der Myonen unterscheiden kann, braucht man für die 90° Messungen tatsächlich nur zwei Messungen, die anderen wären äquivalent. Somit besteht ein vollständiger Messsatz aus sieben Runs.

Zum Messen stellt man das Teleskop in die gewünschte Position, schließt die Gasversorgung an. Um Stromschläge zu vermeiden, muss eingedrungene Luft durch das Gas aus dem Teleskop gespült werden (typischerweise 1 Tag). Danach kann die Messung gestartet werden, indem man die Spannung einstellt, mit der das Teleskop betrieben werden soll, und den Raspberry Pi, der mit dem PC verbunden wurde, startet.

Die Messungen sollten ausreichend lange betrieben werden, um eine gute Statistik zu gewährleisten. Die Messung wird nach entsprechender Zeit durch erneutes Verbinden mit dem Teleskop beendet und die Daten können entnommen werden.

4.2. Oberirdische Messungen

Die oberirdischen Messungen dienten als Referenzmessungen, um die Funktionsweise des Teleskops zu überprüfen. Es wurde dafür ein Messsatz in einem Raum in HZDR aufgenommen. Es wurde angenommen, dass die Abschirmung durch das Gebäude nur minimal ist.

Ziel war es, die Funktionsfähigkeit des Programms und des Teleskops zu testen. Erwartet wurde dabei ein $\cos^2$ verteilter Flux. Allerdings zeigte sich bei dieser Messung besonders deutlich das in Abschnitt 4.4.2 angesprochene Problem der Akzeptanzberechnung. Außerdem schirmte das Gebäude den Myonenfluss stärker ab als erwartet, weshalb eine zweite oberirdische Messung direkt unter dem Dach des Gebäudes durchgeführt wurde. Hier konnte auch die $\cos^2$-Abhängigkeit des Myonenflusses gezeigt werden.

4.3. Messungen im Felsenkeller

Nachdem die oberirdischen Vergleichsmessungen beendet waren, wurde mit den unterirdischen begonnen.

Um das Teleskop vor Spritzwasser zusätzlich zu schützen, wurde eine Kunststofffolie über dieses gestülpt.

Da der Myonenfluss unterirdisch geringer ist als oberhalb, muss ausreichend lange gemessen werden. Aus diesem Grund konnten immer nur zwei Messungen pro Woche durchgeführt werden, sodass eine Messung drei bis vier Tage dauerte und ein Messsatz (sieben Messungen) in etwas unter einem Monat vollständig war.

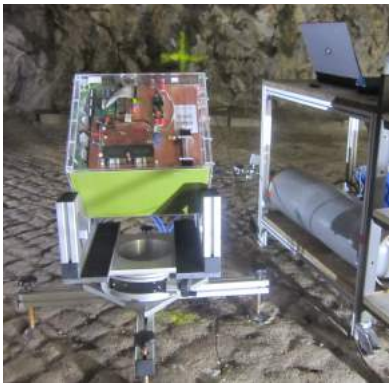


Abbildung 22: Myonentelesekop im Felsenkeller

Es wurde an vier Positionen (MP1-MP4) in den Stollen, in die später der Teilchendetektor eingebaut werden soll, gemessen, diese sind in Abbildung 23 eingezeichnet. Zusätzlich dazu wurden vier Messsätze in dem benachbarten Stollen, in welchem sich das VKTA-Labor befindet, gemessen.

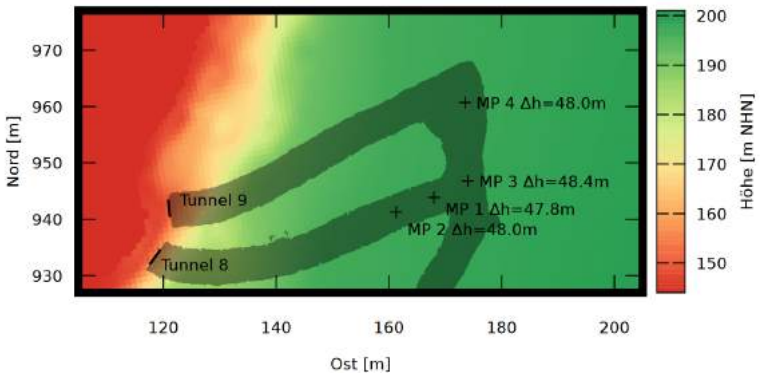


Abbildung 23: Lageplan mit Kennzeichnung der Locations im Felsenkeller

Um später Aussagen über den Myonenfluss treffen zu können, muss neben der Position des Teleskops auch dessen Ausrichtung bekannt sein. Untertage ist dies auf Grund fehlender Orientierung an markanten Punkten oder der Sonne schwierig, sodass auf eine Winkelbestimmung mittels Kompass zurückgegriffen wurde. Diese Messung kann allerdings fehlerbehaftet sein, da das Magnetfeld der Erde von lokalen Magnetfeldern magnetischer Gesteine gestört werden kann. Deshalb ist die Winkelbestimmung nur auf 10° genau.

4.4. Auswertung

Die Messungen liefern viele verschiedene Erkenntnisse über das Teleskop und den Myonenfluss. Im folgenden geht es um die Auswertung dieser.

4.4.1. Channelhits

Eine wichtige Statistik, die Aufschluss über den korrekten Betrieb des Teleskops liefert sind die Channelhits. In Abbildung 24 sieht man eine aktuelle Statistik unseres Teleskops dazu.

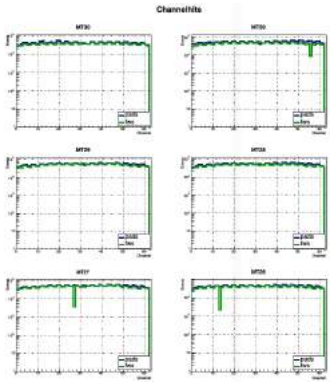


Abbildung 24: Beispiel für aktuelle Channelhits unseres Teleskops

Es handelt sich dabei um ein Histogramm, in welchem die Anzahl der Treffer für jeden Channel aufgetragen ist. Wenn ein Channel weniger oft getroffen ist, ist dies ein Anzeichen dafür, dass er nicht ordnungsgemäß funktioniert. Wie man sieht, trifft dies bei unserem Teleskop für nur drei Fieldwires zu, was nicht weiter problematisch ist, da immer mehrere Channel auf einmal getroffen werden und so Myonen, die diesen Bereich des Chambers ionisieren trotzdem registriert werden. Allerdings mussten wir feststellen, dass bei unseren ersten Messungen im Felsenkeller viele defekt waren, sodass das Teleskop zunächst repariert werden musste.

4.4.2. Akzeptanz

Bei der Addition mehrerer Runs verschiedener Winkel zeigte sich, dass der Fluss untereinander stark abweicht, außerdem konnte eine $\cos^2$-Abhänigkeit nicht bestätigt werden. In Abbildung 25 sieht man den Flux verschiedener Messungen abhängig von Theta bei festem Phi.

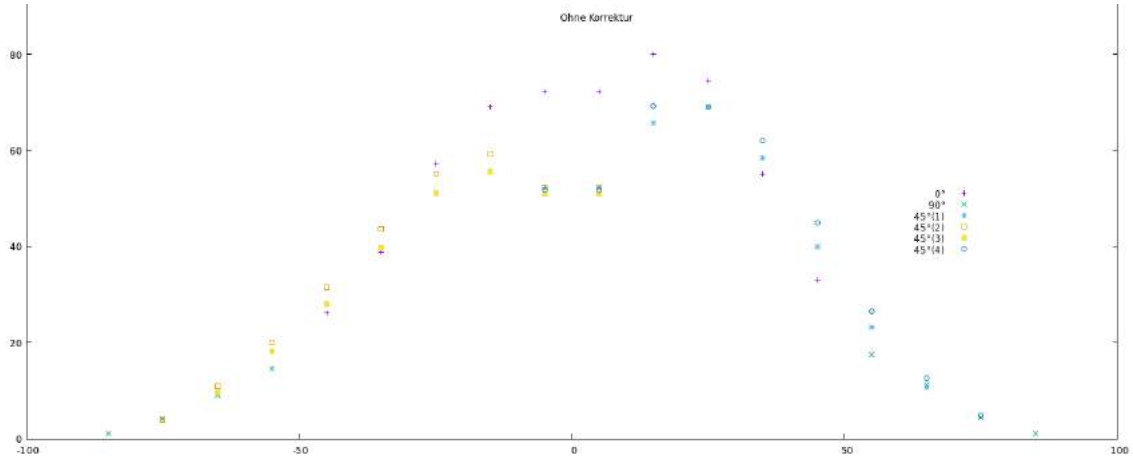


Abbildung 25: Fluss in Abhängigkeit von ϑ ohne Korrektur

Man sieht deutlich, dass die Messpunkte verschiedener Messungen nicht miteinander übereinstimmen. Messpunkte, bei denen die Myonen unter einem steilen Winkel auf den Detektor treffen sind größer, als diejenigen, die unter flachem Winkel einfallen, obwohl

beide gleich groß sein sollten. Dieser Fehler scheint an der Akzeptanzberechnung zu liegen. Bei dieser Berechnung wurde bisher nur die Fläche berücksichtigt, die Myonen dieses Winkels detektieren kann, trotzdem wird die Akzeptanz der unter kleinem Winkel Theta ankommenden Myonen zu hoch eingeschätzt.

Es zeigte sich, dass die Akzeptanz, zusätzlich zum bisherigen Faktor, vom Winkel der eintreffenden Myonen abhängt, genauer gesagt vom Cosinus des Einfallwinkels.

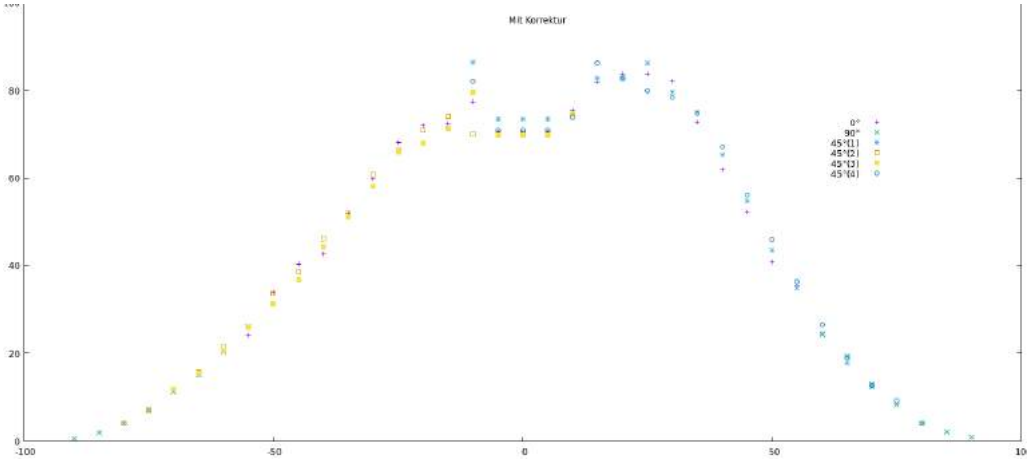


Abbildung 26: Fluss in Abhängigkeit von ϑ mit Korrektur

In Abbildung 26 sieht man nun die deutlich bessere Übereinstimmung der Messpunkte untereinander. Als Qualitätskriterium kann man somit für einen Messsatz die Abweichung der Messwerte vom Mittelwert dividiert durch den Fehler des Mittelwerts berechnen. Das Ergebnis ist in Abbildung 27 zu sehen.

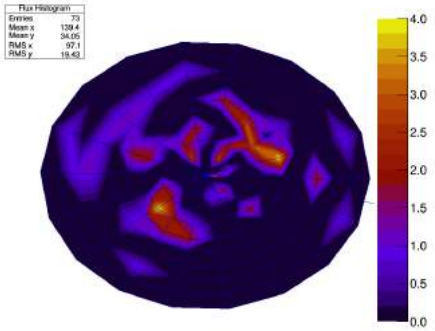


Abbildung 27: Abweichung der Messwerte vom Mittelwert (Messsatz: VKTA mk2)

Dabei ist zu beachten, dass die Punkte nicht alle auf dem Mittelwert liegen müssen, sondern dass sie mit einer gaußverteilten Wahrscheinlichkeit davon abweichen können. Daraus folgt, dass ein Histogramm, das mit den Werten aus der Abweichungsberechnung gefüllt wird, eine Gaußverteilung mit einem Mittelwert von 0 und einer Standardabweichung von 1 zeigen sollte. In Abbildung 28 ist ein solches Histogramm zu sehen.

Man kann erkennen, dass der Mittelwert nahe 0 und die Standardabweichung nur

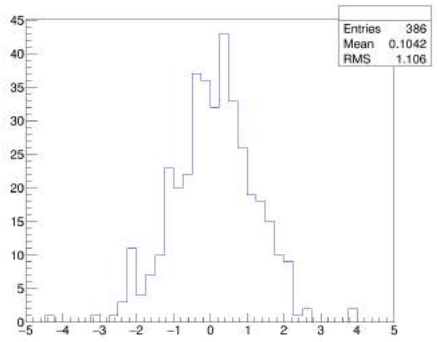


Abbildung 28: Histogramm zur Abweichung zum Mittelwert (Messsatz: VKTA mk2)

gering über 1 liegt. Dies zeigt, dass bei der Fehlerberechnung und Histogrammaddition kein Fehler gemacht wurde.

4.4.3. Flux

Zunächst sollten die oberirdischen Messungen eine $\cos^2$-Abhängigkeit bestätigen. Dazu sollte ursprünglich, wie bereits erwähnt, die Messung in Raum 008/620 (Keller) im HZ-DR dienen. Es stellte sich jedoch heraus, dass die Abschwächung des Flusses durch die Gebäudewände die $\cos^2$-Abhängigkeit stark überlagerte. Deshalb führten wir eine neue Messreihe im Raum 301/620 durch, dieser Raum liegt direkt unter der Decke und erfährt somit eine deutlich geringere Abschirmung.

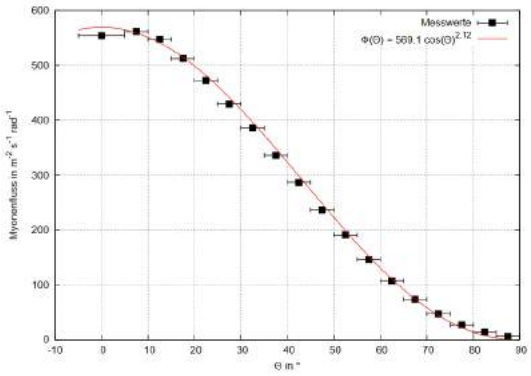


Abbildung 29: Diagramm zur Verdeutlichung der $\cos^2$-Abhängigkeit

In Abbildung 29 [15] ist die Abhängigkeit des Myonenflusses von ϑ dargestellt. Dazu wurde der Fluss der einzelnen Bins über φ integriert. Zu beachten ist dabei das φ=0° ausgelassen wurde, da in dieser Richtung die Abschirmung durch die Wand zu groß war und das Ergebnis beeinträchtigt hätte. Es wurde eine $\cos^{2,12}$-Abhängigkeit festgestellt; man erkennt ein gutes Übereinstimmen aller Messpunkte. Unsere Messungen decken sich damit mit den Literaturwerten.

[15] freundlicherweise von Felix Ludwig zur Verfügung gestellt

Im Vergleich von den oberirdischen zu den unterirdischen Messungen war beim gesamten Myonenfluss eine Abnahme um etwas unter zwei Größenordnungen zu erwarten (siehe Abbildung 2). In Tabelle 2 sieht man den integrierten Myonenfluss der verschiedenen Positionen.

Standort	Fluss in $m^{-2}s^{-1}$	Bemerkung
Raum 301/620	179,64 ± 0,15	unter Dach HZDR
Raum 008/620	173,45 ± 0,10	im Keller HZDR
Loacation 1	4,90 ± 0,22	im Felsenkeller
Loacation 2	5,34 ± 0,23	im Felsenkeller
Loacation 3	4,41 ± 0,23	im Felsenkeller
Loacation 4	5,02 ± 0,22	im Felsenkeller
VKTA-storage room	5,04 ± 0,22	im VKTA
VKTA-Werstatt	7,03 ± 0,18	im VKTA
VKTA-mk1	4,74 ± 0,21	im VKTA
VKTA-mk2	5,98 ± 0,22	im VKTA

Tabelle 2: Integrierter Myonenfluss an verschiedenen Locations

Die beiden oberirdischen Messungen weisen einen Gesamtfluss von über 170 $m^{-2}s^{-1}$ auf, während die unterirdischen Messungen nur einen Fluss von 4 bis 8 $m^{-2}s^{-1}$, je nach Lage, besitzen. Das Gestein schirmt also wie erwartet die Myonen ab. Zusätzlich sieht man bei den Messungen im Felsenkeller, dass sich der Abstand zur Felskante auf den Gesamtfluss auswirkt. Location 2 ist am nächsten an der Kante gelegen und weist den höchsten Fluss auf, während Location 3 am weitesten von dieser entfernt ist und einen entsprechnd geringeren Fluss besitzt.

Für die Winkelabhängigkeit des unterirdischen Myonenflusses ist zum einen die Winkelabhängigkeit der oberirdischen Myonen von Bedeutung, zum anderen ihre Wegstrecke durch das Gestein, welche sie bis zum Teleskop zurücklegen müssen. Entsprechend der $\cos^2$ Verteilung der oberirdischen Myonen treffen die meisten Myonen vertikal auf die Erde. Der Weg, den die Myonen zurücklegen müssen, ist einerseits vertikal besonders kurz, andererseits in Richtung Abbruchkante des Felsen, also Richtung Ausgang. Aus diesen beiden Komponenten lässt sich vermuten, dass die meisten Myonen aus vertikaler Richtung, bzw. aus Richtung des Ausgangs kommen.

Im Anhang ist der Myonenfluss an den einzelnen Positionen zu sehen. Grundsätzlich erkennt man bei allen unterirdischen Messungen in Richtung des Ausgangs einen höheren Myonenfluss, als in die entgegengesetzte Richtung. Auch weisen alle Messungen einen großen Myonenfluss unter einem kleinen Winkel ϑ auf, während er bei großem ϑ gegen 0 geht. Hauptsächlich habe ich mich mit den Messsätzen, die in den Tunneln des späteren Teilchenbeschleunigers gemacht wurden beschäftigt und werde sie nun detaillierter beschreiben, auch wenn viele grundsätzliche Überlegungen auf die Messungen im VKTA übertragbar sind.

Zunächst ist auffällig, dass der vertikale Myonenfluss bei den vier Locations sehr ähnlich ist. Der Myonenfluss bei ϑ gleich 0 °liegt zwischen 1,6 und 1,7 m^{-2} s^{-1} [16] sr^{-1} und

[16] Myonen pro Quadratmeter, Sekunde und Steradiant

weicht somit nur um 7 % voneinander ab. Auf Grund dieser Messungen ist anzunehmen, dass der vertikale Myonenfluss im untersuchten Gebiet relativ homogen ist, was erlaubt, den vertikalen Myonenfluss auch an Stellen, die nur in der Nähe der Messungen liegen, gut abzuschätzen. Außerdem kann man daraus schlussfolgern, dass die Abschirmung durch das Gestein etwa konstant ist, die Felsdecke also sehr homogen beschaffen ist, da der oberirdische Myonenfluss bei jeder Messung und Position konstant bleibt.

Weitere Gemeinsamkeiten zwischen den Locations lassen sich in der, dem Ausgang abgewandten Seite feststellen. Der Myonenfluss fällt dabei abhängig von ϑ vom vertikalen Wert bis auf 0. Dies liegt zum einen daran, dass der oberirdische Myonenfluss einem $\cos^2$-Verlauf folgt und, dass die Wegstrecke durch das Gestein mit größer werdendem ϑ immer länger wird und so mehr Myonen abgefangen werden, und zwar proportional zum Kosinus von ϑ. Wenn man nun näherungsweise annimmt, dass der Myonenfluss indirekt proportional zu einer gewissen Potenz dieser Länge ist, ist somit eine $\cos^b$-Abhängigkeit, mit einem b, das größer als 2 ist, zu erwarten.

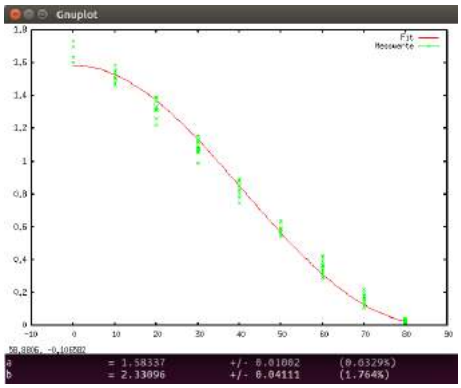


Abbildung 30: Mehrere Messkurven des Myonenflusses (in $m^{-2}\ s^{-1}\ sr^{-1}$), ohne Einfluss des Eingangs
Fitfunktion: $a \cdot \cos^b \theta$

In Abbildung 30 ist der Verlauf des Myonenflusses bei den vier Locations für jeweils drei verschiedene φ, die allerdings alle vom Ausgang wegzeigen, dargestellt.

Abgesehen von ϑ=0° passen die Messpunkte recht gut auf den Fit, der wie erwartet ein b größer als 2 aufweist. Trotz des Abweichens von einer Kosinusfunktion kann man festhalten, dass der Myonenfluss in Richtung Felsmassiv hauptsächlich von ϑ abhängt und nicht von φ oder von der genauen Position.

Die größten Unterschiede zwischen den Locations liegen im Fluss, der aus Richtung des Ausgangs kommt. Dabei unterscheidet sich die Stärke des Myonenflusses sowie die Richtung des maximalen Flusses[17] in Hinsicht auf φ und ϑ.

Die unterschiedliche Stärke des maximalen Flusses hängt in erster Linie von der Distanz zur Abbruchkante ab, deshalb ist er bei Location 3 deutlich geringer, als bei Location 2.

[17] im folgenden ist der maximale Fluss immer als Maximum aus Richtung Ausgang zu verstehen, also abgesehen von einem horizontalen Maximum

Quantitativ lässt sich kaum ein Zusammenhang finden, da nur vier Messungen gemacht wurden.

Weiterhin ist es wichtig die Abhängigkeit des maximalen Flusses in Hinsicht auf φ von der Location zu bestimmen. In Abbildung 31 wurde neben der Lage der Locations auch die Richtung des stärksten Myonenflusses von Richtung Ausgang eingezeichnet. Dabei ist zu beachten, dass die Winkelungenauigkeit des Teleskops etwa 10° auf Grund der Ausrichtung beträgt und dass die Bingröße in φ-Richtung 20° beträgt.

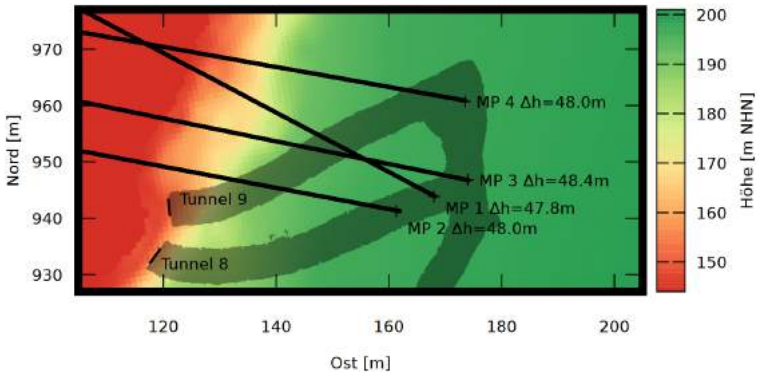


Abbildung 31: Lage der Location mit eingezeichneter Richtung des Maximums aus Richtung Abbruchkante

Man kann feststellen, dass die Hauptrichtung des Myonenflusses nicht der eigentliche, grau eingezeichnete Ausgang ist, sondern dass der größte Fluss aus Richtung der Abbruchkante erfolgt (links im Bild). Dies ist nicht verwunderlich, da der Ausgang eine Höhe von nur etwa 5 m aufweist und so nur für deutlich kleinere Winkel, unter denen sowieso kaum Myonen registriert werden, eine Bedeutung besitzt.

Nun ist noch die Höhe es maximalen Myonenflusses von Bedeutung. Man kann sagen, dass dieser Winkel ϑ unabhängig von der Distanz zur Abbruchkante ist. So weisen die Locations 2 und 3 den gleichen Winkel von ϑ=55° auf, obwohl sie den größten Unterschied in der Entfernung zum Ausgang haben. Unterschiede in dieser Höhe (Location 4) sind darauf zurückzuführen, dass die Abbruchkante nicht glatt ist und nicht geradlinig verläuft. Auch die beschränkte Auflösung des Teleskops und dessen Messfehler führen zu Abweichungen.

Durch die Messung und Auswertung des Myonenflussses an diesen vier Locations konnten vielfältige allgemeine Aussagen über den Myonenfluss im engeren Umfeld dieser Messungen getroffen werden. Der Myonenfluss ohne Einfluss des Eingangs scheint sehr homogen zu sein, während der Fluss in Richtung Abbruchkante sich unregelmäßiger verhält. Außerdem konnte festgestellt werden, dass der erhöhte Myonenfluss in Richtung „Ausgang“ in Wahrheit aus der Richtung der kürzesten Distanz zur Abbruchkante stammt. Diese Informationen sollten, neben den exakten Messungen dabei helfen, die notwendigen Abmessungen der aktiven Abschirmung zu bestimmen. Außerdem kann man sich für spätere Messungen darauf beschränken in Richtung Ausgang zu messen, da nur dort tatsächliche Unterschiede zu erwarten sind.

5. Fazit

Bei dieser Besonderen Lernleistung wurde ein Programm, aufbauend auf ein bereits existierendes, entwickelt, welches viele Verbesserungen bietet. Bei der Analyse können nun auch 5-Punkte-Tracks verwendet werden für die Flussberechnung, was zu einer Reduzierung statistischer Fehler führt. Außerdem wurde eine Polarauswertung hinzugefügt sowie die Zusammenfügung mehrerer Runs ermöglicht. Auch der Gesamtfluss kann nun berechnet werden. Durch eine gute Dokumentation und ein intuitives GUI sollte dieses Programm auch für Außenstehende nutzbar, bzw. erweiterbar sein. So ist es denkbar, dass das Teleskop nun einfacher für weitere Messungen benutzt werden kann.
Auf der anderen Seite konnte durch die Durchführung von 4 Messsätzen im Felsenkeller Dresden sowie deren Auswertung, der Myonenfluss sehr präzise für den Bereich des Standortes des geplanten Detektors bestimmt werden. Diese Informationen können später beim Bau der Versuchsanordnung berücksichtigt werden. So kann eine aktive Abschirmung beispielsweise auf bestimmte Bereiche beschränkt werden.

6. Danksagung

Ich möchte gerne allen denen danken, die mich bei meiner Arbeit unterstützt haben! Zunächst gilt mein Dank meiner schulischen Betreuerin Frau Damm, die sich die Zeit genommen hat mich zu betreuen und so diese BELL erst ermöglichte. Außerdem hat sie mir inhaltlich und sprachlich sehr geholfen. Desweiteren bedanke ich mich bei der TU Dresden und dem HZDR, welche mich als Institutionen unterstützten, indem sie mir das Thema anboten und mich praktisch an diesem Projekt arbeiten ließen. Dabei sei besonders Louis Wagner genannt, der mich als außerschulischer Betreuer unterstützte. Natürlich danke ich auch Felix Ludwig, welcher mit mir an diesem Projekt gearbeitet hat und dementsprechend die meisten fachlichen Fragen bezüglich des Teleskops beantworten konnte, für die gute Zusammenarbeit. Zu guter Letzt bedanke ich mich bei allen, die diese Arbeit Korrektur gelesen und so zur orthografischen und sprachlichen Richtigkeit selbiger beigetragen haben. Besonders sind dabei noch einmal Frau Damm, Louis Wagner sowie Linda Richter und meine Eltern genannt.

A. Übersicht der Runs

In den Tabellen 3, 4 und 5 sind alle Runs der Messungen mit ihren Rotationen und Standorten aufgeführt.

Standort	Runs	Winkel 1	Achse 1	Winkel 2	Achse 2
Location 1	33	90°	y	247°	z
	34	90°	y	337°	z
	35	0°	y	67°	z
	80	45°	y	337°	z
	81	45°	y	66°	z
	82	45°	y	157°	z
	83	45°	y	247°	z
Location 2	29	0°	y	61°	z
	31	90°	y	61°	z
	32	90°	y	151°	z
	84	45°	y	331°	z
	85	45°	y	151°	z
	86	45°	y	61°	z
	87	45°	y	241°	z
Location 3	36	90°	y	78°	z
	37	0°	y	78°	z
	38	90°	y	168°	z
	74	45°	y	258°	z
	75	45°	y	168°	z
	76	45°	y	48°	z
	77	45°	y	348°	z
Location 4	39	90°	y	355°	z
	40	90°	y	265°	z
	41	0°	y	190°	z
	69	45°	y	85°	z
	70	45°	y	175°	z
	71	45°	y	265°	z
	72	45°	y	355°	z
	73	0°	y	355°	z

Tabelle 3: Übersicht über die Runs im Felsenkeller

Standort	Runs	Winkel 1	Achse 1	Winkel 2	Achse 2
VKTA	88	90°	y	270°	z
storage room	89	90°	y	90°	z
	90	0°	y	270°	z
	91	45°	y	0°	z
	92	45°	y	180°	z
	93	45°	y	90°	z
	94	45°	y	270°	z
VKTA	95	0°	y	131°	z
mk2	96	90°	y	131°	z
	97	90°	y	221°	z
	98	45°	y	131°	z
	99	45°	y	311°	z
	100	45°	y	221°	z
	101	45°	y	41°	z
VKTA	102	90°	y	95°	z
mk1	103	90°	y	5°	z
	104	45°	y	95°	z
	105	45°	y	5°	z
	106	45°	y	275°	z
	107	45°	y	185°	z
	108	0°	y	95°	z
VKTA	109	90°	y	76°	z
Werkstatt	110	90°	y	346°	z
	111	45°	y	256°	z
	112	45°	y	76°	z
	113	45°	y	346°	z
	114	45°	y	116°	z
	115	0°	y	76°	z

Tabelle 4: Übersicht über die Runs im VKTA

Standort	Runs	Winkel 1	Achse 1	Winkel 2	Achse 2
Raum 008/620	53	90°	x	0°	z
	54	90°	x	90°	z
	55	45°	y	180°	z
	56	45°	y	270°	z
	57	45°	y	0°	z
	58	45°	y	90°	z
	59	0°	y	270°	z
Raum 301/620	116	0°	y	0°	z
	117	45°	y	90°	z
	118	45°	y	270°	z
	119	90°	y	180°	z
	120	45°	y	0°	z
	121	90°	y	90°	z
	122	45°	y	180°	z

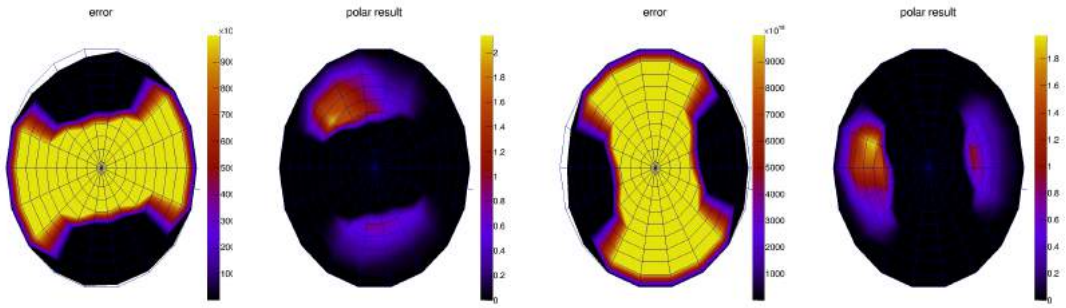
Tabelle 5: Übersicht über die oberirdischen Runs

B. Ausgewählte Messwerte

B.1. Runs

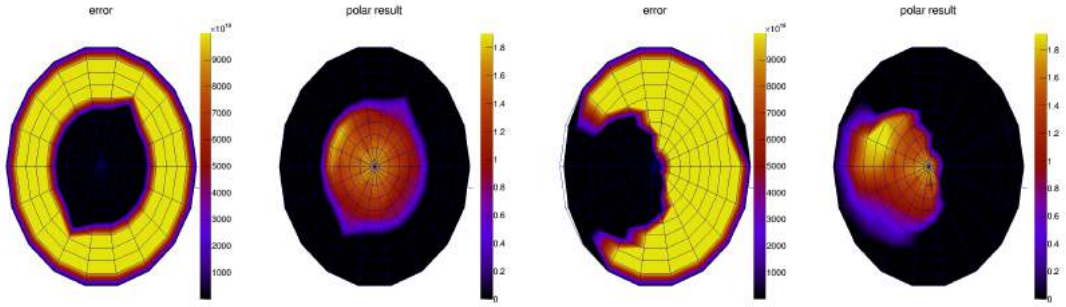
Im folgenden sind alle Polarplots der einzelnen Runs, nach den zugehörigen Locations geordnet, zu sehen. Sie zeigen aud der linken Seite den relativen Fehler, der die Gültigkeit der Messung angibt und auf der Rechtem den Fluss in m $^{-2}$ s $^{-1}$ sr $^{-1}$.

B.1.1. Location 1



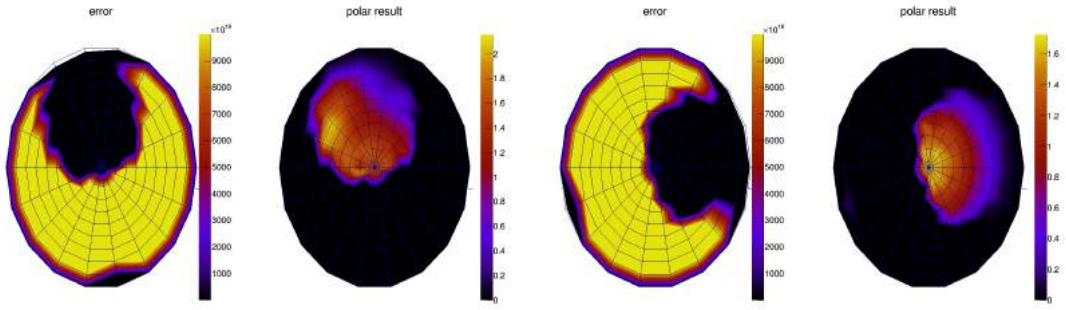
(a) Run 33 (b) Run 34

Abbildung 32: einzelne Runs



(a) Run 35 (b) Run 80

Abbildung 33: einzelne Runs



(a) Run 81 (b) Run 82

Abbildung 34: einzelne Runs

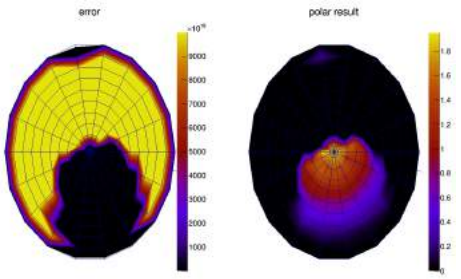
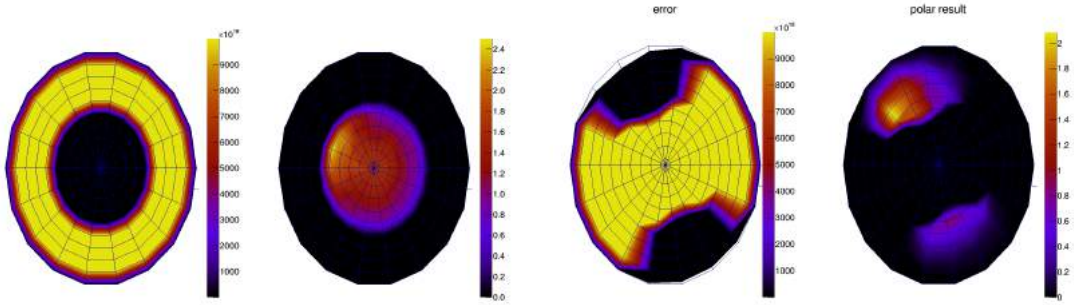


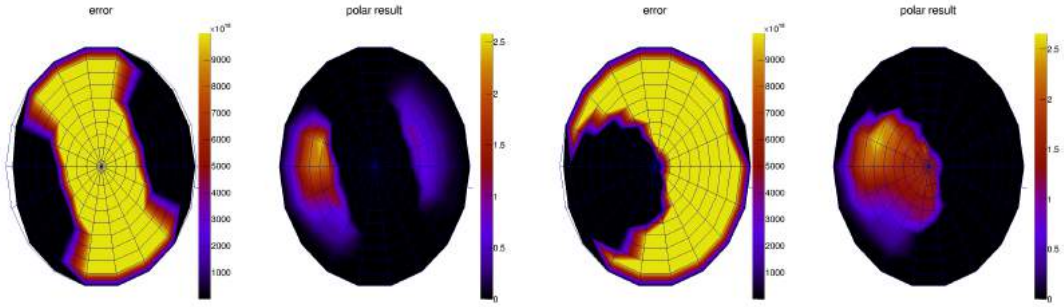
Abbildung 35: Run 83

B.1.2. Location 2



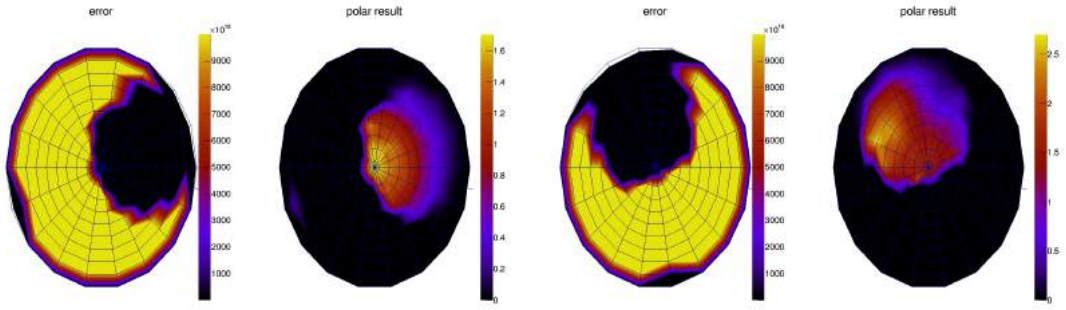
(a) Run 29 (b) Run 31

Abbildung 36: einzelne Runs



(a) Run 32 (b) Run 84

Abbildung 37: einzelne Runs



(a) Run 85 (b) Run 86

Abbildung 38: einzelne Runs

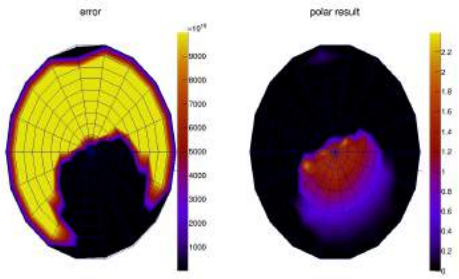
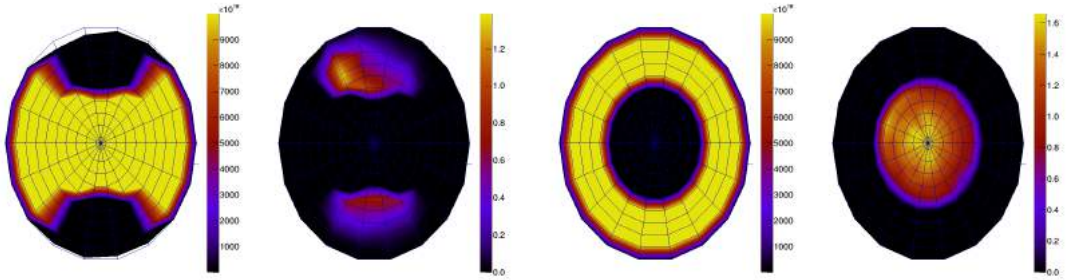


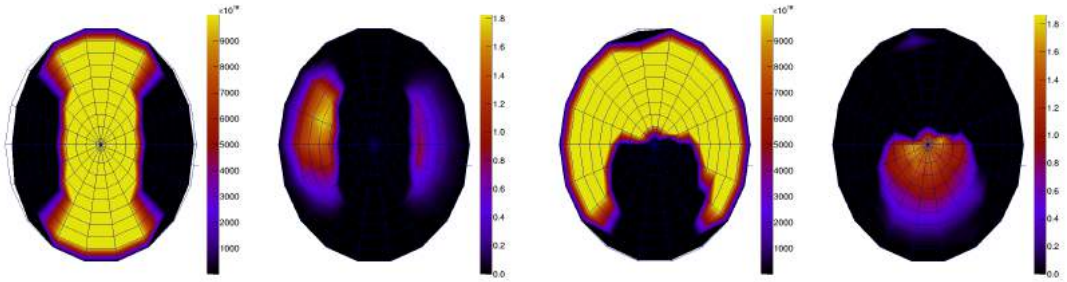
Abbildung 39: Run 87

B.1.3. Location 3



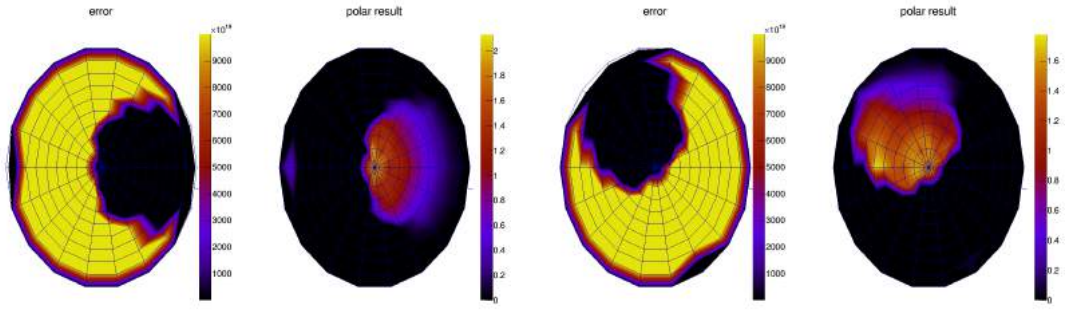
(a) Run 36 (b) Run 37

Abbildung 40: einzelne Runs



(a) Run 38 (b) Run 74

Abbildung 41: einzelne Runs



(a) Run 75 (b) Run 76

Abbildung 42: einzelne Runs

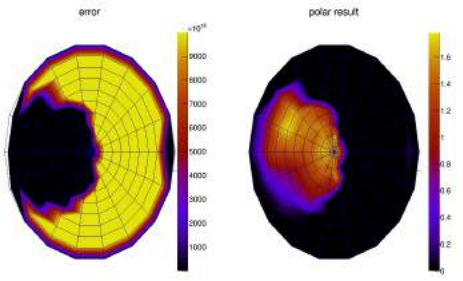
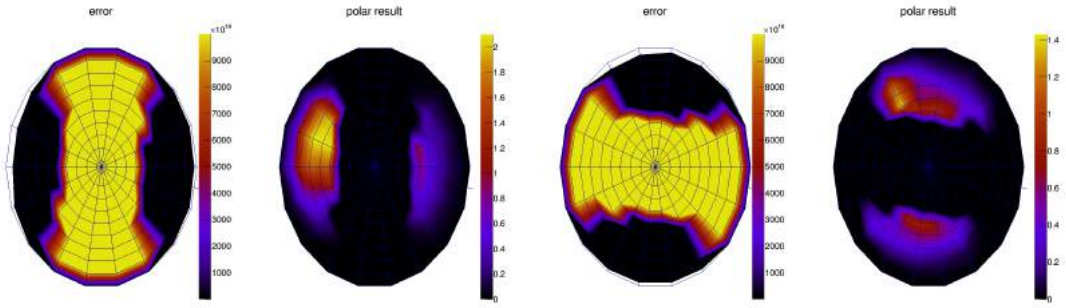


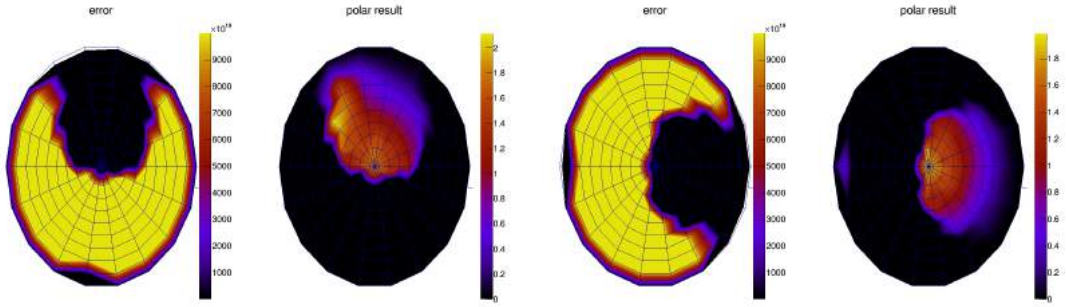
Abbildung 43: Run 77

B.1.4. Location 4



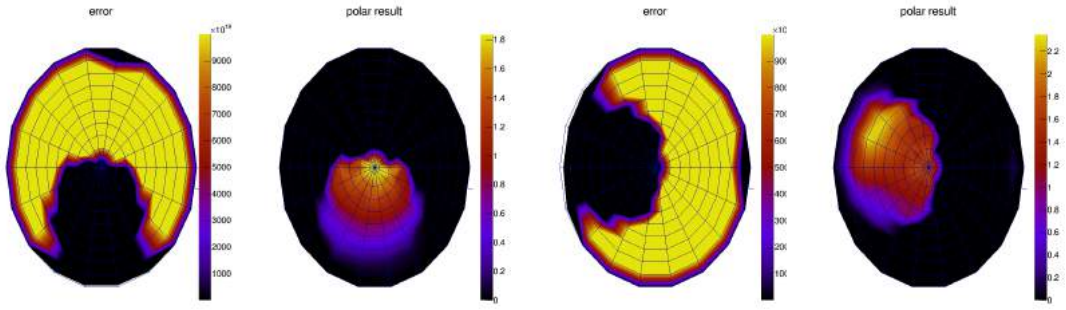
(a) Run 39 (b) Run 40

Abbildung 44: einzelne Runs



(a) Run 69 (b) Run 70

Abbildung 45: einzelne Runs



(a) Run 71 (b) Run 72

Abbildung 46: einzelne Runs

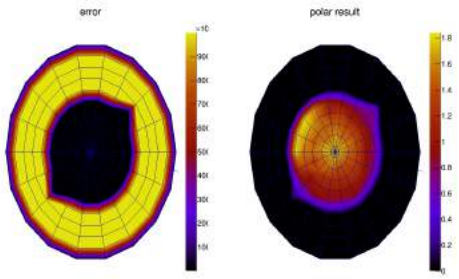
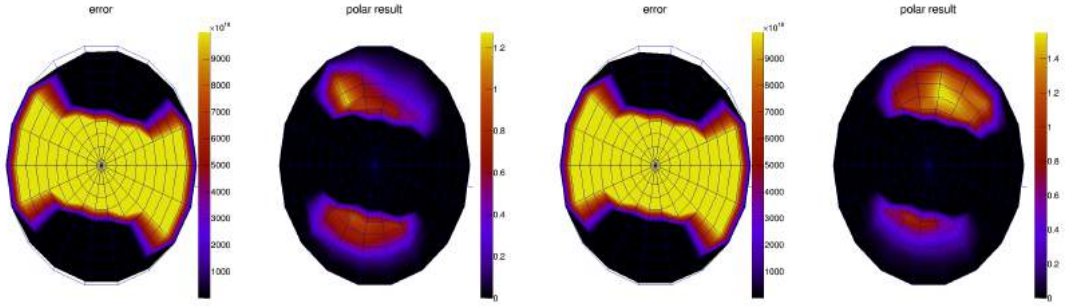


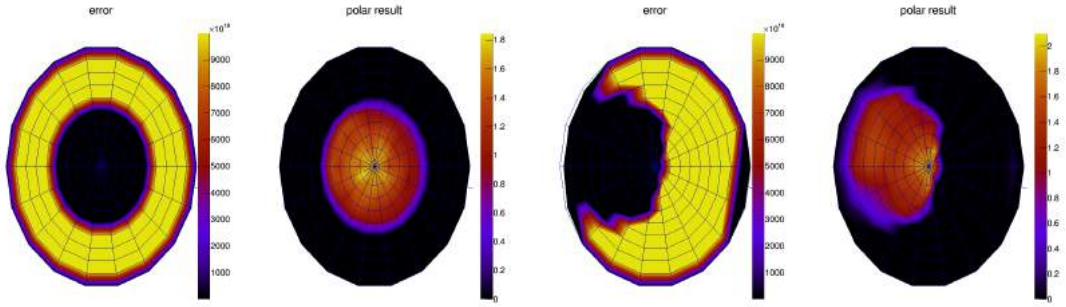
Abbildung 47: Run 73

B.1.5. VKTA storage room



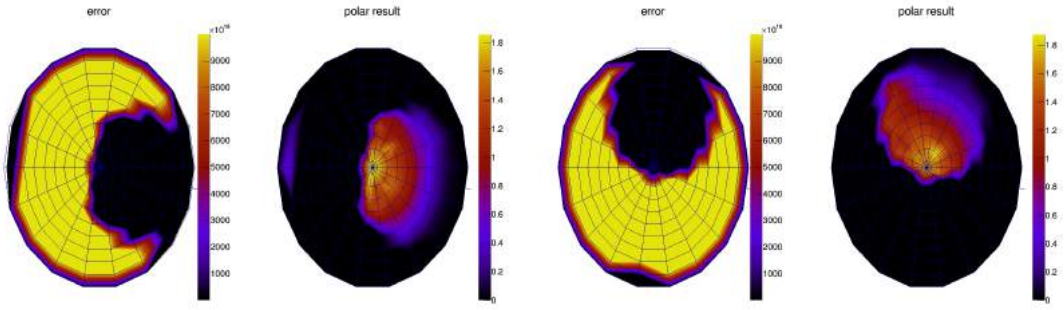
(a) Run 88 (b) Run 89

Abbildung 48: einzelne Runs



(a) Run 90 (b) Run 91

Abbildung 49: einzelne Runs



(a) Run 92 (b) Run 93

Abbildung 50: einzelne Runs

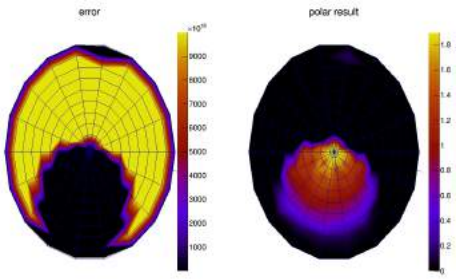
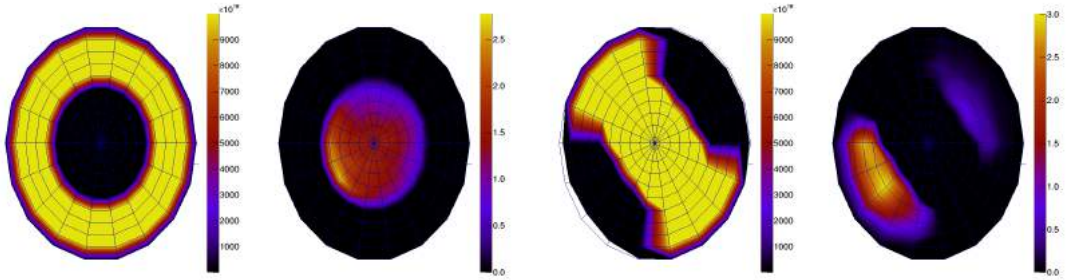


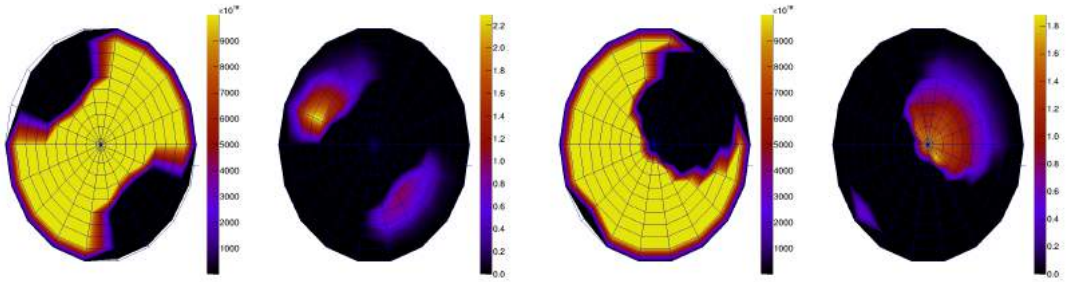
Abbildung 51: Run 94

B.1.6. VKTA mk2



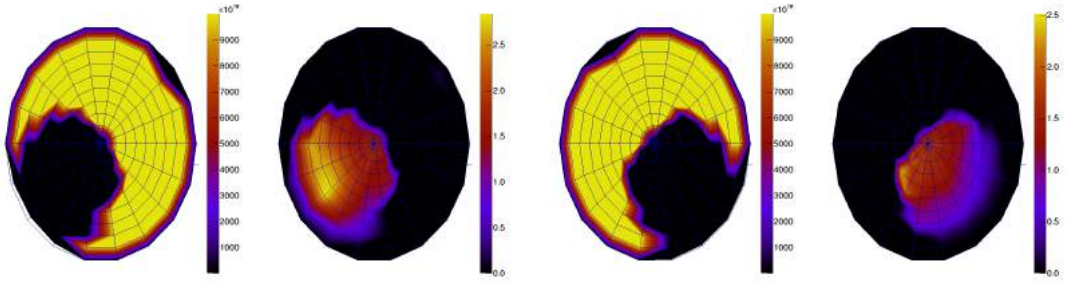
(a) Run 95 (b) Run 96

Abbildung 52: einzelne Runs



(a) Run 97 (b) Run 98

Abbildung 53: einzelne Runs



(a) Run 99 (b) Run 100

Abbildung 54: einzelne Runs

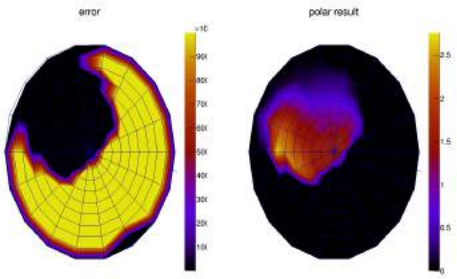
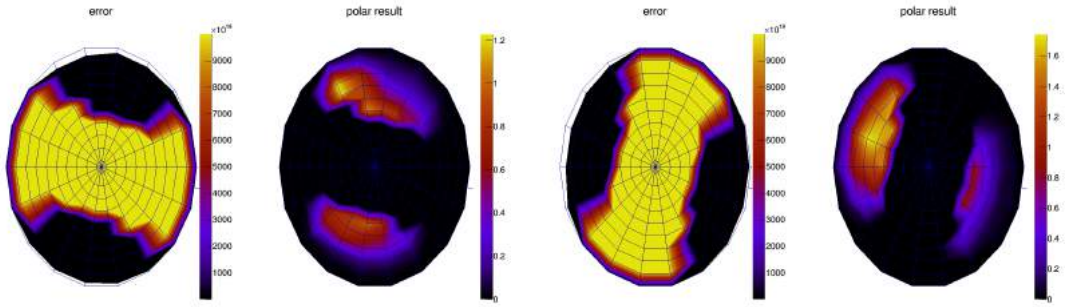


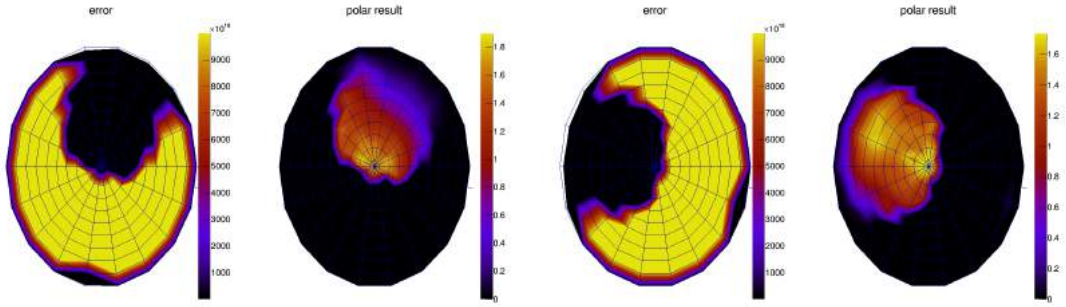
Abbildung 55: Run 101

B.1.7. VKTA mk1



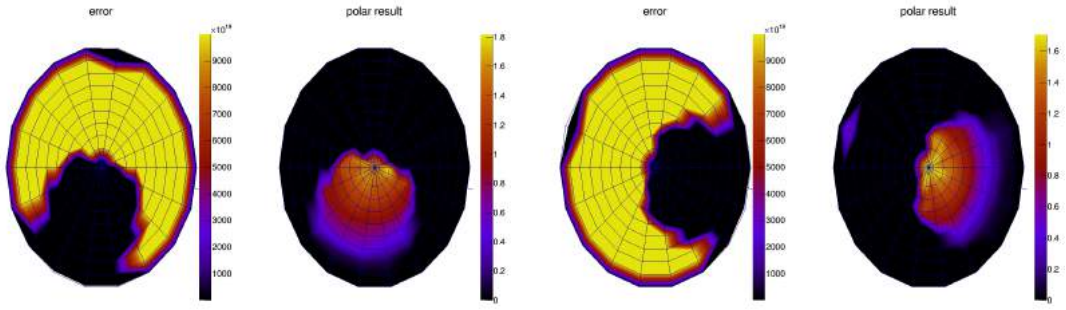
(a) Run 102 (b) Run 103

Abbildung 56: einzelne Runs



(a) Run 104 (b) Run 105

Abbildung 57: einzelne Runs



(a) Run 106 (b) Run 107

Abbildung 58: einzelne Runs

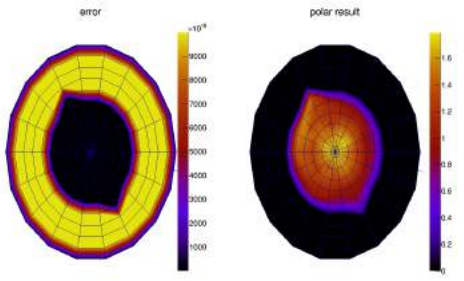
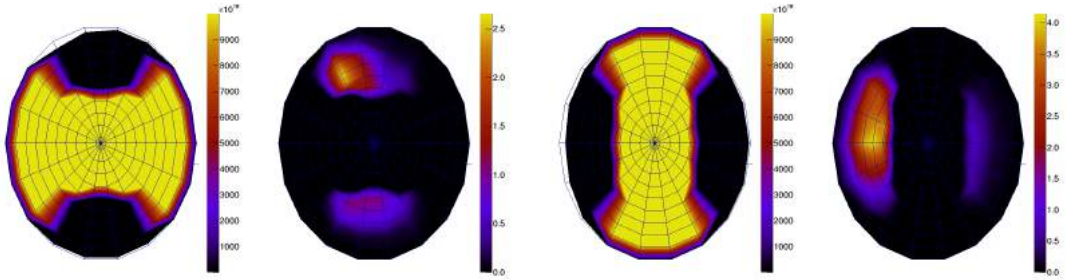


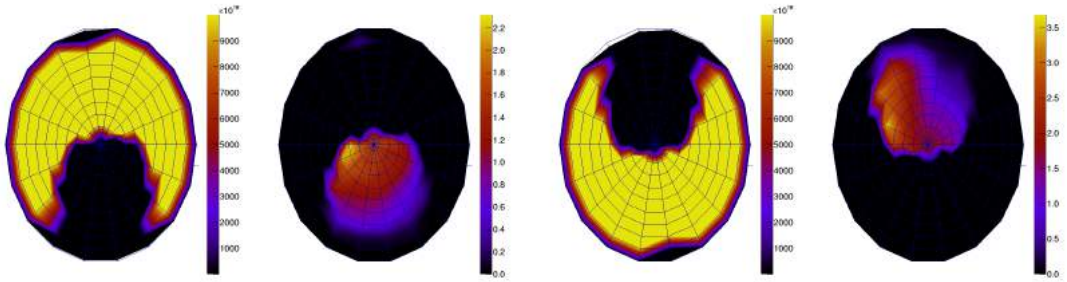
Abbildung 59: Run 108

B.1.8. VKTA Werkstatt



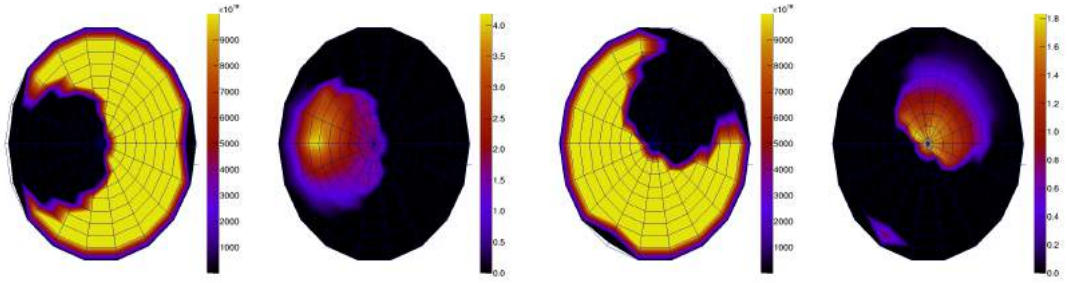
(a) Run 109 (b) Run 110

Abbildung 60: einzelne Runs



(a) Run 111 (b) Run 112

Abbildung 61: einzelne Runs



(a) Run 113 (b) Run 114

Abbildung 62: einzelne Runs

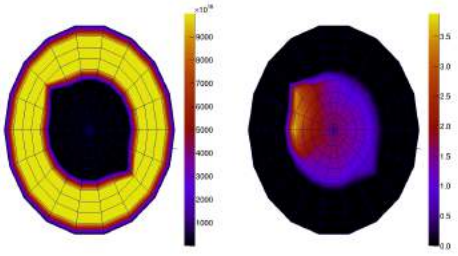
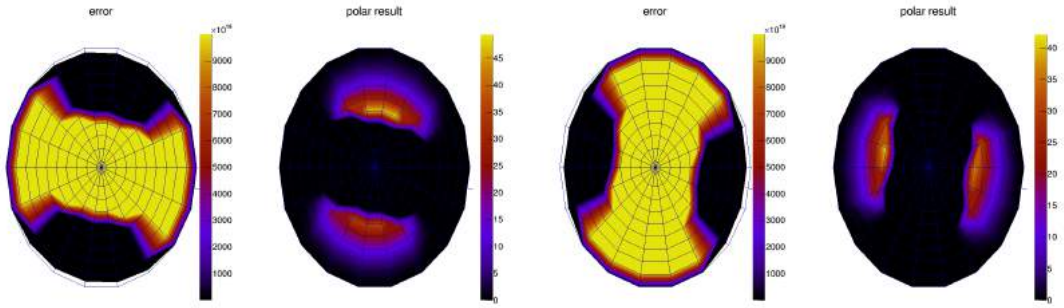


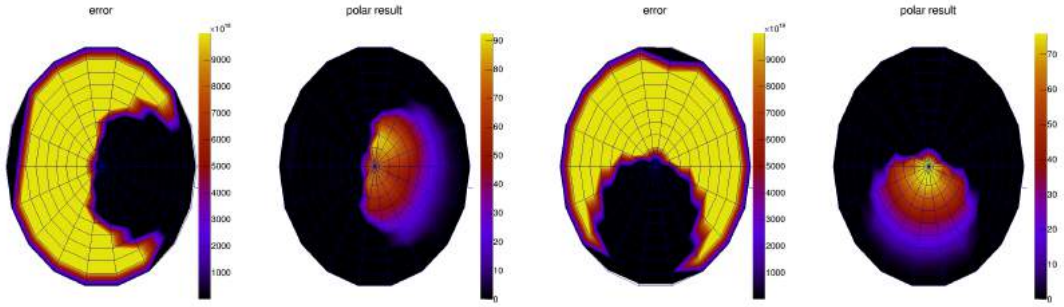
Abbildung 63: Run 115

B.1.9. Raum 008/620



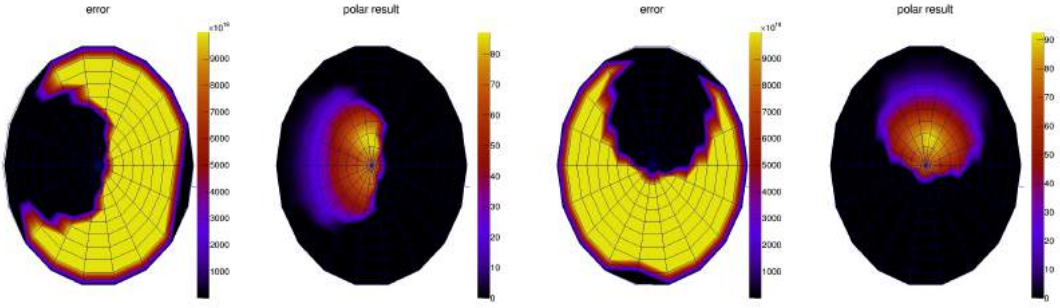
(a) Run 53 (b) Run 54

Abbildung 64: einzelne Runs



(a) Run 55 (b) Run 56

Abbildung 65: einzelne Runs



(a) Run 57 (b) Run 58

Abbildung 66: einzelne Runs

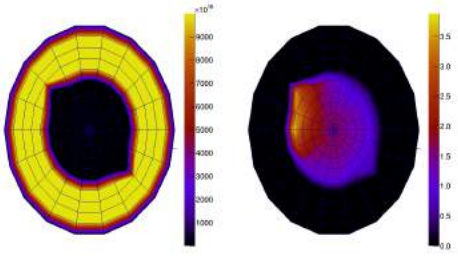
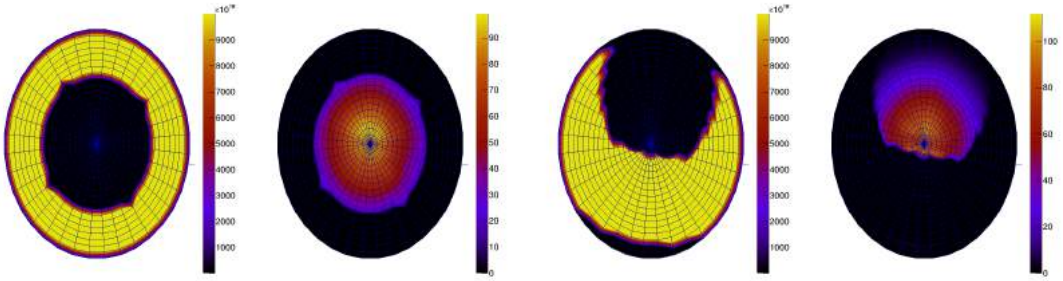


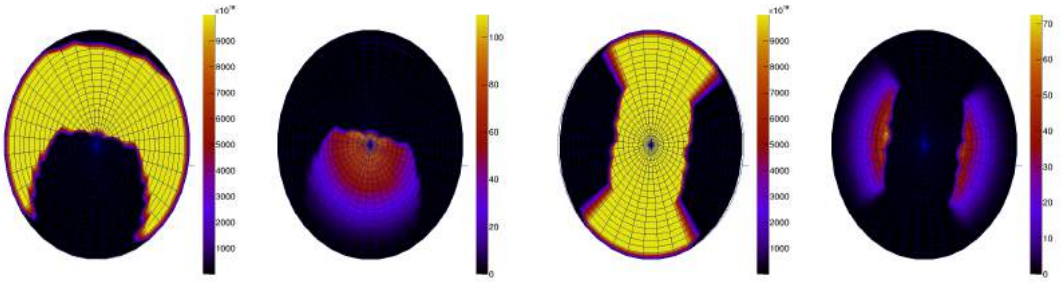
Abbildung 67: Run 59

B.1.10. Raum 301/620



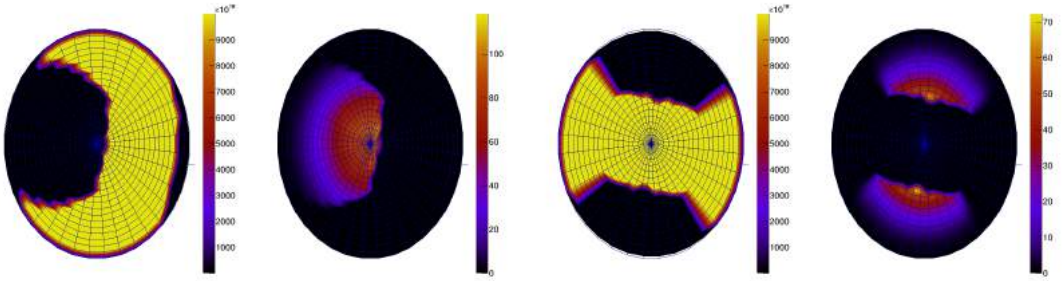
(a) Run 116 (b) Run 117

Abbildung 68: einzelne Runs



(a) Run 118 (b) Run 119

Abbildung 69: einzelne Runs



(a) Run 120 (b) Run 121

Abbildung 70: einzelne Runs

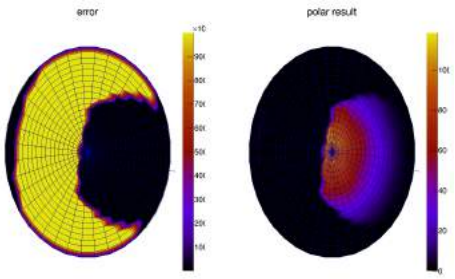


Abbildung 71: Run 122

B.2. Locations

In diesem Bereich sind alle Locations zu finden. Dies beinhaltet den Myonenfluss und den relativen Fehler des Myonenflusses.

B.2.1. Location 1

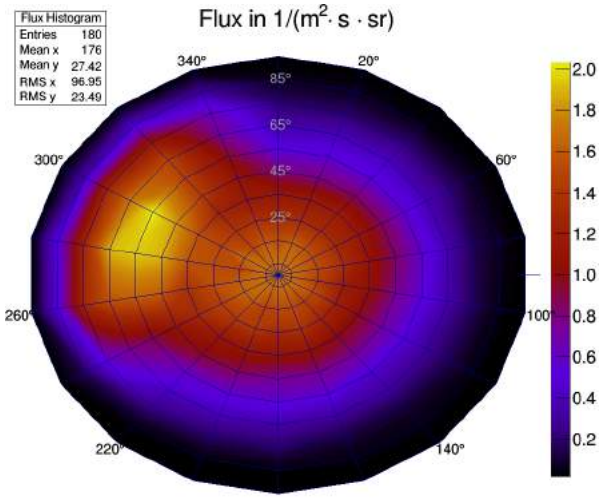


Abbildung 72: Fluss

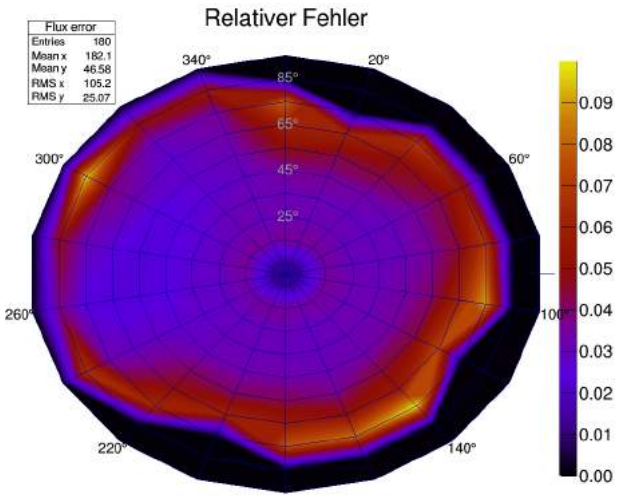


Abbildung 73: relativer Fehler

B.2.2. Location 2

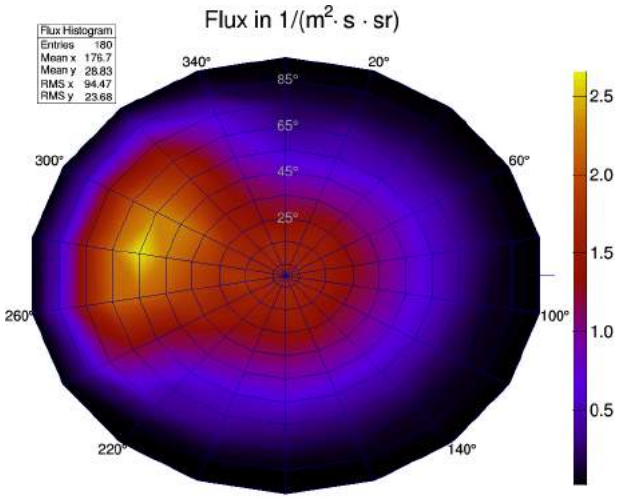


Abbildung 74: Fluss

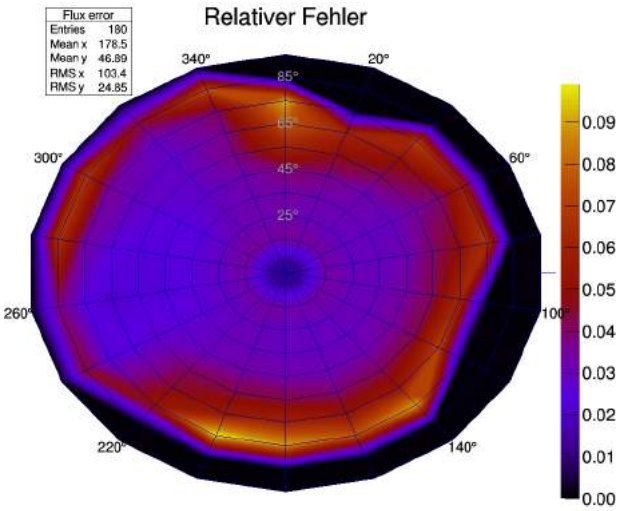


Abbildung 75: relativer Fehler

B.2.3. Location 3

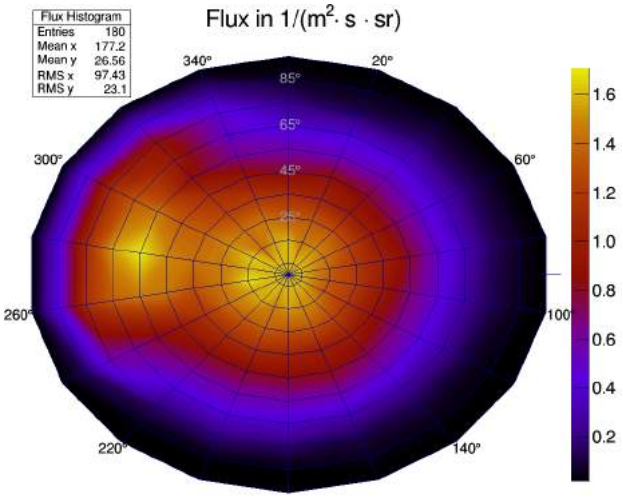


Abbildung 76: Fluss

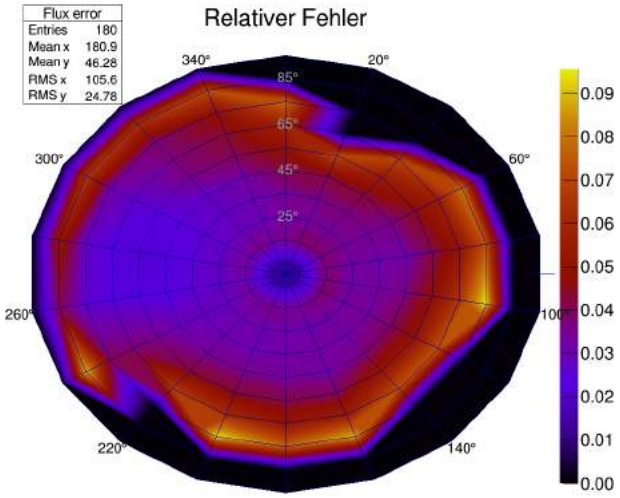


Abbildung 77: relativer Fehler

B.2.4. Location 4

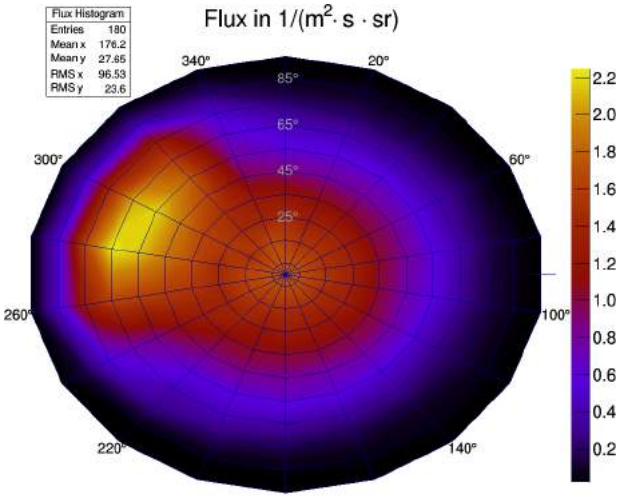


Abbildung 78: Fluss

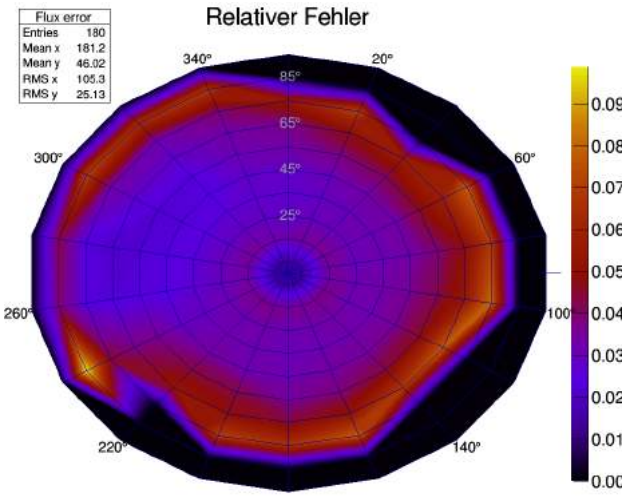


Abbildung 79: relativer Fehler

B.2.5. VKTA storage room

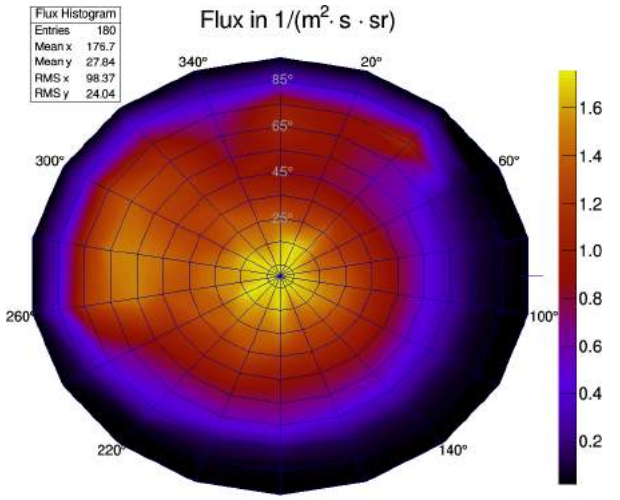


Abbildung 80: Fluss

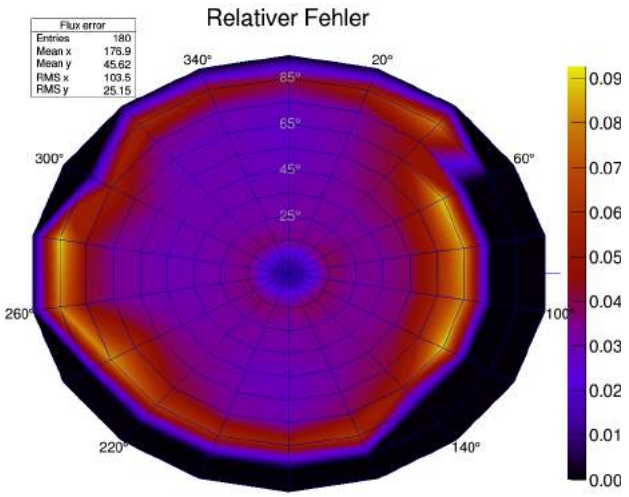


Abbildung 81: relativer Fehler

B.2.6. VKTA mk2

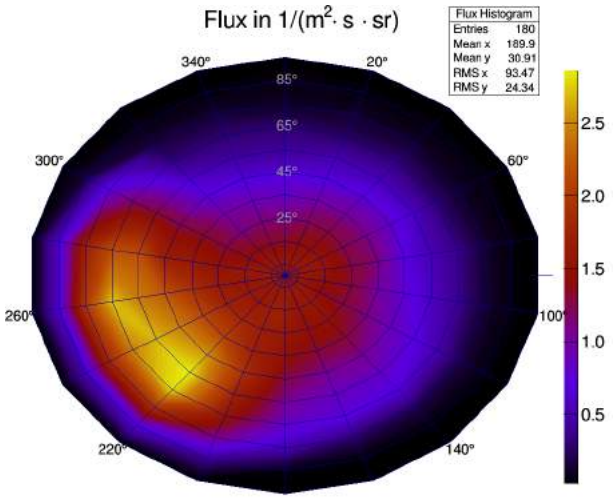


Abbildung 82: Fluss

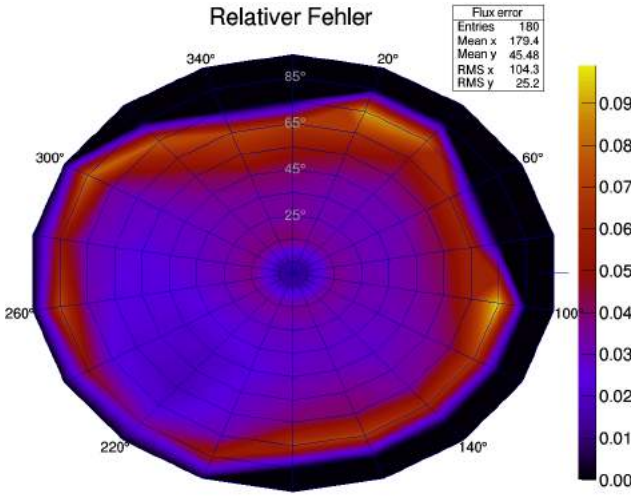


Abbildung 83: relativer Fehler

B.2.7. VKTA mk1

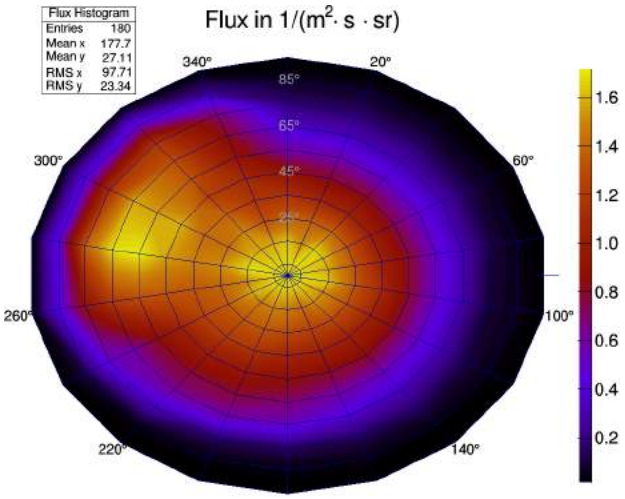


Abbildung 84: Fluss

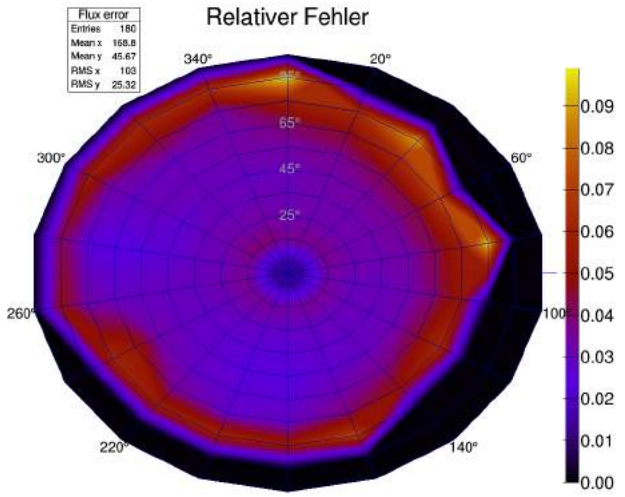


Abbildung 85: relativer Fehler

B.2.8. VKTA Werkstatt

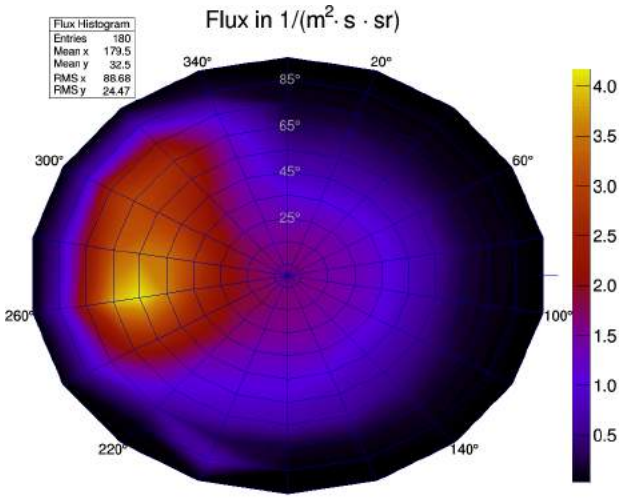


Abbildung 86: Fluss

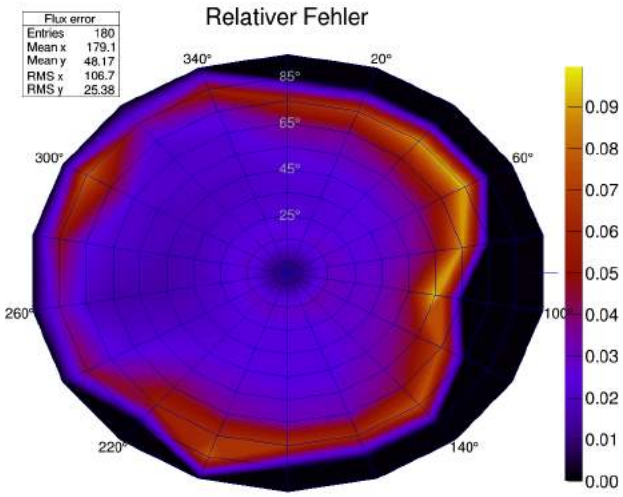


Abbildung 87: relativer Fehler

B.2.9. Raum 008/620

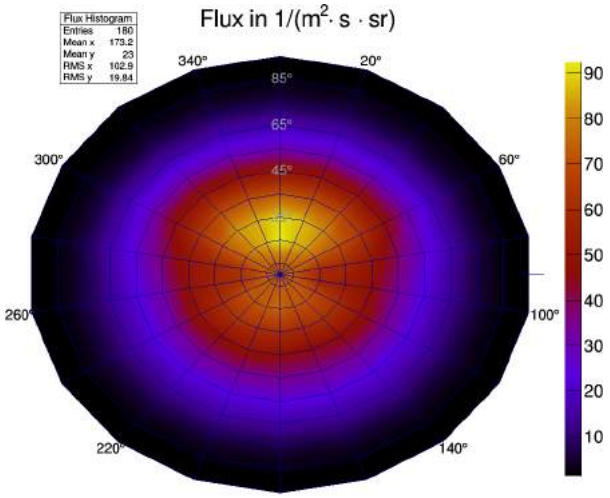


Abbildung 88: Fluss

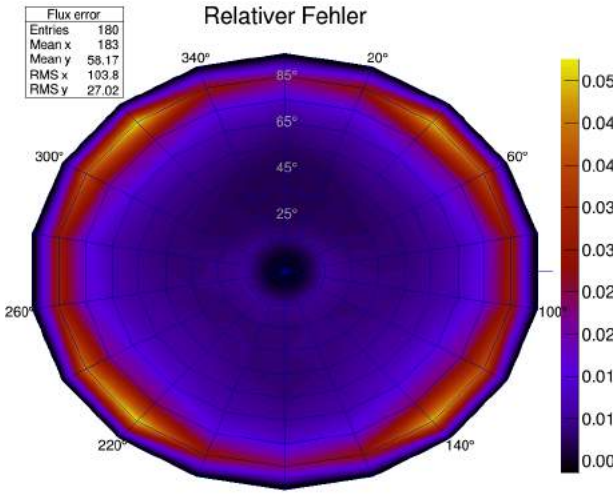


Abbildung 89: relativer Fehler

B.2.10. Raum 301/620

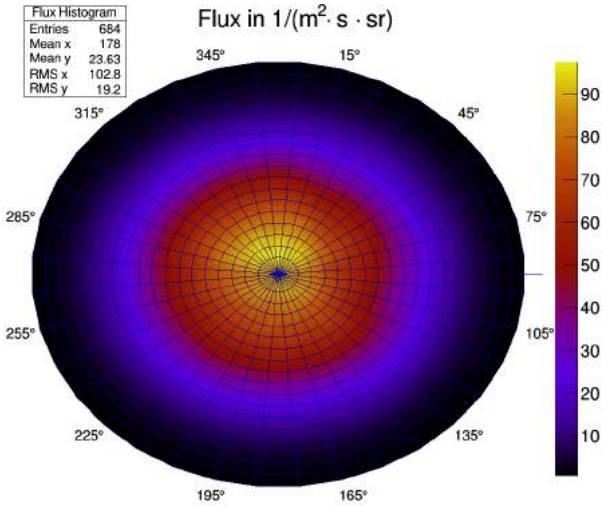


Abbildung 90: Fluss

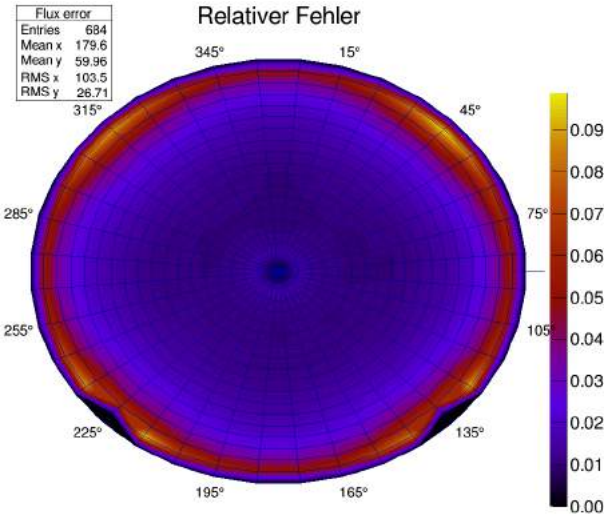


Abbildung 91: relativer Fehler

C. Quellcode

Im folgenden ist der gesammte Quellcode des Programm zu finden[18]. Es handelt sich um zehn Dateien.

C.1. mtparameters_5ch.h

```
1  ////////////////////////////////////////////
2  //Analysis code for the REGARD Mts4 detector//
3  ////////////////////////////////////////////
4
5
6  #ifndef Class_mtparam_5ch_Guard
7  #define Class_mtparam_5ch_Guard
8
9  #include <TMath.h>
10
11 ////////////////////////////////////
12 //MT30-MT39-MT29-MT28-MT27-MT26 setup//
13 ////////////////////////////////////
14
15 enum rotation {X_ROT, Y_ROT, Z_ROT};
16
17
18 const Int_t N_CHAMBERS_MAX = 10;
19 const Double_t PI = TMath::Pi();
20 const Int_t N_CHAMBER = 6; //!<Number of used chambers
21
22 const Int_t N_PAD = 64; //!<Number of pad channels in a chamber
23 //#define N_sw 64
24 const Int_t N_FW = 64; //!<Number of fieldwire channels in a
      chamber
25
26 const Double_t AREA = 0.065536; //!<detector area in m^2
27 const Double_t CHAMBER_DISTANCES[] = {0., 8.75, 17.5, 26.25, 35,
      43.75}; //!<height position of each chamber in channel units (1
       channel = 4mm)
28
29 //Cuts
30 const Int_t START_CUT = 0; //!<number of cut channels at the
      beginning
31 const Int_t END_CUT = 1; //!<number of cut channels at the end + 1
32 const Int_t CLUSTER_SIZE_THRESHOLD = 6; //!<threshold for the size
       of a cluster
33 const Int_t CHI_2_THRESHOLD = 2; //!<maximum chi^2 to be taken as
      a muon track
34
35 #endif
```

[18] stand 16.11.2016, mit leichten Anpassungen zur besseren Darstellung

C.2. coord.h

```
1  #ifndef COORD_H
2  #define COORD_H
3
4  //! Carthesian and polar coordinates:
5  //! <a href="index.html">descripiton of rotations</a>
6  class Coord {
7
8    public:
9    Double_t x;
10   Double_t y;
11   Double_t z;
12   Double_t phi;
13   Double_t theta;
14   void XRotation(Double_t angle);
15   void YRotation(Double_t angle);
16   void ZRotation(Double_t angle);
17   void CalculatePolarCoord();
18   void Rotation(Double_t rotangle1, Int_t rot1, Double_t rotangle2
         , Int_t rot2);
19 };
20
21 #endif
```

C.3. coord.cpp

```
1  #include "coord.h"
2  #include "mtparameters_5ch.h"
3  //!Rotation of carthesian coordinates around the x axis
4  void Coord::XRotation(Double_t angle)
5  {
6    Double_t rotatedY = cos(angle) * y - sin(angle) * z;
7    Double_t rotatedZ = sin(angle) * y + cos(angle) * z;
8    y = rotatedY;
9    z = rotatedZ;
10 }
11
12 //!Rotation of carthesian coordinates around the y axis
13 void Coord::YRotation(Double_t angle)
14 {
15   Double_t rotatedX = cos(angle) * x + sin(angle) * z;
16   Double_t rotatedZ = -sin(angle) * x + cos(angle) * z;
17   x = rotatedX;
18   z = rotatedZ;
19 }
20
21 //!Rotation of carthesian coordinates around the z axis
22 void Coord::ZRotation(Double_t angle)
23 {
24   Double_t rotatedX = cos(angle) * x - sin(angle) * y;
25   Double_t rotatedY = sin(angle) * x + cos(angle) * y;
26   x = rotatedX;
27   y = rotatedY;
28 }
29
30 //!Combination of rotations
31 /*!
32   Rotates the slopes from the muontelescope system of the given
       run to the
33   * system of reference, namely the horizontal telescope
       orientation
34 */
35 void Coord::Rotation(Double_t rotangle1, Int_t rot1,
36                      Double_t rotangle2, Int_t rot2)
37 {
38   if (rot1 == X_ROT)
39   XRotation(rotangle1);
40   else if (rot1 == Y_ROT)
41   YRotation(rotangle1);
42   else
43   ZRotation(rotangle1);
44
45   if (rot2 == X_ROT)
46   XRotation(rotangle2);
47   else if (rot2 == Y_ROT)
48   YRotation(rotangle2);
49   else
50   ZRotation(rotangle2);
51
```

```
52    Double_t rot_m_x = x / z;
53    Double_t rot_m_y = y / z;
54
55    x = rot_m_x;
56    y = rot_m_y;
57    z = 1.;
58  }
59
60  //!Converts slope coordinates to polar ones
61  void Coord::CalculatePolarCoord()
62  {
63    //calculate polar Coordinates phi and theta from slopes
64    //phi not defined for theta = 0
65    if (x == 0 && y == 0) {
66      phi = 45.;
67      theta = 0.01;
68    }
69    else {
70      //put muon tracks, which look like they come from underground
71      //to the other side
72      if (y < 0)
73        phi = 360. - 180. / PI * acos(x / sqrt(x * x + y * y));
74      //phi and theta calculation
75      //note: mPad = cos(phi) tan(theta) and mFw = sin(phi) tan(
          theta)
76      else phi = acos(x / sqrt(x * x + y * y)) * 180. / PI;
77      theta = atan(sqrt(x * x + y * y)) * 180. / PI;
78    }
79  }
```

C.4. line.h

```
1  #ifndef LINE_H
2  #define LINE_H
3
4  #include "TROOT.h"
5
6  #include "mtparameters_5ch.h"
7
8  class Chamber;
9
10 class Line {
11
12   public:
13   Double_t m; //!<slope of the linear function
14   Double_t b; //!<intercept of the linear function
15   Double_t chi2; //!<chi^2 of the linear function
16   Double_t TrackPoints[N_CHAMBER]; //!<channel coordinate
17   Bool_t isInside; //!<line completely in detector volume
18
19   Line();
20   void Tracking(Chamber** chamber);
21   Int_t Track5pts(Chamber** chamber, Int_t currentChamber, Int_t
        nChannel);
22   void LinearFit(const Double_t* Xpoints, Int_t N_points);
23 };
24
25 #endif
```

C.5. line.cpp

```
1  #include "line.h"
2
3  #include <iostream>
4  #include "mtparameters_5ch.h"
5
6  //using namespace std;
7
8  Line::Line()
9  {
10   chi2 = 999.;
11 }
12
13
14 //!Linear fit for a given number of points
15 /*!
16   Calculates the slope, intercept and chi^2 of a linear function
       for a given number of points.
17 */
18 void Line::LinearFit(const Double_t* xPoints/*!Detector height
     coordinate for muon track*/, Int_t nPoints/*!Number of Points,
     which are used for the fit*/)
19 {
20   Double_t sum_x = 0, sum_y = 0, sum_xx = 0, sum_xy = 0;
21   for (Int_t i = 0; i < nPoints; i++) {
22     //cerr << xPoints[i] << " " << TrackPoints[i] << endl;
23     sum_x += xPoints[i];
24     sum_y += TrackPoints[i];
25     sum_xx += xPoints[i] * xPoints[i];
26     sum_xy += xPoints[i] * TrackPoints[i];
27   }
28
29   if (sum_x * sum_x - N_CHAMBER * sum_xx != 0) {
30     m = (sum_x * sum_y - nPoints * sum_xy) /
31         (sum_x * sum_x - nPoints * sum_xx);
32     b = (sum_xy * sum_x - sum_xx * sum_y) / (sum_x * sum_x -
         nPoints * sum_xx);
33     chi2 = 0;
34     for (Int_t i = 0; i < nPoints; i++) {
35       chi2 += pow(TrackPoints[i]-(m * xPoints[i] + b), 2.0);
36     }
37     chi2 /= (nPoints - 2);
38   }
39 }
40
41
42
43 //!Extracts the muon tracks from the cluster data
44 /*!
45   Goes through all possible cluster combinations, fits them and
       saves the muon tracks if the chi^2 criteria is met
46 */
47 void Line::Tracking(Chamber** chamber)
48 {
```

```
49    Int_t missingChamber = -1;
50    chamber[0]->nFiredChamber = 0;
51    for (Int_t i = 0; i < N_CHAMBER; i++) {
52      //calculate number of fired channels
53      if (chamber[i]->nCluster > 0)
54        chamber[0]->nFiredChamber++;
55      else
56        missingChamber = i;
57    }
58
59    //cerr << endl;
60    //Tracks with 6 points
61    if (chamber[0]->nFiredChamber == N_CHAMBER) {
62      Int_t cl[N_CHAMBER] = {0};
63      //loop through all cluster combinations
64      for (cl[0] = 0; cl[0] < chamber[0]->nCluster; cl[0]++) {
65      for (cl[1] = 0; cl[1] < chamber[1]->nCluster; cl[1]++) {
66      for (cl[2] = 0; cl[2] < chamber[2]->nCluster; cl[2]++) {
67      for (cl[3] = 0; cl[3] < chamber[3]->nCluster; cl[3]++) {
68      for (cl[4] = 0; cl[4] < chamber[4]->nCluster; cl[4]++) {
69      for (cl[5] = 0; cl[5] < chamber[5]->nCluster; cl[5]++) {
70        Line* candidate = new Line();
71        //check if track candidate fits chi2-criterium
72        for (Int_t i = 0; i < N_CHAMBER; i++) {
73          candidate->TrackPoints[i] = chamber[i]->globalCoord[cl[i
              ]];
74        }
75        candidate->LinearFit(CHAMBER_DISTANCES, N_CHAMBER);
76        if (candidate->chi2 < chi2) {
77          //pass fit parameters and trackpoints of candidate to line
78          m = candidate->m;
79          b = candidate->b;
80          chi2 = candidate->chi2;
81          for (Int_t i = 0; i < N_CHAMBER; i++) {
82            TrackPoints[i] = candidate->TrackPoints[i];
83            //cerr << i << " "  << candidate->TrackPoints[i] << " "
                 << CHAMBER_DISTANCES[i] << endl;
84          }
85          delete candidate;
86        }
87        else delete candidate;
88      }
89      }
90      }
91      }
92      }
93      }
94    }
95
96    //Tracks with 5 points
97    if (chamber[0]->nFiredChamber == N_CHAMBER - 1) {
98      Int_t nCluster[N_CHAMBER-1] = {0};
99      Int_t cl[N_CHAMBER] = {0};
100     Double_t z[N_CHAMBER-1] = {0};
101     Int_t l = 0;
```

```
102       //without current chamber
103       for (Int_t k = 0; k < N_CHAMBER; k++) {
104         if (k != missingChamber) {
105           nCluster[l] = chamber[k]->nCluster;
106           l++;
107         }
108       }
109       for (cl[0] = 0; cl[0] < nCluster[0]; cl[0]++) {
110       for (cl[1] = 0; cl[1] < nCluster[1]; cl[1]++) {
111       for (cl[2] = 0; cl[2] < nCluster[2]; cl[2]++) {
112       for (cl[3] = 0; cl[3] < nCluster[3]; cl[3]++) {
113       for (cl[4] = 0; cl[4] < nCluster[4]; cl[4]++) {
114         Line* candidate = new Line();
115         //fit track with 5 points
116         Int_t j = 0;
117         for (Int_t i = 0; i < N_CHAMBER; i++) {
118           if (missingChamber != i) {
119             candidate->TrackPoints[j] = chamber[i]->globalCoord[cl[i
                  ]];
120             z[j] = CHAMBER_DISTANCES[i];
121             //cerr  << z[j] << endl;
122             j++;
123           }
124         }
125         candidate->LinearFit(z, N_CHAMBER - 1);
126         if (candidate->chi2 < chi2) {
127           Double_t fitCoord = candidate->m * CHAMBER_DISTANCES[
                missingChamber] + candidate->b - chamber[missingChamber
                ]->alignmentCoord;
128           //check if this fitted coordinate is inside the detector
                volume
129           isInside = fitCoord >= START_CUT && fitCoord <= chamber
                [0]->nChannel - END_CUT;
130           if (!isInside) {
131             candidate->m = 1000;
132             candidate->chi2 = 1000;
133           }
134           //pass fit parameters and trackpoints of candidate to line
135           m = candidate->m;
136           b = candidate->b;
137           chi2 = candidate->chi2;
138           for(Int_t i = 0; i < N_CHAMBER - 1; i++){
139             TrackPoints[i] = candidate->TrackPoints[i];
140           }
141           delete candidate;
142         }
143         else delete candidate;
144       }
145       }
146       }
147       }
148       }
149     }
150     return;
151   }
```

```
152
153
154
155
156 //!Extracts the muon tracks from the cluster data excluding the
       current Chamber
157 /*!
158   Goes through all possible cluster combinations in the chambers
        excluding the current one and
159   * fits them. If the chi^2 criteria is met, it is checked, if
        there is a hit in the channeldata
160   * on the position the fit indicates. This step is necessary for
        the efficiency calculation.
161   * returns 1 for a triggered hit, 0 for a not triggered hit and
        -1, if no track was found.
162 */
163 Int_t Line::Track5pts(Chamber** chamber, Int_t currentChamber,
       Int_t nChannel)
164 {
165   //5 fired chambers for efficiency calculation
166   if (chamber[0]->nFiredChamber < N_CHAMBER - 1)
167     return -1;
168   Int_t nCluster[N_CHAMBER-1] = {0};
169   Int_t cl[N_CHAMBER] = {0};
170   Double_t z[N_CHAMBER-1] = {0};
171   Int_t l = 0;
172   //without current chamber
173   for (Int_t k = 0; k < N_CHAMBER; k++) {
174     if (k != currentChamber) {
175       nCluster[l] = chamber[k]->nCluster;
176       l++;
177     }
178   }
179   //loop through all chambers but current one
180   for (cl[0] = 0; cl[0] < nCluster[0]; cl[0]++) {
181   for (cl[1] = 0; cl[1] < nCluster[1]; cl[1]++) {
182   for (cl[2] = 0; cl[2] < nCluster[2]; cl[2]++) {
183   for (cl[3] = 0; cl[3] < nCluster[3]; cl[3]++) {
184   for (cl[4] = 0; cl[4] < nCluster[4]; cl[4]++) {
185     Line* candidate = new Line();
186         //fit track with 5 points
187         Int_t j = 0;
188         for (Int_t i = 0; i < N_CHAMBER; i++) {
189       if (currentChamber != i) {
190         candidate->TrackPoints[j] = chamber[i]->globalCoord[cl[i
              ]];
191               z[j] = CHAMBER_DISTANCES[i];
192               j++;
193           }
194         }
195         candidate->LinearFit(z, N_CHAMBER - 1);
196         if (candidate->chi2 < chi2) {
197           //pass fit parameters and trackpoints of candidate to
                line
198           m = candidate->m;
```

```
199             b = candidate->b;
200             chi2 = candidate->chi2;
201             for(Int_t i = 0; i < N_CHAMBER - 1; i++){
202           TrackPoints[i] = candidate->TrackPoints[i];
203             }
204             delete candidate;
205           }
206           else delete candidate;
207       }
208       }
209       }
210       }
211     }
212     //return -1 for no track found
213     if (chi2 > CHI_2_THRESHOLD)
214       return -1;
215
216     //calculate, where the muon would have hit in current chamber
          with the fit
217     Double_t fitCoord = m * CHAMBER_DISTANCES[currentChamber] + b
218                         - chamber[currentChamber]->alignmentCoord;
219     //check if this fitted coordinate is inside the detector volume
220     isInside = fitCoord >= START_CUT && fitCoord <= nChannel -
          END_CUT;
221     if (!isInside)
222       return -1;
223
224     Bool_t isFired = 0;
225     //check if one of the channels around the calculated channel was
          hit
226     if ((Int_t) fitCoord < nChannel - 2 && fitCoord > 1) {
227       if (chamber[currentChamber]->channel[(Int_t) (fitCoord)] == 1
          ||
228           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 1]
              == 1 ||
229           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 1]
              == 1 ||
230           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 2]
              == 1 ||
231           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 2]
              == 1) {
232         isFired = 1;
233       }
234     }
235     //if it's an outer channel
236     if ((Int_t) fitCoord == nChannel - 2) {
237       if (chamber[currentChamber]->channel[(Int_t) (fitCoord)] == 1
          ||
238           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 1]
              == 1 ||
239           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 1]
              == 1 ||
240           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 2]
              == 1){
241         isFired = 1;
```

```
242       }
243     }
244     if ((Int_t) fitCoord == nChannel - 1) {
245       if (chamber[currentChamber]->channel[(Int_t) (fitCoord)] == 1
          ||
246           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 1]
              == 1 ||
247           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 2]
              == 1){
248             isFired = 1;
249       }
250     }
251     if ((Int_t) fitCoord == 0) {
252       if (chamber[currentChamber]->channel[(Int_t) (fitCoord)] == 1
          ||
253           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 1]
              == 1 ||
254           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 2]
              == 1) {
255             isFired = 1;
256       }
257     }
258     if ((Int_t) fitCoord == 1) {
259       if (chamber[currentChamber]->channel[(Int_t) (fitCoord)] == 1
          ||
260           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 1]
              == 1 ||
261           chamber[currentChamber]->channel[(Int_t) (fitCoord) - 1]
              == 1 ||
262           chamber[currentChamber]->channel[(Int_t) (fitCoord) + 2]
              == 1) {
263             isFired = 1;
264       }
265     }
266     //return 1, if the channel in current chamber was fired
267     if (isFired == 1)
268       return 1;
269     else
270       return 0;
271   }
```

C.6. chamber.h

```
1  #ifndef CHAMBER_H
2  #define CHAMBER_H
3
4  #include "TROOT.h"
5  #include "line.h"
6
7  //!all channels in one direction per detector layer
8  class Chamber {
9
10   public:
11   Bool_t channel[N_PAD];      //!<channels of the chamber fired or
         not
12   //Double_t z_coordinate; //used?
13   Double_t clusterCoord[N_PAD/2]; //!<middle of a cluster (but
        statistically smoothed)
14   Double_t globalCoord[N_PAD/2];  //!<clusterCoordinate with
        alignment
15   Int_t nCluster;       //!<number of Clusters for the chamber
16   Int_t clusterSize[N_PAD/2];   //!<number of successive fired
        channels
17   Double_t alignmentCoord;    //!<displacement of the Chamber
18   Double_t SmoothProbability[8];
19   Double_t SmoothSumProbability[8];
20   Int_t nFiredChamber;      //!<number of fired Chambers, saved in
         pad[0] and fw[0]
21   Int_t nChannel;         //!<number of channels in the chamber
22   //Int_t missingChamber;
23   //Bool_t TriggerPattern;
24   TH1D* HistSmooth;     //!<Histogram needed for smoothing
25   TH1D* HistAlignment;      //!<Histogram for determining
        alignment coordinate
26   TH1D* HistAligned;      //!<Histogram for plotting effect of
        alignment
27
28   Line* Best5Pts;       //!<Line with 5 points and chi2 <
        CHI_2_THRESHOLD
29   Line* Best5PtsTrigger;      //!<Line with 5 points and channel
        on position of fit or adjacent channels in the 6th chamber
        are fired
30   Int_t functionValue;      //!<value for calculation of
        efficiency, -1 -> no track; 0 -> track, but not triggered in
        current chamber; 1 -> track and triggered in 6th chamber
31   Chamber(Int_t N, Int_t i);
32   ~Chamber();
33     void FindCluster();
34     void Smooth(Line* line, Int_t i);
35     void Align(Line* line, Int_t i, TH1D* hist);
36 };
37
38 #endif
```

C.7. chamber.cpp

```
1  #include "TH1.h"
2
3  #include <iostream>
4
5  #include "mtparameters_5ch.h"
6
7  #include "chamber.h"
8
9  //! Constructor
10 /*!
11  * creates  Histograms necessary for preanalysis(HistSmooth,
        HistAlignment) and result plots (HistAligned), and initializes
         smoothing variables and lines
12 */
13 Chamber::Chamber(Int_t N /*! number of channels*/, Int_t i /*!
       chamber number*/)
14 {
15   nFiredChamber = 0;
16   alignmentCoord = 0;
17   nChannel = N;
18   HistSmooth = new TH1D(Form("a%d", i), "Smooth", 8, 0, 4);
19   HistAlignment = new TH1D(Form("b%d", i), "AlignmentHist", 15 * N
         , -N, N);
20   HistAligned = new TH1D(Form("c%d", i), "AlignedHist", 15 * N, -N
         , N);
21   Best5Pts = new Line();
22   Best5PtsTrigger = new Line();
23   for(Int_t j = 0; j < 8; j++){
24     SmoothSumProbability[j] = 0.125 * j;
25     SmoothProbability[j] = 0.125;
26   }
27 }
28
29 //! Destructor
30 /*!
31  * deletes Histograms and lines
32 */
33 Chamber::~Chamber()
34 {
35   delete HistSmooth;
36   delete HistAlignment;
37   delete Best5Pts;
38   delete Best5PtsTrigger;
39 }
40
41 //!Finding Clusters in the raw data
42 /*!
43   Search raw data for successive fired channels and unite them to
        a cluster
44   found clusters have to be smaller than CLUSTER_SIZE_THRESHOLD
45
46 */
47 void Chamber::FindCluster()
```

```
48 {
49   nCluster = 0;
50   for (Int_t i = 0; i < nChannel / 2; i++) {
51     clusterSize[i] = 0;
52   }
53   for(Int_t j = 0; j < 64; j++) {
54     if (channel[j] == 1)
55       clusterSize[nCluster]++;
56     Bool_t isEndOfCluster = false;
57     Bool_t isLastChannelHit = false;
58     if (channel[j] == 0 && (j!= 0 && channel[j-1] == 1))
59       isEndOfCluster = true;
60     if (j == 63 && channel[j]==1)
61       isLastChannelHit = true;
62     if (isEndOfCluster || isLastChannelHit) {
63       if (clusterSize[nCluster] >= CLUSTER_SIZE_THRESHOLD)
64         clusterSize[nCluster] = 0;
65       else {
66         clusterCoord[nCluster] = 0.5 + 0.5 * (2.0 * (j - 1)
67                                  - clusterSize[nCluster]);
68         Int_t temp = (Int_t) (clusterCoord[nCluster] / 4.0);
69         Int_t k = (clusterCoord[nCluster] - 4 * temp) * 2.0;
70         clusterCoord[nCluster]= (temp + SmoothSumProbability[k]
71                                  + SmoothProbability[k] *
72                                  rand() / RAND_MAX) * 4.0;
73         globalCoord[nCluster] = clusterCoord[nCluster] +
               alignmentCoord;
74         nCluster++;
75       }
76     }
77   }
78 }
79
80
81 //!Function preparing Smoothing
82 /*!
83   Due to different amplification of the channels caused by the
       electronics there is a
84   4-channel-periodicity in the channel hitmaps, which is taken
       care of by the Smoothing procedure.
85   The necessary Probabilities are calculated from the histogram
       filled by this function.
86 */
87 void Chamber::Smooth(Line* line/*! line used for smoothing */,
     Int_t i/*! number of Chamber */)
88 {
89   if (line->TrackPoints[i] >= 16 && line->TrackPoints[i] < 32) {
90     Double_t C = line->TrackPoints[i] -
91                  (Int_t) (line->TrackPoints[i] / 4.0) * 4.0;
92     HistSmooth->Fill(C);
93   }
94 }
95
96 //!Calculate Alignment
97 /*!
```

```
98      Correction for small tilts of the chambers. Takes the
            difference between TrackPoints and Fit and fills them into a
            histogram.
99      The average of the histogram is later used to correct eventual
            deviations from the perfect chamber position.
100 */
101 void Chamber::Align(Line* line/*! line used for Align */, Int_t i
        /*! number of Chamber */, TH1D* hist/*! Histogramm putting in
        */)
102 {
103   Double_t dx = line->TrackPoints[i] - line->m * CHAMBER_DISTANCES
          [i] - line->b;
104   hist->Fill(dx);
105 }
```

C.8. analysis.cpp

```
1  #include "TTree.h"
2  #include "TFile.h"
3  #include "TH1.h"
4  #include "TH2.h"
5  #include "TCanvas.h"
6  #include "TStyle.h"
7  #include "TColor.h"
8  #include "TLegend.h"
9  #include "TGraphAsymmErrors.h"
10 #include "TPaveLabel.h"
11
12 #include <iostream>
13 #include <time.h>
14
15 #include "mtparameters_5ch.h"
16
17 #include "line.h"
18 #include "chamber.h"
19
20 //only for aclic not needed for building later on
21 #include "line.cpp"
22 #include "chamber.cpp"
23
24 using namespace std;
25 using std::ofstream;
26
27
28 void analysis(Int_t runNumber/*! Number of the run, which is
       analyzed */,
29               Int_t nTimeBins/*! number of timebins */,
30               Int_t chamberNumbers[]/*!production # of chamber*/)
31 {
32   //initialize random number generator
33   srand (time(NULL));
34   //srand(0);
35   Int_t returnCode;
36   /////histograms////////////
37   TH2D* Hitmap = new TH2D(Form("Hitmap␣Run␣%d", runNumber), "
       Hitmap",
38                           45, -1.5, 1.5, 45, -1.5, 1.5);
39   TH1I* PadChannelHits[N_CHAMBER];
40   TH1I* FwChannelHits[N_CHAMBER];
41   //Initialize
42   for (Int_t i = 0; i < N_CHAMBER; i++) {
43     PadChannelHits[i]= new TH1I(Form("Pad␣Run%d␣Chamber%d",
         runNumber, i),
44                                 "PadChannelHits", N_PAD, 0, N_PAD
                                     -1);
45     FwChannelHits[i] = new TH1I(Form("Fw␣Run%d␣Chamber%d",
         runNumber, i),
46                                 "FwChannelHits", N_FW, 0, N_FW-1);
47   }
48
```

```
49    Chamber* Pads[N_CHAMBER];  //pad direction
50    Chamber* Fws[N_CHAMBER];    //fieldwire direction
51
52    for (Int_t i = 0; i < N_CHAMBER; i++) {
53      Pads[i] = new Chamber(N_PAD, i);
54      Fws[i] = new Chamber(N_FW, i);
55    }
56
57    ////////////////////////////////////////
58    //Read in data
59    ////////////////////////////////////////
60
61    //open raw data file
62    FILE* DataFile;
63    ostringstream rn;
64    rn << runNumber;
65    string run_number = rn.str();
66    string inputFile = "Measurements/Mts4Run";
67    inputFile.append(run_number);
68    inputFile.append(".ebe");
69
70    DataFile = fopen(inputFile.c_str(), "r");
71    if (DataFile == 0) {
72      cout << "file:␣" << inputFile << "␣couldn't␣be␣opened" << endl
          ;
73      return;
74    }
75    else
76      cout << "file:␣" << inputFile << "␣was␣opened" << endl;
77
78    //readInTree
79    Bool_t channel[N_CHAMBER][N_FW + N_PAD] = {0};
80    UInt_t event;
81    Double_t eventTime = 0;
82    string pdfout = "Results/Mts4Run";
83    pdfout.append(run_number);
84    pdfout.append(".pdf");
85    string rootout = "Results/Mts4Run";
86    rootout.append(run_number);
87    rootout.append(".root");
88    TFile* RootFile = new TFile(rootout.c_str(), "RECREATE");
89    TTree* ReadInTree = new TTree("ReadInTree","ReadInTree");
90    ReadInTree->Branch("event", &event, "event/i");
91    ReadInTree->Branch("channel",channel,"channel[6][128]/O");
92    ReadInTree->Branch("eventTime", &eventTime, "eventTime/D");
93
94    //raw data columns: #entry, time to last event, 6 times (#fired
         channels and
95    //                  # of hit channels), busy, triggerOK and
         trigger pattern
96    Double_t time = 0;
97    long unsigned int entry;
98    long unsigned int dt;
99    int nFired = 0;
100   int firedChannel;
```

```
101   Int_t busy;
102   Int_t triggerOK;
103   Int_t triggerPat;
104
105   while (!feof(DataFile)) {
106     returnCode = fscanf(DataFile, "%ld", &entry);
107     event = (UInt_t) entry;
108     returnCode = fscanf(DataFile, "%ld", &dt);
109     time += (dt - 100.0) / pow(10, 6); //seconds
110     eventTime = time;
111     if (event % 10000 == 0 && event > 0)
112       cout << event / 1000 << "k␣events␣were␣read␣in!" << endl;
113     for (Int_t j = 0; j < N_CHAMBER; j++) {
114       returnCode = fscanf(DataFile, "%d", &nFired);
115       for (Int_t i = 0; i < nFired; i++) {
116         returnCode = fscanf(DataFile, "%d", &firedChannel);
117         channel[j][firedChannel] = 1;
118       }
119     }
120     returnCode = fscanf(DataFile, "%d", &busy);
121     returnCode = fscanf(DataFile, "%d", &triggerOK);
122     returnCode = fscanf(DataFile, "%d", &triggerPat);
123     ReadInTree->Fill();
124     for (Int_t j = 0; j < N_CHAMBER; j++)
125       for (Int_t i = 0; i < N_FW + N_PAD; i++)
126         channel[j][i] = 0;
127   }
128   //write event#, time and channel array to root tree
129   ReadInTree->Write();
130   Long64_t nEntries = ReadInTree->GetEntries();
131
132   ////////////////////////////////////////////////////
133   //Preanalysis to determine alignment and smoothing
134   ////////////////////////////////////////////////////
135
136   Int_t nPreTracks = 0;
137   Int_t preEvent = 0;
138   while (nPreTracks < 2000 && preEvent <= nEntries) {
139     if (preEvent % 10000 == 0 && preEvent > 0)
140       cout << preEvent / 1000 << "k␣events␣were␣analyzed!" << endl
            ;
141     ReadInTree->GetEntry(preEvent);
142     //initialize channels for each chamber
143     for (Int_t i = 0; i < N_CHAMBER; i++) {
144       for (Int_t j = 0; j < N_FW; j++)
145         Fws[i]->channel[j] = channel[i][j];
146       for (Int_t j = 0; j < N_PAD; j++)
147         Pads[i]->channel[j] = channel[i][j+N_FW];
148       Pads[i]->FindCluster();
149       Fws[i]->FindCluster();
150     }
151     preEvent++;
152     //get tracks from cluster data
153     Line* BestPad = new Line();
154     Line* BestFw = new Line();
```

```
155       BestPad->Tracking(Pads);
156       BestFw->Tracking(Fws);
157       //only take tracks with 6 points and good chi^2 for pre
              analysis
158       Bool_t isGoodChi2 = BestPad->chi2 <= CHI_2_THRESHOLD &&
159                           BestFw->chi2 <= CHI_2_THRESHOLD;
160       Bool_t areAllChambersFired = Pads[0]->nFiredChamber == 6 &&
161                                    Fws[0]->nFiredChamber == 6;
162       if (!(isGoodChi2 && areAllChambersFired)) {
163         continue; //->bad events
164       }
165       //fill good events into histogram
166       for (Int_t i = 0; i < N_CHAMBER; i++) {
167         Pads[i]->Smooth(BestPad, i);
168         Pads[i]->Align(BestPad, i, Pads[i]->HistAlignment);
169         Fws[i]->Smooth(BestFw, i);
170         Fws[i]->Align(BestFw, i, Fws[i]->HistAlignment);
171       }
172       nPreTracks++;
173       delete BestPad;
174       delete BestFw;
175     }
176
177     //Set alignmentCoordinate and smoothing probabilities
178     for (Int_t i = 0; i < N_CHAMBER; i++) {
179       Pads[i]->alignmentCoord = Pads[i]->HistAlignment->GetMean();
180       Fws[i]->alignmentCoord = Fws[i]->HistAlignment->GetMean();
181       //cerr << Pads[i]->alignmentCoord << " " << Fws[i]->
              alignmentCoord << endl;
182
183       for (Int_t j = 0; j < 8; j++) {
184         Pads[i]->SmoothProbability[j] = Pads[i]->HistSmooth->
                GetBinContent(j+1)
185                                       / Pads[i]->HistSmooth->
                                            GetEntries();
186         Fws[i]->SmoothProbability[j] = Fws[i]->HistSmooth->
                GetBinContent(j+1)
187                                      / Fws[i]->HistSmooth->
                                           GetEntries();
188
189         if (j > 0) {
190           Pads[i]->SmoothSumProbability[j] = Pads[i]->
                  SmoothSumProbability[j-1]
191                                            + Pads[i]->
                                                SmoothProbability[j
                                                -1];
192           Fws[i]->SmoothSumProbability[j] = Fws[i]->
                  SmoothSumProbability[j-1]
193                                           + Fws[i]->
                                               SmoothProbability[j
                                               -1];
194         }
195         else {
196           Pads[i]->SmoothSumProbability[j] = 0;
197           Fws[i]->SmoothSumProbability[j] = 0;
```

```
198         }
199       }
200     }
201     cout  << "end␣of␣preanalysis" <<endl;
202
203     //variables for result root tree
204     Double_t mPad;
205     Double_t mFw;
206     Double_t mPadEffTrack[N_CHAMBER];
207     Double_t mFwEffTrack[N_CHAMBER];
208     Double_t mPadEffTrigger[N_CHAMBER];
209     Double_t mFwEffTrigger[N_CHAMBER];
210
211     UInt_t actualEvent = 0;
212
213     TTree* ResultsTree = new TTree("resultsTree","resultsTree");
214     ResultsTree->Branch("actualEvent", &actualEvent, "actualEvent/i"
          );
215     ResultsTree->Branch("mPad", &mPad, "mPad/D");
216     ResultsTree->Branch("mFw", &mFw, "mFw/D");
217     ResultsTree->Branch("mPadEffTrack", &mPadEffTrack, "mPadEffTrack
          [6]/D");
218     ResultsTree->Branch("mFwEffTrack", &mFwEffTrack, "mFwEffTrack
          [6]/D");
219     ResultsTree->Branch("mPadEffTrigger", &mPadEffTrigger, "
          mPadEffTrigger[6]/D");
220     ResultsTree->Branch("mFwEffTrigger", &mFwEffTrigger, "
          mFwEffTrigger[6]/D");
221
222     //histograms for plots (helpful to see, if everything worked
          alright)
223     TH1D* EventRate = new TH1D("Events␣vs␣time", "Events␣vs␣time",
224                                nTimeBins, 0, time / 60.);
225     TH1D* GoodEventRate = new TH1D("Good␣events␣vs␣time", "Good␣
          events␣vs␣time",
226                                    nTimeBins, 0, time / 60.);
227     //efficiency histograms
228     TEfficiency* Efficiency[N_CHAMBER];
229
230     for (Int_t i = 0; i < N_CHAMBER; i++) {
231       Efficiency[i] = new TEfficiency(Form("Efficiency/time␣%d␣%d",
            runNumber, i),
232                                       Form("MT%d;Time␣[min]",
                                             chamberNumbers[i]),
233                                       nTimeBins, 0, time/60.);
234     }
235
236     ofstream TempFile("output");
237
238     ////////////////////////
239     //Main analysis
240     ////////////////////////
241
242     while (actualEvent < nEntries - 1) {
243       Bool_t isImportant = false; //event useful for analysis (5+
```

```
        trackpoints)
244     Bool_t isTriggered = false;
245     mPad = 1000;
246     mFw = 1000;
247     for (Int_t i = 0; i < N_CHAMBER; i++) {
248         mPadEffTrack[i] = 1000;
249         mFwEffTrack[i] = 1000;
250         mPadEffTrigger[i] = 1000;
251         mFwEffTrigger[i] = 1000;
252     }
253     if (actualEvent % 10000 == 0 && actualEvent > 0)
254       cout << actualEvent/1000 << "k events analyzed!" << endl;
255
256     //same procedure as in preanalysis
257     ReadInTree->GetEntry(actualEvent);
258     for (Int_t i = 0; i < N_CHAMBER; i++) {
259       for (Int_t j = 0; j < N_FW; j++) {
260         Fws[i]->channel[j] = channel[i][j];
261         if (Fws[i]->channel[j] == 1)
262           FwChannelHits[i]->Fill(j);
263       }
264       for (Int_t j = 0; j < N_PAD; j++) {
265         Pads[i]->channel[j] = channel[i][j + N_FW];
266         if (Pads[i]->channel[j] == 1)
267           PadChannelHits[i]->Fill(j);
268       }
269       Pads[i]->FindCluster();
270       Fws[i]->FindCluster();
271     }
272     Line* BestPad = new Line();
273     Line* BestFw = new Line();
274     BestPad->Tracking(Pads);
275     BestFw->Tracking(Fws);
276     EventRate->Fill(eventTime/60.);
277
278     //efficiency calculation: check if 6th chamber got hit on the
        channel
279     //pointed to by the linear fit, if there was found a track in
        the other 5
280     //chambers
281     for (Int_t i = 0; i < N_CHAMBER; i++) {
282       isTriggered = false;
283       Line* Pad5Pts = new Line();
284       Line* Fw5Pts = new Line();
285       Pads[i]->functionValue = Pad5Pts->Track5pts(Pads, i, N_PAD);
286       Fws[i]->functionValue = Fw5Pts->Track5pts(Fws, i, N_FW);
287
288       //functionValue = -1, if no track with 5 points
289       //              = 0, if track was found
290       if (Pads[i]->functionValue > -1 && Fws[i]->functionValue >
            -1) {
291         isImportant = true;
292         Pads[i]->Best5Pts->m = Pad5Pts->m;
293         Pads[i]->Best5Pts->b = Pad5Pts->b;
294         Pads[i]->Best5Pts->chi2 = Pad5Pts->chi2;
```

```
295           Fws[i]->Best5Pts->m = Fw5Pts->m;
296           Fws[i]->Best5Pts->b = Fw5Pts->b;
297           Fws[i]->Best5Pts->chi2 = Fw5Pts->chi2;
298           mPadEffTrack[i] = Pads[i]->Best5Pts->m;
299           mFwEffTrack[i] = Fws[i]->Best5Pts->m;
300           mPadEffTrigger[i] = 1000;
301           mFwEffTrigger[i] = 1000;
302
303           //functionValue = 1 if 6th chamber got hit at the right
                 channel
304           if (Pads[i]->functionValue == 1 && Fws[i]->functionValue
                 == 1) {
305             Pads[i]->Best5PtsTrigger->m = Pad5Pts->m;
306             Pads[i]->Best5PtsTrigger->b = Pad5Pts->b;
307             Pads[i]->Best5PtsTrigger->chi2 = Pad5Pts->chi2;
308             Fws[i]->Best5PtsTrigger->m = Fw5Pts->m;
309             Fws[i]->Best5PtsTrigger->b = Fw5Pts->b;
310             Fws[i]->Best5PtsTrigger->chi2 = Fw5Pts->chi2;
311             mPadEffTrigger[i] = Pads[i]->Best5PtsTrigger->m;
312             mFwEffTrigger[i] = Fws[i]->Best5PtsTrigger->m;
313             isTriggered = true;
314           }
315           Efficiency[i]->Fill(isTriggered, eventTime/60.);
316         }
317
318         delete Pad5Pts;
319         delete Fw5Pts;
320       }
321       if (BestPad->chi2 < CHI_2_THRESHOLD && BestFw->chi2 <
             CHI_2_THRESHOLD) {
322         TempFile << actualEvent << "\t" << BestPad->m << "\t" <<
               BestFw->m << endl;
323         isImportant = true;
324         for(Int_t i = 0; i < N_CHAMBER; i++){
325           Pads[i]->Align(BestPad, i, Pads[i]->HistAligned);
326           Fws[i]->Align(BestFw, i, Fws[i]->HistAligned);
327         }
328
329         GoodEventRate->Fill(eventTime/60.);
330         mPad = BestPad->m;
331         mFw = BestFw->m;
332         Hitmap->Fill(BestPad->m, BestFw->m);
333       }
334       //else cerr << endl;
335       if (isImportant) {
336         ResultsTree->Fill();
337       }
338       delete BestPad;
339       delete BestFw;
340       actualEvent++;
341     }
342     TempFile.close();
343     cout  << "end" <<endl;
344     //End of event loop
345     //////////save/////////
```

```
346   ResultsTree->Write();
347
348   //calculate histograms where time is on the x-axis
349   Double_t timeStep = time / nTimeBins;
350
351   for (Int_t i = 1; i < nTimeBins + 1; i++) {
352     EventRate->SetBinError(i, sqrt(EventRate->GetBinContent(i)) /
          timeStep);
353     EventRate->SetBinContent(i, EventRate->GetBinContent(i) /
          timeStep);
354     GoodEventRate->SetBinError(i, sqrt(GoodEventRate->
          GetBinContent(i))
355                                / timeStep);
356     GoodEventRate->SetBinContent(i, GoodEventRate->GetBinContent(i
          )
357                                  / timeStep);
358   }
359   ////////////////////////////////
360   //Output
361   ////////////////////////////////
362   gStyle->SetOptStat(0);
363   gStyle->SetErrorX(0);
364   //Events and good events over time
365   TCanvas *c1 = new TCanvas("c1", "Events␣vs␣Time", 200, 10, 620,
        700);
366   c1->SetGrid();
367   EventRate->SetDirectory(0);
368   EventRate->SetMarkerSize(0.7);
369   EventRate->SetMarkerStyle(21);
370   EventRate->SetMarkerColor(kBlue + 2);
371   EventRate->SetLineColor(kBlue + 2);
372   EventRate->SetMinimum(0);
373   EventRate->GetXaxis()->SetTitle("Time␣[min]");
374   EventRate->GetYaxis()->SetTitle("Rate␣[Hz]");
375   EventRate->Draw("P0E1");
376
377   GoodEventRate->SetDirectory(0);
378   GoodEventRate->SetMarkerSize(0.7);
379   GoodEventRate->SetMarkerStyle(21);
380   GoodEventRate->SetMarkerColor(kGreen + 2);
381   GoodEventRate->SetLineColor(kGreen + 2);
382   GoodEventRate->Draw("SAME");
383
384   TLegend* eventLegend = new TLegend(0.75, 0.65, 0.89, 0.77);
385   eventLegend->AddEntry(EventRate, "events", "lep");
386   eventLegend->AddEntry(GoodEventRate, "tracks", "lep");
387   eventLegend->SetFillColor(kWhite);
388   eventLegend->Draw();
389
390   //Efficiency for each chamber and direction
391   TCanvas *c2 = new TCanvas("c2", "Efficiency␣vs␣Time", 200, 10,
        620, 700);
392   TPaveLabel* effTitle = new TPaveLabel(0.1, 0.96, 0.9, 0.99,
393                                         "Efficiency␣over␣time");
394   effTitle->SetLineColor(kWhite);
```

```
395   effTitle->SetFillColor(kWhite);
396   effTitle->SetShadowColor(kWhite);
397   effTitle->Draw();
398   TPad* effPad = new TPad("Graphs","Graphs",0.01,0.05,0.95,0.95);
399   effPad->Draw();
400   effPad->cd();
401   effPad->Divide(2,3);
402   for (Int_t i = 0; i < N_CHAMBER; i++) {
403     effPad->cd(i+1);
404     effPad->cd(i+1)->SetGrid();
405     Efficiency[i]->SetDirectory(0);
406     Efficiency[i]->SetMarkerSize(0.7);
407     Efficiency[i]->SetMarkerStyle(21);
408     Efficiency[i]->SetMarkerColor(kBlue + 2);
409     Efficiency[i]->SetLineColor(kBlue + 2);
410     Efficiency[i]->Draw("AP");
411     gPad->Update();
412     Efficiency[i]->GetPaintedGraph()->GetYaxis()->SetRangeUser
          (0.5, 1);
413     Efficiency[i]->GetPaintedGraph()->GetXaxis()->SetRangeUser(0,
          time/60.);
414     gPad->Update();
415   }
416
417   //ChannelHits for each chamber and direction
418   TCanvas *c3 = new TCanvas("c3","Channelhits",200,10,620,700);
419   TPaveLabel* hitsTitle = new TPaveLabel(0.1,0.96,0.9,0.99,"
        Channelhits");
420   hitsTitle->SetLineColor(kWhite);
421   hitsTitle->SetFillColor(kWhite);
422   hitsTitle->SetShadowColor(kWhite);
423   hitsTitle->Draw();
424   TPad* hitsPad = new TPad("Graphs","Graphs",0.01,0.05,0.95,0.95);
425   hitsPad->Draw();
426
427   hitsPad->SetGrid();
428   //~ hitsPad->cd();
429   //~ hitsPad->Divide(2,3);
430   //~ for (Int_t i = 0; i < N_CHAMBER; i++) {
431   Int_t i = 0;
432   cerr << PadChannelHits[i]->GetXaxis()->GetLabelSize();
433     //PadChannelHits[i]->SetTitle(Form("MT%d", chamberNumbers[i]))
          ;
434     PadChannelHits[i]->SetTitle(Form("MT30"));
435     PadChannelHits[i]->GetXaxis()->SetTitle("Channel");
436     PadChannelHits[i]->GetXaxis()->CenterTitle();
437     PadChannelHits[i]->GetXaxis()->SetLabelSize(0.04);
438     PadChannelHits[i]->GetYaxis()->SetTitle("Events");
439     PadChannelHits[i]->GetYaxis()->CenterTitle();
440     PadChannelHits[i]->GetYaxis()->SetLabelSize(0.04);
441     PadChannelHits[i]->GetYaxis()->SetTitleOffset(1.3);
442     cerr << PadChannelHits[i]->GetYaxis()->GetTitleSize()<< endl;
443     PadChannelHits[i]->GetYaxis()->SetTitleSize(0.04);
444     PadChannelHits[i]->GetXaxis()->SetTitleSize(0.04);
445     PadChannelHits[i]->SetMinimum(1);
```

```
446       PadChannelHits[i]->SetLineWidth(3);
447       PadChannelHits[i]->SetLineColor(kBlue+2);
448       //~ hitsPad->cd(i+1);
449       //~ hitsPad->cd(i+1)->SetGrid();
450       PadChannelHits[i]->Draw();
451       FwChannelHits[i]->SetLineWidth(3);
452       FwChannelHits[i]->SetLineColor(kGreen+2);
453       FwChannelHits[i]->Draw("SAME");
454       gPad->SetLogy();
455       PadChannelHits[i]->GetYaxis()->SetNdivisions(10);
456
457       TLegend* channelLegend = new TLegend(0.72, 0.11, 0.89, 0.21);
458       channelLegend->AddEntry(PadChannelHits[i], "pads","l");
459       channelLegend->AddEntry(FwChannelHits[i], "fws", "l");
460       channelLegend->SetFillColor(kWhite);
461       channelLegend->Draw();
462     //~ }
463
464     //Alignment for each chamber and direction
465     TCanvas *c4 = new TCanvas("c4", "Alignment", 200, 10, 620, 700);
466     TPaveLabel* alignTitle = new TPaveLabel(0.1,0.96,0.9,0.99,"
          Alignment");
467     alignTitle->SetLineColor(kWhite);
468     alignTitle->SetFillColor(kWhite);
469     alignTitle->SetShadowColor(kWhite);
470     alignTitle->Draw();
471     TPad* alignPad = new TPad("Graphs","Graphs",0.01,0.05,0.95,0.95)
          ;
472     alignPad->Draw();
473
474     alignPad->SetGrid(); // wieder raus
475     //~ alignPad->cd();
476     //~ alignPad->Divide(2,3);
477     //~ for(Int_t i = 0; i < N_CHAMBER; i++){
478       //Pads[i]->HistAlignment->SetTitle(Form("MT%d", chamberNumbers
            [i]));
479       Pads[i]->HistAlignment->SetTitle(Form("MT30"));
480       Pads[i]->HistAlignment->GetXaxis()->SetTitle("Alignment");
481       Pads[i]->HistAlignment->GetXaxis()->CenterTitle();
482       Pads[i]->HistAlignment->GetYaxis()->SetTitle("Events");
483       Pads[i]->HistAlignment->GetYaxis()->CenterTitle();
484   Pads[i]->HistAlignment->GetXaxis()->SetLabelSize(0.04);
485       Pads[i]->HistAlignment->GetYaxis()->SetLabelSize(0.04);
486       Pads[i]->HistAlignment->GetYaxis()->SetTitleOffset(1.3);
487       Pads[i]->HistAlignment->GetYaxis()->SetTitleSize(0.04);
488       Pads[i]->HistAlignment->GetXaxis()->SetTitleSize(0.04);
489       Pads[i]->HistAlignment->SetAxisRange(-2.5,2.5);
490       Pads[i]->HistAlignment->SetMinimum(1);
491       Pads[i]->HistAlignment->SetLineWidth(3);
492       Pads[i]->HistAlignment->SetLineColor(kBlue+2);
493       Fws[i]->HistAlignment->SetAxisRange(-2.5,2.5);
494       //~ alignPad->cd(i+1);
495       //~ alignPad->cd(i+1)->SetGrid();
496       Pads[i]->HistAlignment->Draw();
497       Fws[i]->HistAlignment->SetLineWidth(3);
```

```
498       Fws[i]->HistAlignment->SetLineColor(kGreen+2);
499       Fws[i]->HistAlignment->Draw("SAME");
500       gPad->SetLogy();
501       Pads[i]->HistAlignment->GetYaxis()->SetNdivisions(10);
502
503       //clone histogram, so the legend displays the right linestyle
504       TH1D* fwClone = (TH1D*) Fws[i]->HistAlignment->Clone();
505       fwClone->SetLineColor(kGreen+2);
506       fwClone->SetLineWidth(3);
507       TH1D* padClone = (TH1D*) Pads[i]->HistAlignment->Clone();
508       padClone->SetLineWidth(3);
509       TLegend* alignmentLegend = new TLegend(0.72, 0.79, 0.89, 0.89)
              ;
510       alignmentLegend->AddEntry(padClone, "pads", "l");
511       alignmentLegend->AddEntry(fwClone, "fws", "l");
512       alignmentLegend->SetFillColor(kWhite);
513       alignmentLegend->Draw();
514     //~ }
515
516
517     //carthesian hitmap
518     TCanvas *c5 = new TCanvas("c5","Hitmap", 200, 10, 620, 700);
519     Hitmap->GetYaxis()->SetTitle("m_fw");
520     Hitmap->GetXaxis()->SetTitle("m_pad");
521     gPad->SetFrameFillColor(kBlack);
522     Double_t red[9] = {0./255., 61./255., 89./255., 122./255.,
            143./255.,
523                        160./255., 185./255., 204./255., 231./255.};
524     Double_t green[9] = {0./255., 0./255., 0./255., 0./255.,
            14./255.,
525                          37./255., 72./255., 132./255., 235./255.};
526     Double_t blue[9] = {0./255., 140./255., 224./255., 144./255.,
            4./255.,
527                         5./255., 6./255., 9./255., 13./255.};
528     Double_t stops[9] = {0.0000, 0.1250, 0.2500, 0.3750, 0.5000,
529                          0.6250, 0.7500, 0.8750, 1.0000};
530     TColor::CreateGradientColorTable(9, stops, red, green, blue,
            255);
531     Hitmap->SetContour(99);
532     Hitmap->Draw("COLZ");
533
534     //print to pdf and put it into results folder
535     cerr << "time␣in␣s␣" << time << endl;
536     c1->Print("test.pdf(");
537     c2->Print("test.pdf");
538     c3->Print("test.pdf");
539     c4->Print("test.pdf");
540     c5->Print("test.pdf)");
541     Char_t Command[100];
542     sprintf(Command,"mv␣test.pdf␣Results/Mts4Run%d.pdf", runNumber);
543     returnCode = system(Command);
544     if (returnCode != 0) {
545       cerr << "file␣couldn't␣be␣moved" << endl;
546     }
547
```

```
548     //write runnumber and calculated time to result text file
549     ostringstream OutputStream;
550     OutputStream << "Results/Mts4Run" << runNumber;
551     string txtout = OutputStream.str();
552     txtout.append(".txt");
553     ofstream txt(txtout.c_str());
554     txt << "Run" << "\t" << runNumber << endl;
555     txt << "Time" << "\t" << time  << endl;
556     txt.close();
557     RootFile->Close();
558 }
```

C.9. GUI.h

```
1  #ifndef GUI_H
2  #define GUI_H
3
4
5  #include "TGNumberEntry.h"
6  #include "TGFrame.h"
7  #include "TGTextEdit.h"
8  #include "TGButton.h"
9  #include "TGLabel.h"
10
11
12 class MtsMain{
13
14   RQ_OBJECT("MtsMain")
15   private:
16   public:
17   MtsMain();
18   ~MtsMain();
19   TGMainFrame *MainFrame;
20   TGTextEntry *IpAddressT; //!<Ip address of the muon telescopes
        raspberry pi, standard is 192.168.0.1
21   TGNumberEntry *RunNumberT; //!<run number for current
        measurements
22   TGNumberEntry *StatisticsT; //!<number of events till the
        measurements stop, -1 for infinite measuring until manual
        stopping
23   TGTextEntry *UsedGasT; //!<information on used gas
24   TGNumberEntry *HV_SenseWireT; //!<High Voltage on the sense
        wires in V
25   TGNumberEntry *HV_SW_CurrentT; //!<Current on the sense wires
26   TGTextEdit *AdditionalInformationT;
27   TGLabel *StatusL;
28   TGLabel *EventsL;
29   TGNumberEntry *UpdateIntervalT; //!<time in minutes in between
        updates
30   TGNumberEntry *ProcessRunNumberT; //!<run number for analysing a
         single run
31   TGNumberEntry *BinsizeThetaT; //!<size of one theta bin (polar
        direction) in degree
32   TGNumberEntry *BinsizePhiT; //!<size of one phi bin (azimuthal
        direction) in degree
33   TGNumberEntry *TimebinT; //!<number of timebins for the analysis
34   TGNumberEntry *PolarAngleT; //!<rotation angle for the polar
        angle theta in degree, check documentation for more
        information on rotations
35   TGNumberEntry *AzimuthalAngleT; //!<rotation angle for the
        azimuthal angle phi in degree, check documentation for more
        information on rotations
36   TGCheckButton *OtherRotationB; //!<check, if you want to rotate
        around different axes, check documentation for more
        information on rotations
37   TGTextEntry *RunNumbersT; //!<compilation of runs for one
        telescope location, that are to be merged via a weighted mean
```

```
38    TTimer *timer;
39    void Startup();
40    void CloseWindow();
41    void TextEdit(char*);
42    void UpdateIntervalEdit(Long_t val);
43    //~ void test();
44    void Start();
45    void StartRun(); //maybe switch to popup
46    void Stop();
47    void Update();
48    void LoadFormerSettings();
49    void GetWeightedHistogramMean();
50    void RemoveData();
51    void CopyData();
52    void Analyze();
53    void SinglePlot();
54    void MultiplePlot();
55    void CalculateIntegratedFlux();
56    void AnalyzePolar();
57    //void AnalyzeRun();
58    //~ void thetabin_edit(Long_t val);
59    //~ void phibin_edit(Long_t val);
60    char Line[200], ValueName[200];
61    char buffer[100];
62    char Command[400];
63    Int_t runNumberInt;
64    Int_t returnCode;
65    Int_t nChamber;
66    Int_t nUsedChambers;
67    Bool_t usedChambers[N_CHAMBERS_MAX];
68    const char* ipAddressString;
69    const char* additionalInformation;
70    char filename[500];
71    Int_t chamberSizeX;
72    Int_t chamberSizeY;
73    Int_t nAdcChannels;
74    Bool_t reorderBitsForMt;
75    Int_t prevalence;
76    Int_t wtsBreakLine;
77    Int_t chamberPosition[N_CHAMBERS_MAX];
78    Int_t ChamberNumber[N_CHAMBERS_MAX];
79    Int_t dPhi, dTheta;
80    Int_t nBinPhi, nBinTheta;
81    Int_t runToAnalyze;
82    Double_t time;
83    Double_t phiMin;
84    Double_t phiMax;
85    Double_t thetaMin;
86    Double_t thetaMax;
87    Int_t nTimeBins;
88    Double_t polarAngleDegree, azimuthalAngleDegree;
89    Int_t polarRotation, azimuthalRotation;
90    Bool_t isConnected;
91 };
92
```

```
93  class MtsPopup{
94    RQ_OBJECT("MtsPopup")
95    private:
96    public:
97    TGTransientFrame *PopupFrame;
98    MtsPopup(const TGWindow *p, const TGWindow *main, UInt_t w,
          UInt_t h,
99              const char *test, Int_t preset = 100,
100             UInt_t options = kVerticalFrame);
101   ~MtsPopup();
102   char buffer[100];
103   Char_t Command[400];
104   Int_t returnCode;
105   TGTextButton *YesPopupB;
106   //~ TGTextButton *NoPopupB;
107   TGTextButton *RemovePopupB;
108   TGTextButton *CopyPopupB;
109   TGTextButton *ClosePopupB;
110   TGTextButton *FullPopupB;
111   TGTextButton *RunPopupB;
112   TGTextButton *PolarPopupB;
113   TGTextEntry *RemoveT;
114   void FullAnalysis();
115   void RunAnalysis();
116   void PolarAnalysis();
117   void Remove();
118   void Copy();
119   void CloseWindow();
120   //~ void DoClose();
121   //void CloseApp();
122 };
123
124 class RotationPopup{
125   RQ_OBJECT("RotationPopup")
126   private:
127   public:
128   TGTransientFrame *RotationFrame;
129   RotationPopup(const TGWindow *p, const TGWindow *main, UInt_t w,
          UInt_t h,
130                 UInt_t options = kVerticalFrame);
131   ~RotationPopup();
132   TGRadioButton *x1B;
133   TGRadioButton *y1B;
134   TGRadioButton *z1B;
135   TGNumberEntry *FirstAngleT;
136   TGRadioButton *x2B;
137   TGRadioButton *y2B;
138   TGRadioButton *z2B;
139   TGNumberEntry *SecondAngleT;
140   TGTextButton *OkRotationB;
141   TGTextButton *CloseRotationB;
142   void Ok();
143   void CloseWindow();
144   //~ void DoClose();
145 };
```

```
146
147 #endif
```

C.10. GUI.cpp

```
1  #include <TG3DLine.h>
2  #include <TGNumberEntry.h>
3  #include <TGComboBox.h>
4  #include <TGFrame.h>
5  #include <TGButtonGroup.h>
6  #include <TGButton.h>
7  #include <TGLabel.h>
8  #include <TGTextEdit.h>
9  #include <Riostream.h>
10 #include <RQ_OBJECT.h>
11 #include <TApplication.h>
12 #include <TH1.h>
13 #include <TH2.h>
14 #include <TStyle.h> //gstyle manipulation
15 #include <TLatex.h> //latex headlines
16 #include <TPaletteAxis.h> //color map
17 #include <TCanvas.h>
18 #include <TFile.h>
19 #include <TTree.h>
20 #include <TMath.h>
21 #include <TColor.h>
22 #include <TEfficiency.h>
23 #include <TLegend.h>
24 #include <TGraphAsymmErrors.h>
25 #include <TPaveLabel.h>
26 #include <fstream>
27 #include <sstream> //name streaming
28 #include <string>
29 #include <iostream> //debugging output
30 #include <time.h>
31 #include <TF1.h>
32
33 #include "mtparameters_5ch.h"
34 #include "GUI.h"
35 #include "coord.h"
36
37 //because >ACLIC
38 #include "analysis.cpp"
39 #include "coord.cpp"
40
41 MtsMain *mtsMain;
42
43
44
45 //!Popup Frames with different presets for error messages and user
       input
46 /*!
47   2 - Popup for overwriting an already existing run
48   * 3 - Copy data files from raspberry pi to notebook
49   * 4 - Remove data files from raspberry pi to notebook
50   * 5 - Analysis of one file
51   * every other number: label for error message + close button
52 */
```

```
53 MtsPopup::MtsPopup(const TGWindow *p, const TGWindow *main, UInt_t
       w, UInt_t h,
54                       const char* message, Int_t preset, UInt_t
                            options)
55 {
56   //Popups with different presets for errors and user input
57
58   //new frame
59   TGLayoutHints* layoutNormal = new TGLayoutHints(kLHintsTop |
       kLHintsCenterX,
60                                                   2, 2, 2, 2);
61   TGLayoutHints* layoutFrame = new TGLayoutHints(kLHintsTop|
       kLHintsExpandX |
62                                                  kLHintsExpandY);
63   TGLayoutHints* layoutGroup = new TGLayoutHints(kLHintsCenterX,
       10, 10, 2, 2);
64   PopupFrame = new TGTransientFrame(p, main, w, h, options);
65   PopupFrame->SetName("PopupFrame");
66   PopupFrame->SetWindowName("Mts4GUI");
67   PopupFrame->CenterOnParent();
68   //so cleanup works for textentries as well
69   PopupFrame->SetCleanup(kDeepCleanup);
70   TGVerticalFrame* VFrame1 = new TGVerticalFrame(PopupFrame, 1, 1)
       ;
71   //message label, necessary for all presets
72   TGLabel* messageL = new TGLabel(VFrame1, message);
73   VFrame1->AddFrame(messageL, layoutNormal);
74   if (preset == 2) {
75     //Yes - No Popup for starting the run even though it already
         exists
76     TGHButtonGroup *fButtonGroup = new TGHButtonGroup(VFrame1, "")
         ;
77     YesPopupB = new TGTextButton(fButtonGroup, "Yes");
78     ClosePopupB = new TGTextButton(fButtonGroup, "No");
79     fButtonGroup->SetLayoutHints(layoutGroup);
80     fButtonGroup->Show();
81     VFrame1->AddFrame(fButtonGroup, layoutNormal);
82     VFrame1->Resize(messageL->GetWidth() + 4, messageL->GetHeight
         () +
83                     fButtonGroup->GetHeight() + 20);
84     PopupFrame->AddFrame(VFrame1, layoutFrame);
85     PopupFrame->Resize(VFrame1->GetWidth() + 4, VFrame1->GetHeight
         () + 6);
86     YesPopupB->Connect("Clicked()", "MtsMain", mtsMain, "StartRun
         ()");
87     YesPopupB->Connect("Clicked()", "MtsPopup", this, "CloseWindow
         ()");
88     ClosePopupB->Connect("Clicked()", "MtsPopup", this, "
         CloseWindow()");
89   }
90   else if (preset == 3 || preset == 4) {
91     //Popup for copying (preset 4) and removing (preset 3) data
         from the pi
92     //text entry to specify the runs
93     RemoveT = new TGTextEntry(VFrame1);
```

```
94      RemoveT->Resize(174,18);
95      VFrame1->AddFrame(RemoveT, layoutNormal);
96      TGHButtonGroup *fButtonGroup2 = new TGHButtonGroup(VFrame1, ""
            );
97      if (preset == 3) {
98        RemovePopupB = new TGTextButton(fButtonGroup2, "Remove");
99        RemovePopupB->Connect("Clicked()", "MtsPopup", this, "Remove
              ()");
100     }
101     else {
102       CopyPopupB = new TGTextButton(fButtonGroup2, "Copy");
103       CopyPopupB->Connect("Clicked()", "MtsPopup", this, "Copy()")
              ;
104     }
105     ClosePopupB = new TGTextButton(fButtonGroup2, "Close");
106     fButtonGroup2->SetLayoutHints(layoutGroup);
107     fButtonGroup2->Show();
108     VFrame1->AddFrame(fButtonGroup2, layoutNormal);
109     VFrame1->Resize(messageL->GetWidth()+4, messageL->GetHeight()
            +
110                     fButtonGroup2->GetHeight() + RemoveT->
                        GetHeight() + 20);
111     PopupFrame->AddFrame(VFrame1, layoutFrame);
112     PopupFrame->Resize(VFrame1->GetWidth() + 4, VFrame1->GetHeight
            () + 6);
113   }
114   else if (preset == 5) {
115     //popup for the single file analysis: gives you three options:
116     //full analysis, run analysis and polar analysis
117     TGHButtonGroup *fButtonGroup3 = new TGHButtonGroup(VFrame1, ""
            );
118     FullPopupB = new TGTextButton(fButtonGroup3, "Full␣Analysis");
119     RunPopupB = new TGTextButton(fButtonGroup3, "Run␣Analysis");
120     PolarPopupB = new TGTextButton(fButtonGroup3, "Polar␣Analysis"
            );
121     ClosePopupB = new TGTextButton(fButtonGroup3, "Close");
122     fButtonGroup3->SetLayoutHints(layoutGroup);
123     fButtonGroup3->Show();
124     VFrame1->AddFrame(fButtonGroup3, layoutNormal);
125     VFrame1->Resize(fButtonGroup3->GetWidth() + 10,
126                     messageL->GetHeight() + fButtonGroup3->
                        GetHeight() + 10);
127     PopupFrame->AddFrame(VFrame1, layoutFrame);
128     PopupFrame->Resize(VFrame1->GetWidth() + 4, VFrame1->GetHeight
            () + 6);
129     FullPopupB->Connect("Clicked()", "MtsPopup", this, "
            FullAnalysis()");
130     RunPopupB->Connect("Clicked()", "MtsPopup", this, "RunAnalysis
            ()");
131     PolarPopupB->Connect("Clicked()", "MtsPopup", this, "
            PolarAnalysis()");
132     //close the popup as well after pressing the button
133     FullPopupB->Connect("Clicked()", "MtsPopup", this, "
            CloseWindow()");
134     RunPopupB->Connect("Clicked()", "MtsPopup", this, "CloseWindow
```

```
          ()");
135     PolarPopupB->Connect("Clicked()", "MtsPopup", this, "
          CloseWindow()");
136   }
137   else {
138     //close button for normal error messages
139     ClosePopupB = new TGTextButton(VFrame1, "Close");
140     VFrame1->AddFrame(ClosePopupB, layoutNormal);
141     VFrame1->Resize(messageL->GetWidth() + 4,
142                     messageL->GetHeight() + ClosePopupB->GetHeight
                          () + 17);
143     PopupFrame->AddFrame(VFrame1, layoutFrame);
144     PopupFrame->Resize(VFrame1->GetWidth() + 4, VFrame1->GetHeight
          () + 6);
145   }
146
147   //connect closewindow to the close button
148   ClosePopupB->Connect("Clicked()", "MtsPopup", this, "CloseWindow
        ()");
149   //connect closewindow to the x in the upper right corner
150   PopupFrame->Connect("CloseWindow()", "MtsPopup", this, "
        CloseWindow()");
151   PopupFrame->MapSubwindows();
152
153   PopupFrame->MapWindow();
154 }
155
156
157
158 //!Copy data files from the raspberry pi to your notebook
159 /*!
160  * Specified runs or all data will be copied via scp.
161 */
162 void MtsPopup::Copy()
163 {
164   //copy files from the pi to the laptop
165   const char *copyChar = RemoveT->GetText();
166   Int_t runMin, runMax, nTest;
167   Int_t runNumber[10];
168   //first option: type in all and every run will be copied
169   if (strcmp(copyChar, "all") == 0 ||
170       strcmp(copyChar, "All") == 0 ||
171       strcmp(copyChar, "ALL") == 0) {
172     sprintf(Command, "scp␣pi@%s:%s*␣Measurements/␣",
173             mtsMain->ipAddressString, mtsMain->filename);
174     returnCode = system(Command);
175   }
176   //second option: first_run - last_run to copy a range of runs
177   else if (strchr(copyChar, '-') != NULL) {
178     nTest = sscanf(copyChar, "%d␣%s␣%d", &runMin, buffer, &runMax)
          ;
179     if (nTest == 3) {
180       for (Int_t i = 0; i < (runMax - runMin + 1); i++) {
181         sprintf(Command, "scp␣pi@%s:%s%d.*␣Measurements/",
182                 mtsMain->ipAddressString, mtsMain->filename,
```

```
                    runMin+i);
183           returnCode = system(Command);
184         }
185       }
186       else
187         new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
188                      "No run numbers found. Check your input.\n"
189                      "Possibilities:\n- all to delete all runs\n"
190                      "- lowestrun-highest_run to delete a range of
                             runs\n"
191                      "- run numbers separated by spaces (max 10)");
192     }
193     //last option copy runs by their numbers, limited to 10 runs
194     else {
195       nTest = sscanf(copyChar, "%d %d %d %d %d %d %d %d %d %d %d",
196                      &runNumber[0], &runNumber[1], &runNumber[2], &
                           runNumber[3],
197                      &runNumber[4], &runNumber[5], &runNumber[6], &
                           runNumber[7],
198                      &runNumber[8], &runNumber[9], &runNumber[10]);
199       if (nTest <= 0) {
200         new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
201                      "No run numbers found. Check your input.\n"
202                      "Possibilities:\nall to copy all runs\n"
203                      "lowestrun - highest_run to copy a range of
                             runs\n"
204                      "run numbers separated by spaces (max 10)");
205         return;
206       }
207       else if (nTest > 10)
208         new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
209                      "10 is the maximum number of runs, that can be
                             copied."
210                      " First 10 will be copied.\n"
211                      "If you want to copy more than that write all"
212                      " or lowest_run - highest_run");
213       else {
214         for (Int_t i = 0; i < nTest; i++) {
215           sprintf(Command, "scp pi@%s:%s%d.* Measurements/",
216                   mtsMain->ipAddressString, mtsMain->filename,
                        runNumber[i]);
217           returnCode = system(Command);
218         }
219       }
220     }
221     delete this;
222   }
223
224
225
226   //!Remove data files from the raspberry pi
227   /*!
228    * Specified runs or all data will be removed.
229   */
230   void MtsPopup::Remove()
```

```
231 {
232   //see MtsPopup::Copy() for comments, follows the same procedure
233   const char *removeChar = RemoveT->GetText();
234   Int_t runMin, runMax, nTest;
235   Int_t runNumber[10];
236   if (strcmp(removeChar, "all") == 0 ||
237       strcmp(removeChar, "All") == 0 ||
238       strcmp(removeChar, "ALL") == 0) {
239     sprintf(Command, "ssh pi@%s 'rm -f %s*'", mtsMain->
          ipAddressString,
240               mtsMain->filename);
241     returnCode = system(Command);
242   }
243   else if (strchr(removeChar, '-') != NULL) {
244     nTest = sscanf(removeChar, "%d %s %d", &runMin, buffer, &
          runMax);
245   if (nTest == 3) {
246     for (Int_t i = 0; i < (runMax - runMin + 1); i++) {
247       sprintf(Command, "ssh pi@%s 'rm -f %s%d.*'", mtsMain->
            ipAddressString,
248                 mtsMain->filename, runMin + i);
249       returnCode = system(Command);
250     }
251   }
252   else
253     new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
254                  "No run numbers found. Check your input.\n"
255                  "Possibilities:\n- all to delete all runs\n"
256                  "- lowestrun-highest_run to delete a range of
                       runs\n"
257                  "- run numbers separated by spaces (max 10)");
258   }
259   else {
260     nTest = sscanf(removeChar, "%d %d %d %d %d %d %d %d %d %d",
261                    &runNumber[0], &runNumber[1], &runNumber[2],
262                    &runNumber[3], &runNumber[4], &runNumber[5],
263                    &runNumber[6], &runNumber[7], &runNumber[8],
264                    &runNumber[9], &runNumber[10]);
265     if (nTest <= 0) {
266       new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
267                    "No run numbers found. Check your input.\n"
268                    "Possibilities:\nall to delete all runs\n"
269                    "lowestrun - highest_run to delete a range of
                         runs\n"
270                    "run numbers separated by spaces (max 10)");
271       return;
272     }
273     else if (nTest > 10)
274       new MtsPopup(gClient->GetRoot(), PopupFrame, 5, 5,
275                    "10 is the maximum number of runs, that can be
                         deleted."
276                    " First 10 will be deleted.\n"
277                    "If you want to delete more than that, write
                         all"
278                    " or lowest_run - highest_run");
```

```
279       else {
280         for (Int_t i = 0; i < nTest; i++) {
281           sprintf(Command, "ssh␣pi@%s␣'rm␣%s%d.*'", mtsMain->
                ipAddressString,
282                 mtsMain->filename, runNumber[i]);
283           returnCode = system(Command);
284         }
285       }
286     }
287     delete this;
288   }
289
290
291
292   //!Computes a polar plot of the muon flux taking the results of
        the run analysis
293   void MtsPopup::PolarAnalysis()
294   {
295     mtsMain->runToAnalyze = (Int_t) mtsMain->ProcessRunNumberT->
          GetIntNumber();
296     sprintf(buffer, "Results/Mts4Run%d.txt", mtsMain->runToAnalyze);
297     FILE* run = fopen(buffer, "r");
298     if (run == NULL) {
299       new MtsPopup(gClient->GetRoot(), mtsMain->MainFrame, 5, 5,
300                    "Please␣do␣run␣analysis␣first.");
301       return;
302     }
303     double dummy;
304     returnCode = fscanf(run, "%s␣%d", buffer, &mtsMain->runToAnalyze
          );
305     returnCode = fscanf(run, "%s␣%lf", buffer, &dummy);
306     mtsMain->time = (Double_t) dummy;
307     fclose(run);
308     //uses the output of MtsPopup::RunAnalysis()
309     //to generate a polar plot of the flux
310
311       //check if Other Rotation box is checked
312     if (mtsMain->OtherRotationB->IsOn()) {
313       new RotationPopup(gClient->GetRoot(), mtsMain->MainFrame, 5,
            5);
314     }
315     //else use the values given in the interface
316     else {
317       mtsMain->polarAngleDegree = mtsMain->PolarAngleT->GetNumber();
318       mtsMain->azimuthalAngleDegree = mtsMain->AzimuthalAngleT->
            GetNumber();
319       mtsMain->polarRotation = Y_ROT;
320       mtsMain->azimuthalRotation = Z_ROT;
321
322       //call analysis procedure
323       mtsMain->AnalyzePolar();
324     }
325   }
326
327
```

```
328
329 //!Calculates the slope of muon tracks from the measurement data
330 void MtsPopup::RunAnalysis()
331 {
332   //first check for run on laptop, then on raspberry,
333   //error if both empty, copy it to laptop if only on raspberry
334   mtsMain->runToAnalyze = (Int_t) mtsMain->ProcessRunNumberT->
        GetIntNumber();
335   sprintf(Command, "ls␣Measurements/Mts4Run%d.ebe␣>␣/dev/null",
336           mtsMain->runToAnalyze);
337   returnCode = system(Command);
338   if (returnCode != 0) {
339     sprintf(Command,"ssh␣pi@%s␣'ls␣%s%d.ebe␣>␣/dev/null'␣",
340             mtsMain->ipAddressString, mtsMain->filename, mtsMain->
                runToAnalyze);
341     returnCode = system(Command);
342     if (returnCode != 0) {
343       new MtsPopup(gClient->GetRoot(), mtsMain->MainFrame, 5, 5,
344                    "Run␣does␣not␣exist.");
345       return;
346     }
347     else {
348       sprintf(Command,"scp␣pi@%s:Measurements/Mts4Run%d.*␣
            Measurements/"
349               ">␣/dev/null", mtsMain->ipAddressString, mtsMain->
                  runToAnalyze);
350     }
351   }
352   mtsMain->nTimeBins = mtsMain->TimebinT->GetIntNumber();
353   analysis(mtsMain->runToAnalyze, mtsMain->nTimeBins, mtsMain->
        ChamberNumber);
354 }
355
356
357 //!First runs RunAnalysis(), then PolarAnalysis()
358 void MtsPopup::FullAnalysis()
359 {
360   RunAnalysis();
361   PolarAnalysis();
362 }
363
364 //Deconstructor and functions to close popup windows
365 void MtsPopup::CloseWindow()
366 {
367   delete this;
368 }
369
370 MtsPopup::~MtsPopup()
371 {
372   PopupFrame->DeleteWindow();
373 }
374
375 //~ void MtsPopup::CloseApp()
376 //~ {
377   //~ gApplication->Terminate();
```

```
378 //~ }
379
380
381 //! Popup window to handle the Other Rotation Checkbox
382 RotationPopup::RotationPopup(const TGWindow *p, const TGWindow *
        main,
383                              UInt_t w, UInt_t h, UInt_t options)
384 {
385   TGLayoutHints* layoutFrame = new TGLayoutHints(kLHintsTop|
        kLHintsExpandX |
386                                                  kLHintsExpandY);
387   TGLayoutHints* layoutGroup = new TGLayoutHints(kLHintsCenterX,
        10, 10, 2, 2);
388   TGLayoutHints* layoutRadio = new TGLayoutHints(kLHintsCenterX |
        kLHintsExpandX
389                                                  | kLHintsExpandY,
                                                       2, 2, 2, 2);
390   TGLayoutHints* layoutText = new TGLayoutHints(kLHintsLeft |
        kLHintsCenterY,
391                                                 2, 2, 2, 2);
392   //Popup for custom user rotation, if checkbox is checked
393   RotationFrame = new TGTransientFrame(p, main, w, h, options);
394   RotationFrame->SetName("RotationFrame");
395   //RotationFrame->SetLayoutBroken(kTRUE);
396   RotationFrame->SetWindowName("Mts4GUI");
397   RotationFrame->CenterOnParent();
398   RotationFrame->SetCleanup(kDeepCleanup);
399   TGVerticalFrame *VFrame1 = new TGVerticalFrame(RotationFrame,1,
        1);
400
401   TGHorizontalFrame *HFrame1 = new TGHorizontalFrame(VFrame1, 1,
        1);
402
403   //radio boxes to determine axis of rotation
404   TGHButtonGroup *radio1 = new TGHButtonGroup(HFrame1, "First␣
        Rotation");
405   x1B = new TGRadioButton(radio1,"x");
406   y1B = new TGRadioButton(radio1,"y");
407   z1B = new TGRadioButton(radio1,"z");
408   radio1->SetExclusive();
409   radio1->SetLayoutHints(layoutRadio);
410   radio1->Show();
411   HFrame1->AddFrame(radio1, layoutFrame);
412   //numberentry for input of rotation angle
413   FirstAngleT = new TGNumberEntry(HFrame1, 0, 6, -1,
        TGNumberFormat::kNESReal,
414                                   TGNumberFormat::kNEAAnyNumber);
415   HFrame1->AddFrame(FirstAngleT, layoutText);
416   VFrame1->AddFrame(HFrame1, layoutFrame);
417
418   TGHorizontalFrame *HFrame2 = new TGHorizontalFrame(VFrame1, 1,
        1);
419
420   //same for the second rotation
421   TGHButtonGroup *radio2 = new TGHButtonGroup(HFrame2, "Second␣
```

```
      Rotation");
422   x2B = new TGRadioButton(radio2,"x");
423   y2B = new TGRadioButton(radio2,"y");
424   z2B = new TGRadioButton(radio2,"z");
425   radio2->SetExclusive();
426   radio2->SetLayoutHints(layoutRadio);
427   radio2->Show();
428   HFrame2->AddFrame(radio2, layoutText);
429   SecondAngleT = new TGNumberEntry(HFrame2, 0,6,-1,TGNumberFormat
      ::kNESReal,
430                                    TGNumberFormat::kNEAAnyNumber);
431   HFrame2->AddFrame(SecondAngleT, layoutText);
432   HFrame2->Resize(radio2->GetWidth() + SecondAngleT->GetWidth() +
      80,
433                   SecondAngleT->GetHeight());
434   VFrame1->AddFrame(HFrame2, layoutFrame);
435
436   //buttons for start of analysis or closing without doing the
      analysis
437   TGHButtonGroup *fButtonGroup = new TGHButtonGroup(VFrame1, "");
438   OkRotationB = new TGTextButton(fButtonGroup,"␣Ok␣");
439   CloseRotationB = new TGTextButton(fButtonGroup,"Close");
440   fButtonGroup->SetLayoutHints(layoutGroup);
441   fButtonGroup->Show();
442   VFrame1->AddFrame(fButtonGroup, new TGLayoutHints(kLHintsTop |
      kLHintsCenterX
443                                                     |
                                                        kLHintsExpandX
                                                        , 0,0,0,0))
                                                        ;
444   VFrame1->Resize(200, 200);
445
446   RotationFrame->AddFrame(VFrame1, layoutFrame);
447   RotationFrame->Resize(VFrame1->GetWidth() + 4, VFrame1->
      GetHeight() + 6);
448
449   //connect functions to buttons
450   CloseRotationB->Connect("Clicked()", "RotationPopup", this, "
      CloseWindow()");
451   OkRotationB->Connect("Clicked()", "RotationPopup", this, "Ok()")
      ;
452   RotationFrame->Connect("CloseWindow()", "RotationPopup", this,
453                          "CloseWindow()");
454   RotationFrame->MapSubwindows();
455
456   RotationFrame->MapWindow();
457
458   Int_t returnCode;
459   int scanInt;
460   double scanFloat;
461   Char_t buffer[100];
462   FILE* RotationFile = fopen("temp/OtherRotation.temp", "r");
463   if (RotationFile == NULL) {
464     return;
465   }
```

```
466   returnCode = fscanf(RotationFile, "%s␣%d", buffer, &scanInt);
467   if (returnCode == 2) {
468     if (scanInt == 0) x1B->SetState(kButtonDown);
469     if (scanInt == 1) y1B->SetState(kButtonDown);
470     if (scanInt == 2) z1B->SetState(kButtonDown);
471   }
472   returnCode = fscanf(RotationFile, "%s␣%lf", buffer, &scanFloat);
473   FirstAngleT->SetNumber((Double_t) scanFloat);
474   returnCode = fscanf(RotationFile, "%s␣%d", buffer, &scanInt);
475   if (returnCode == 2) {
476     if (scanInt == 0) x2B->SetState(kButtonDown);
477     if (scanInt == 1) y2B->SetState(kButtonDown);
478     if (scanInt == 2) z2B->SetState(kButtonDown);
479   }
480   returnCode = fscanf(RotationFile, "%s␣%lf", buffer, &scanFloat);
481   SecondAngleT->SetNumber((Double_t) scanFloat);
482   fclose(RotationFile);
483 }
484
485
486 //!passes the newly specified rotation angles and axes to
487 void RotationPopup::Ok()
488 {
489   //get angles and rotations axes
490   mtsMain->polarAngleDegree = FirstAngleT->GetNumber();
491   mtsMain->azimuthalAngleDegree = SecondAngleT->GetNumber();
492   if (x1B->IsOn()) mtsMain->polarRotation = X_ROT;
493   if (y1B->IsOn()) mtsMain->polarRotation = Y_ROT;
494   if (z1B->IsOn()) mtsMain->polarRotation = Z_ROT;
495   if (x2B->IsOn()) mtsMain->azimuthalRotation = X_ROT;
496   if (y2B->IsOn()) mtsMain->azimuthalRotation = Y_ROT;
497   if (z2B->IsOn()) mtsMain->azimuthalRotation = Z_ROT;
498
499   //start analysis
500   mtsMain->AnalyzePolar();
501   delete this;
502 }
503
504 void RotationPopup::CloseWindow()
505 {
506   delete this;
507 }
508
509 RotationPopup::~RotationPopup()
510 {
511   ofstream rotation("temp/OtherRotation.temp");
512   if (!rotation.good()) {
513     return;
514   }
515   if (x1B->IsOn()) rotation << "FirstRotationAxis␣" << 0 << endl;
516   if (y1B->IsOn()) rotation << "FirstRotationAxis␣" << 1 << endl;
517   if (z1B->IsOn()) rotation << "FirstRotationAxis␣" << 2 << endl;
518   rotation << "FirstAngle␣" << FirstAngleT->GetNumber() << endl;
519   if (x2B->IsOn()) rotation << "SecondRotationAxis␣" << 0 << endl;
520   if (y2B->IsOn()) rotation << "SecondRotationAxis␣" << 1 << endl;
```

```
521   if (z2B->IsOn()) rotation << "SecondRotationAxis " << 2 << endl;
522   rotation << "SecondAngle " << SecondAngleT->GetNumber() << endl;
523   rotation.close();
524   RotationFrame->DeleteWindow();
525 }
526
527
528
529 //!Starts the measurement on the raspberry pi
530 /*!Creates all necessary files (run settings, run number, ...),
      copies them to
531  the raspberry and executes the measurement program on it
532  */
533 void MtsMain::StartRun()
534 {
535   //Create ini and settings files
536   // - RunNumber
537   sprintf(Command,"echo %d > ini/RunNumber.ini",
538           (Int_t) RunNumberT->GetIntNumber());
539   returnCode = system(Command);
540   // - Ini file
541   FILE * IniFile = fopen("ini/ReadoutSettings.ini", "w");
542   if(IniFile == NULL){
543     new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
544                  "Cannot write into the file 'ini/ReadoutSettings.
                        ini'\n"
545                  "Check attributes\nExit");
546     return;
547   }
548   nUsedChambers = 0;
549   for(Int_t N = 0; N < N_CHAMBERS_MAX; N++)
550     if (usedChambers[N] == true)
551       nUsedChambers++;
552   fprintf(IniFile, "NChambers %d\n", nUsedChambers);
553   fprintf(IniFile, "Channels ");
554   for(Int_t N = 0; N < N_CHAMBERS_MAX; N++) {
555     if (usedChambers[N] == true)
556       fprintf(IniFile, " %d", N);
557   }
558   fprintf(IniFile,"\n");
559   fprintf(IniFile, "ChamberSize %d %d\n", chamberSizeX,
        chamberSizeY);
560   fprintf(IniFile, "Statistics %d\n", (Int_t) StatisticsT->
        GetIntNumber());
561   fprintf(IniFile, "NAdcVAlues %d\n", nAdcChannels);
562   if(reorderBitsForMt == true)
563     fprintf(IniFile, "ReorderBits Yes\n");
564   else
565     fprintf(IniFile, "ReorderBits No\n");
566   fprintf(IniFile, "Prevalence %d\n", prevalence);
567   if(wtsBreakLine > 0)
568     fprintf(IniFile, "WtsBreakLine %d\n", wtsBreakLine);
569   fclose(IniFile);
570
571   // - Sett file
```

```
572   FILE* SettFile = fopen("ini/SettFile.ini", "w");
573   if (SettFile == NULL) {
574     new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
575                  "Cannot write into the file 'ini/SettFile.ini'\n"
576                  "Check attributes\nExit");
577     return;
578   }
579   fprintf(SettFile, "IpAddress %s\n", ipAddressString);
580   for (Int_t i = 0; i < nUsedChambers; i++) {
581     fprintf(SettFile, "Chamber %d PositionZ %d\n", ChamberNumber[i
          ],
582             chamberPosition[i]);
583     //read from file
584   }
585   fprintf(SettFile, "Gas %s\n", UsedGasT->GetText());
586   fprintf(SettFile, "HV %d %d \n", (Int_t) HV_SenseWireT->
          GetIntNumber(),
587           (Int_t) HV_SW_CurrentT->GetIntNumber());
588   fprintf(SettFile, "Notes \n");
589   mtsMain->AdditionalInformationT->SaveFile("temp/
          additionalinformation.temp");
590   FILE* AdditionalInfo = fopen("temp/additionalinformation.temp",
          "r");
591   if (AdditionalInfo == NULL) {
592     new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
593                  "Cannot read the file 'temp/additionalinformation
                       .temp'\n"
594                  "Check attributes\nExit");
595     return;
596   }
597   while (fgets(buffer, 100, AdditionalInfo)) {
598     fputs(buffer, SettFile);
599   }
600   fclose(AdditionalInfo);
601   fclose(SettFile);
602
603   // Start measurement
604   sprintf(Command, "scp ini/*.ini pi@%s:MtRpiDaq/. > /dev/null",
605           mtsMain->ipAddressString);
606   returnCode = system(Command);
607   sprintf(Command, "ssh pi@%s 'cd MtRpiDaq;"
608           "nohup sudo ./MtRpiDaq.run -f &> StdDump &' ",
609           mtsMain->ipAddressString);
610   returnCode = system(Command);
611   Update();
612 }
613
614
615 //!Restart Update Timer after the interval in the GUI got changed
616 void MtsMain::UpdateIntervalEdit(Long_t val)
617 {
618   //change update interval, when it is changed in the ui and
619   //restart update timer with new interval
620   timer->Stop();
621   if (UpdateIntervalT->GetIntNumber() > 0)
```

```
622     timer->Start(UpdateIntervalT->GetIntNumber() * 60 * 1000);
623 }
624
625
626 //!Load old Settings and Parameters on Startup
627 void MtsMain::Startup()
628 {
629   //things to do on starting the app
630   // Check IpAddress
631   sprintf(Command,"ssh␣pi@%s␣'ls␣&>␣/dev/null'␣",ipAddressString);
632   returnCode = system(Command);
633   //error message if there's no connection to the pi
634   if (returnCode != 0) {
635     new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "RPi␣cannot␣
          be␣reached␣\n"
636                  "Check␣internet␣connection␣on␣both␣devices\n"
637                  "\nonly␣analysis␣options␣available");
638     LoadFormerSettings();
639     Update();
640     return;
641   }
642   else {
643     isConnected = true;
644     // Check existance of the directory: ../Measurements
645     sprintf(Command,"ls␣Measurements␣&>␣/dev/null");
646     returnCode = system(Command);
647     if (returnCode != 0) {
648       sprintf(Command, "mkdir␣/Measurements");
649       returnCode = system(Command);
650       if (returnCode != 0) {
651         new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
652                      "Directory␣Measurements␣couldn't␣be␣created\n
                         "
653                      "Check␣attributes␣or␣create␣the␣necessary␣
                         directory");
654       }
655     }
656   }
657   LoadFormerSettings();
658   Update();
659 }
660
661
662 //!Start measurement
663 /*!Verifies connection to the raspberry and non-existence of the
      specified run.
664  * After that the measurement is started with the StartRun()
       function
665  */
666 void MtsMain::Start()
667 {
668   if (!isConnected) {
669     new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "Please␣fix␣
          connection␣to"
670                  "␣the␣Muon␣Tomograph␣first.");
```

```
671     }
672     else {
673       // Check status
674       sprintf(Command,"ssh␣pi@%s␣'cat␣MtRpiDaq/Running␣&>␣/dev/null'
          ␣",
675               ipAddressString);
676       returnCode = system(Command);
677       if (returnCode == 0) {
678         new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "MT␣is␣
            running!␣\n"
679                      "Stop␣it␣before␣starting␣a␣new␣run");
680         return;
681       }
682       // Check RUN existance
683       sprintf(Command,"ssh␣pi@%s␣'head␣-n␣1␣%s%d.ebe␣&>␣/dev/null'␣"
          ,
684               ipAddressString,filename,runNumberInt);
685       returnCode = system(Command);
686       if (returnCode == 0)
687         new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,"Run␣
            already␣exists␣\n"
688                      "Would␣you␣like␣to␣overwrite␣it?", 2);
689       else
690         StartRun();
691     }
692   }
693
694
695   //!Stops the measurement
696   void MtsMain::Stop()
697   {
698     if (!isConnected) {
699       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "Please␣fix␣
            connection␣to"
700                    "the␣Muon␣Tomograph␣first.");
701     }
702     else {
703     //stop the run and update the status labels, also increase the
          runNumber
704       sprintf(Command,"ssh␣pi@%s␣'sudo␣rm␣-f␣MtRpiDaq/Running␣␣2>␣/
            dev/null␣;"
705               "sudo␣killall␣MtRpiDaq.run␣2>␣/dev/null␣'␣",
706                  ipAddressString);
707       returnCode = system(Command);
708       Update();
709       RunNumberT->IncreaseNumber();
710       runNumberInt = RunNumberT->GetIntNumber();
711     }
712   }
713
714   //!Update run status labels with current measurement status
715   void MtsMain::Update()
716   {
717     ipAddressString = IpAddressT->GetText();
```

```
718   //TString ip = IpAddressT->GetDisplayText();
719   //IpAddressString = (const char*) ip;
720   cerr << ipAddressString << endl;
721   sprintf(Command,"ssh␣pi@%s␣'ls␣&>␣/dev/null'␣",ipAddressString);
722   returnCode = system(Command);
723   //error message if there's no connection to the pi
724   if (returnCode == 0) {
725     isConnected = true;
726   }
727   else {
728     StatusL->SetText("The␣MuonTomograph␣is␣not␣connected.");
729     EventsL->SetText("Press␣Update␣after␣fixing␣connection.");
730   }
731   if (isConnected) {
732     sprintf(Command,"ssh␣pi@%s␣'cat␣MtRpiDaq/Running␣&>␣/dev/null␣
          '␣",
733             ipAddressString);
734     returnCode = system(Command);
735     //update status label with current status
736     if (returnCode != 0) StatusL->SetText("The␣MT␣is␣ready␣to␣take
          ␣data");
737     else StatusL->SetText("The␣MuonTomography␣setup␣is␣running");
738     //StatusL->DoRedraw();
739
740     sprintf(Command,"ssh␣pi@%s␣'tail␣-n␣4␣%s%d.ebe␣2>␣/dev/null'"
741             "|␣cat␣>␣temp/Status.temp␣", ipAddressString, filename
                ,
742             runNumberInt);
743     returnCode = system(Command);
744     FILE * TempFile = fopen("temp/Status.temp", "r");
745     if (TempFile == NULL) {
746       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
747                    "temp/Status.temp␣file␣cannot␣be␣opened!");
748       return;
749     }
750     Int_t CollectedEvents = 0;
751     //update event label with number of collected events
752     if (fscanf(TempFile, "%d", &CollectedEvents) == 1) {
753       CollectedEvents += 4;
754       sprintf(buffer, "There␣are␣%d␣events␣in␣Run%d",
             CollectedEvents,
755                  runNumberInt);
756       EventsL->SetText(buffer);
757       //EventsL->DoRedraw();
758       sprintf(Command,"head␣-n␣3␣temp/Status.temp␣|␣awk␣-f␣display
             .awk␣");
759       returnCode = system(Command);
760     }
761     else {
762       sprintf(buffer, "Run%d␣does␣not␣exist", runNumberInt);
763       EventsL->SetText(buffer);
764       //EventsL->DoRedraw();
765     }
766     fclose(TempFile);
767     //restart timer
```

```
768       timer->TurnOff();
769       timer->Start(UpdateIntervalT->GetIntNumber() * 60 * 1000);
770     }
771   }
772
773
774   //!Change IP address, if changed in the GUI
775   void MtsMain::TextEdit(char *val){
776     //change ipaddress, when it is changed in the user interface
777     ipAddressString = val;
778   }
779
780
781   //!Load former settings from .ini files
782   void MtsMain::LoadFormerSettings()
783   {
784     //copy old ini files from pi
785     if (isConnected) {
786       sprintf(Command, "scp␣pi@%s:MtRpiDaq/*.ini␣ini/␣>␣/dev/null",
787               ipAddressString);
788       returnCode = system(Command);
789     }
790
791     Int_t Value1, Value2, oldUsedChamber[N_CHAMBERS_MAX];
792     for (Int_t i = 0; i < N_CHAMBERS_MAX; i++)
793       usedChambers[i] = false;
794
795     // Get the current run number
796     FILE* RunNumberFile = fopen("ini/RunNumber.ini", "r");
797     if (RunNumberFile != NULL) {
798       returnCode = fscanf(RunNumberFile, "%d", &runNumberInt);
799       fclose(RunNumberFile);
800       RunNumberT->SetIntNumber(runNumberInt);
801     }
802
803     // Load former readout settings
804     // maybe easier?
805     FILE* IniFile = fopen("ini/ReadoutSettings.ini", "r");
806     if (IniFile == NULL)
807       return;
808     while (!feof(IniFile)) {
809       if (fgets(Line, 198, IniFile) == NULL) {
810         continue;
811       }
812       sscanf(Line,"%s",ValueName);
813       if (strcmp(ValueName, "NChambers") == 0) {
814         sscanf(Line,"%s␣%d",ValueName,&nChamber);
815       }
816       else if (strcmp(ValueName, "Statistics") == 0) {
817         sscanf(Line,"%s␣%d",ValueName,&Value1);
818         StatisticsT->SetIntNumber(Value1);
819       }
820       else if (strcmp(ValueName, "NAdcValues") == 0) {
821         sscanf(Line,"%s␣%d",ValueName,&Value1); nAdcChannels =
              Value1;
```

```
822       }
823       else if (strcmp(ValueName, "Prevalence") == 0) {
824         sscanf(Line,"%s %d",ValueName,&prevalence);
825       }
826       else if (strcmp(ValueName, "WtsBreakLine") == 0) {
827         sscanf(Line,"%s %d",ValueName,&wtsBreakLine);
828       }
829       else if (strcmp(ValueName, "ReorderBits") == 0) {
830         sscanf(Line,"%s %s",ValueName,ValueName);
831         if (strcmp(ValueName, "Y") == 0 || strcmp(ValueName, "y") ==
              0 ||
832             strcmp(ValueName, "yes") == 0 || strcmp(ValueName, "Yes"
                ) == 0 ||
833             strcmp(ValueName, "YES") == 0  || strcmp(ValueName, "1")
                 == 0 )
834           reorderBitsForMt = true;
835         else reorderBitsForMt = false;
836       }
837       else if (strcmp(ValueName, "ChamberSize") == 0) {
838         sscanf(Line, "%s %d %d", ValueName, &Value1, &Value2);
839         chamberSizeX = Value1;
840         chamberSizeY = Value2;
841       }
842       else if (strcmp(ValueName, "Channels") == 0) {
843         nUsedChambers = sscanf(Line, "%s %d %d %d %d %d %d %d %d %d %
              d",
844                                ValueName, &oldUsedChamber[0], &
                                   oldUsedChamber[1],
845                                &oldUsedChamber[2], &oldUsedChamber
                                   [3],
846                                &oldUsedChamber[4], &oldUsedChamber
                                   [5],
847                                &oldUsedChamber[6], &oldUsedChamber
                                   [7],
848                                &oldUsedChamber[8], &oldUsedChamber
                                   [9]);
849         if (nChamber != nUsedChambers-1) {
850           new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
851                        "Minor inconsistency in the GUI settings file
                           .");
852         }
853         for (Int_t i = 0; i < nChamber; i++) {
854           if (oldUsedChamber[i] < N_CHAMBERS_MAX)
855             usedChambers[oldUsedChamber[i]] = true;
856         }
857       }
858     }
859     fclose(IniFile);
860
861     // Get the filenaming settings
862     FILE* FilenamingFile = fopen("ini/Filenaming.ini", "r");
863     if (FilenamingFile == NULL) {
864       strcpy(filename, "/home/pi/Measurements/Mts4Run");
865       return;
866     }
```

```
867   returnCode = fscanf(FilenamingFile, "%s", buffer);
868   if (returnCode != 1) {
869     strcpy(filename, "/home/pi/Measurements/Mts4Run");
870     return;
871   }
872
873   fclose(FilenamingFile);
874   if (buffer[0] == '/')
875     strcpy(filename, buffer);
876   else {
877     strcpy(filename,"/home/pi/");
878     strcat(filename, buffer);
879   }
880
881   // Check the former run's settings
882   FILE* SettFile = fopen("ini/SettFile.ini", "r");
883   FILE* AddInfos = fopen("temp/additionalinformation.temp", "w");
884   if (SettFile == NULL)
885     return;
886   nUsedChambers = 0;
887   while (fgets(Line, 198, SettFile)) {
888     sscanf(Line, "%s", ValueName);
889     if (strcmp(ValueName, "Chamber") == 0) {
890       sscanf(Line, "%s␣%d␣%s␣%d", ValueName, &Value1, ValueName, &
          Value2);
891       if (nUsedChambers < nChamber) {
892         ChamberNumber[nUsedChambers] = Value1;
893         chamberPosition[nUsedChambers] = Value2;
894       }
895       nUsedChambers++;
896     }
897     else if (strcmp(ValueName, "HV") == 0) {
898       sscanf(Line, "%s␣%d␣%d", ValueName, &Value1, &Value2);
899       HV_SenseWireT->SetIntNumber(Value1);
900       HV_SW_CurrentT->SetIntNumber(Value2);
901     }
902     else if (strcmp(ValueName, "Gas") == 0) {
903       Line[strlen(Line)-1] = '\0';
904       UsedGasT->SetText(&Line[4]);
905     }
906     else if (strcmp(ValueName, "Notes") == 0);
907     else if (strcmp(ValueName, "IpAddress") == 0);
908     else
909       fputs(Line, AddInfos);
910   }
911   fclose(SettFile);
912   fclose(AddInfos);
913   AdditionalInformationT->LoadFile("temp/additionalinformation.
        temp");
914
915   FILE* OldSettings = fopen("temp/OldSettings.temp", "r");
916   if (OldSettings == NULL) {
917     return;
918   }
919   int scanInt;
```

```
920     float scanFloat;
921     returnCode = fscanf(OldSettings, "%s %d", buffer, &scanInt);
922     UpdateIntervalT->SetIntNumber(scanInt);
923     returnCode = fscanf(OldSettings, "%s %d", buffer, &scanInt);
924     ProcessRunNumberT->SetIntNumber(scanInt);
925     returnCode = fscanf(OldSettings, "%s %f", buffer, &scanFloat);
926     BinsizeThetaT->SetNumber(scanFloat);
927     returnCode = fscanf(OldSettings, "%s %f", buffer, &scanFloat);
928     BinsizePhiT->SetNumber(scanFloat);
929     returnCode = fscanf(OldSettings, "%s %d", buffer, &scanInt);
930     TimebinT->SetIntNumber(scanInt);
931     returnCode = fscanf(OldSettings, "%s %f", buffer, &scanFloat);
932     PolarAngleT->SetNumber(scanFloat);
933     returnCode = fscanf(OldSettings, "%s %f", buffer, &scanFloat);
934     AzimuthalAngleT->SetNumber(scanFloat);
935     returnCode = fscanf(OldSettings, "%s %d", buffer, &scanInt);
936     if (scanInt == 1) OtherRotationB->SetState(kButtonDown);
937     if (scanInt == 0) OtherRotationB->SetState(kButtonUp);
938     char runNumbers[100];
939     while (fgets(runNumbers, 98, OldSettings)) {
940       runNumbers[strlen(runNumbers)-1] = '\0';
941       RunNumbersT->SetText(&runNumbers[11]);
942     }
943     fclose(OldSettings);
944   }
945
946
947   //!Popup window for copy dialog
948   void MtsMain::CopyData()
949   {
950     if (!isConnected) {
951       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
952                    "Please fix connection to the Muon Tomograph
                          first.");
953     }
954     else
955       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
956                    "Which runs do you want to copy?\n"
957                    "Type all, if you want to copy all runs.", 4);
958   }
959
960
961   //!Popup window for remove dialog
962   void MtsMain::RemoveData()
963   {
964     if (!isConnected) {
965       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
966                    "Please fix connection to the Muon Tomograph
                          first.");
967     }
968     else
969       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
970                    "Which runs do you want to delete?\n"
971                    "Type all, if you want to delete all runs.", 3);
972   }
```

```
973
974
975  //!Popup window for analysis dialog
976  void MtsMain::Analyze()
977  {
978    dPhi = BinsizePhiT->GetNumber();
979    nBinPhi = (Int_t) ((phiMax - phiMin) / dPhi);
980    dTheta = BinsizeThetaT->GetNumber();
981    nBinTheta = (Int_t) ((thetaMax - thetaMin) / dTheta);
982    //cerr << dTheta << "\t" << nBinTheta << endl;
983    new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5,
984                 "Choose the type of analysis, that you want to
                        perform.", 5);
985  }
986
987
988  //!Show output of single file analysis
989  void MtsMain::SinglePlot()
990  {
991    sprintf(Command,"xpdf Results/histRun%d.pdf ",
992            (Int_t) ProcessRunNumberT->GetIntNumber());
993    returnCode = system(Command);
994  }
995
996
997  //!Weighted mean of polar histograms
998  /*!
999   * Takes the polar histograms calculated in PolarAnalysis() and
          performs a
1000  * weighted arithmetic mean of each bin. The result gets saved as
          .pdf
1001  */
1002 void MtsMain::GetWeightedHistogramMean()
1003 {
1004   const Int_t nMax = 10;
1005   Int_t runNumber[nMax+1];
1006   Int_t runMin, runMax;
1007   Int_t nHist;
1008   const char *runs_char = RunNumbersT->GetText();
1009   //get run numbers
1010   if (strchr(runs_char, '-') == NULL) {
1011     nHist = sscanf(runs_char, "%d %d %d %d %d %d %d %d %d %d %d",
1012                    &runNumber[0], &runNumber[1], &runNumber[2], &
                           runNumber[3],
1013                    &runNumber[4], &runNumber[5], &runNumber[6], &
                           runNumber[7],
1014                    &runNumber[8], &runNumber[9], &runNumber[10]);
1015     if (nHist > nMax) {
1016       sprintf(buffer, "too many runs, merge only first %d", nMax);
1017       new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1018     }
1019   }
1020   else {
1021     sscanf(runs_char, "%d %s %d", &runMin, buffer, &runMax);
1022     nHist = runMax - runMin + 1;
```

```
1023      if ((runMax - runMin + 1) > nMax) {
1024        sprintf(buffer, "too many runs, only runs %d to %d will be
              merged",
1025                runMin, runMin + nMax - 1);
1026        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1027      }
1028      for (Int_t i = 0; i < nHist; i++)
1029        runNumber[i] = runMin + i;
1030    }
1031
1032    Int_t nBinPhiAddition = -1;
1033    Int_t nBinThetaAddition = -1;
1034
1035    //Get bin setting from first run
1036    ostringstream txtStream;
1037    txtStream << "Results/Mts4Run" << runNumber[0] << ".txt";
1038    FILE* TxtFile = fopen(txtStream.str().c_str(), "r");
1039    if (TxtFile == NULL) {
1040      sprintf(buffer, "Mts4Run%d wasn't analyzed before or does not
            exist.",
1041              runNumber[0]);
1042      new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1043      return;
1044    }
1045    while (fgets(Line, 198, TxtFile)) {
1046      sscanf(Line, "%s", ValueName);
1047      if (strcmp(ValueName, "nBinPhi") == 0) {
1048        returnCode = sscanf(Line, "%s %d", buffer, &nBinPhiAddition)
              ;
1049      }
1050      if (strcmp(ValueName, "nBinTheta") == 0) {
1051        returnCode = sscanf(Line, "%s %d", buffer, &
              nBinThetaAddition);
1052      }
1053    }
1054    if (nBinPhiAddition < 1 || nBinThetaAddition < 1) {
1055      sprintf(buffer, "Check File %s for binning setting",
1056              txtStream.str().c_str());
1057      new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1058      return;
1059    }
1060    fclose(TxtFile);
1061
1062    cerr << nBinPhiAddition << " " << nBinThetaAddition << endl;
1063
1064    //array for theta bin edges, so theta = 0 is plotted
1065    Double_t* array = new Double_t[nBinThetaAddition+2];
1066
1067    array[0] = 0;
1068    array[1] = 0.000000001;
1069    for (Int_t i = 2; i < nBinThetaAddition+2; i++) {
1070      array[i] = (i - 1) * (thetaMax - thetaMin) / nBinThetaAddition
            ;
1071    }
1072
```

```
1073
1074    //histogram initialization
1075    TH2F *MergedHist = new TH2F("Flux␣Histogram", "Flux␣Histogram",
           nBinPhiAddition, phiMin,
1076                                phiMax, nBinThetaAddition+1, array);
1077    TH2F *dMergedHist = new TH2F("Flux␣error", "Flux␣error",
           nBinPhiAddition,
1078                                    phiMin, phiMax, nBinThetaAddition
                                       +1, array);
1079
1080
1081  TH2F *abc = new TH2F("Flux␣Histogram", "Flux␣Histogram",
        nBinPhiAddition, phiMin,
1082                                phiMax, nBinThetaAddition+1, array);
1083
1084    delete array;
1085    TH2F *Hists[nHist];
1086    TH2F *dHists[nHist];
1087    //get filenames for runs
1088    for (Int_t j = 0; j < nHist; j++) {
1089      ostringstream histogramNameStream;
1090      ostringstream histogramTxtStream;
1091      histogramNameStream << "Results/Mts4Run" << runNumber[j] << ".
           root";
1092      histogramTxtStream << "Results/Mts4Run" << runNumber[j] << ".
           txt";
1093
1094      //after the first run, check if binning is consistent
           throughout the runs
1095      if (j > 0) {
1096        Int_t nBinPhiTest, nBinThetaTest;
1097        TxtFile = fopen(histogramTxtStream.str().c_str(), "r");
1098        if (TxtFile == NULL) {
1099          sprintf(buffer, "Mts4Run%d␣wasn't␣analyzed␣before␣or␣does␣
               not␣exist.",
1100                  runNumber[0]);
1101          new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1102          return;
1103        }
1104        while (fgets(Line, 198, TxtFile)) {
1105          sscanf(Line, "%s", ValueName);
1106          if (strcmp(ValueName, "nBinPhi") == 0) {
1107            returnCode = sscanf(Line, "%s␣%d", buffer, &nBinPhiTest)
                 ;
1108          }
1109          if (strcmp(ValueName, "nBinTheta") == 0) {
1110            returnCode = sscanf(Line, "%s␣%d", buffer, &
                 nBinThetaTest);
1111          }
1112        }
1113        if (nBinPhiAddition != nBinPhiTest ||
1114            nBinThetaAddition != nBinThetaTest)
1115        {
1116          sprintf(buffer, "Reanalyze␣Run␣%d␣with␣Binsize␣Theta␣=␣%d␣
               and"
```

```
1117                  "␣Binsize␣Phi␣=␣%d", runNumber[j],
1118                  (Int_t) ((thetaMax - thetaMin) / nBinThetaAddition
                         ),
1119                  (Int_t) ((phiMax - phiMin) / nBinPhiAddition));
1120          new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1121          return;
1122        }
1123        fclose(TxtFile);
1124      }
1125      ////////////////////////////////
1126      //Read-In
1127      ////////////////////////////////
1128      //check if file exists
1129      TFile h(histogramNameStream.str().c_str());
1130      if (h.IsZombie() == true) {
1131        sprintf(buffer, "File␣%s␣does␣not␣exist,␣please␣analyze␣this
             ␣run␣first.",
1132                histogramNameStream.str().c_str());
1133        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1134        return;
1135      }
1136      //get flux and error histogram
1137      TH2F* TempHist = (TH2F*) h.Get(Form("hist␣%d", runNumber[j]));
1138      TH2F* dTempHist = (TH2F*) h.Get(Form("error␣%d", runNumber[j])
           );
1139      //change directory so TFile can be closed without deleting the
           histograms
1140      TempHist->SetDirectory(0);
1141      dTempHist->SetDirectory(0);
1142      h.Close();
1143      //put them into an array
1144      Hists[j] = TempHist;
1145      dHists[j] = dTempHist;
1146    }
1147    ofstream gnu("gnuplot.txt");
1148    //loop over ...
1149
1150    for (Int_t Ny = 1; Ny < nBinThetaAddition + 2; Ny++) { //...
         theta-bins
1151      for (Int_t Nx = 1; Nx < nBinPhiAddition + 1; Nx++) {  //...phi
           -bins
1152        Double_t add = 0, dAdd = -1.;
1153        for (Int_t j = 0; j < nHist; j++) {  //...histograms
1154          //if (Ny == 1)cerr << Nx << "\t" << Ny << "\t";
1155          if (dAdd < 0) {
1156            //first step: loop till there's a histogram with a hit
                 in this bin
1157            if (dHists[j]->GetBinContent(Nx,Ny) > pow(10,5));
1158              //if there is no entry in this bin skip the addition
1159            else {
1160              //else set add and dAdd to the respective bincontents
1161              add = Hists[j]->GetBinContent(Nx,Ny);
1162              dAdd = dHists[j]->GetBinContent(Nx,Ny);
1163              //cerr << Ny << " " << Nx << " " << j << " ";
1164              //cerr << add << " " << dAdd << " first" << endl;
```

```
1165            }
1166          }
1167          else {
1168            //now weighted addition of the histograms,
1169            //calculate new content of current bin and the relative
                  error
1170            //cerr << Ny << " " << Nx << " " << j << " " << Hists[j
                  ]->GetBinContent(Nx,Ny) << " ";
1171            //cerr << dHists[j]->GetBinContent(Nx,Ny) << endl;
1172            add = ( (add / pow(dAdd, 2) + (Hists[j]->GetBinContent(
                  Nx,Ny) /
1173                    pow(dHists[j]->GetBinContent(Nx,Ny),2) ) ) /
1174                    (1. / pow(dAdd,2) +
1175                     1. / pow(dHists[j]->GetBinContent(Nx,Ny),2) )
1176                 );
1177            dAdd = sqrt(1. / (1. / pow(dAdd, 2) + 1. /
1178                       pow(dHists[j]->GetBinContent(Nx,Ny), 2) ) );
1179            //cerr << "\t" << "\t" << add << " " << dAdd << endl;
1180          }
1181          if (j == nHist - 1) {
1182            //when on the last histogram, fill the result into a new
                   histogram
1183            MergedHist->SetBinContent(Nx, Ny, add);
1184            //cerr << (Nx-1) * 360./nBinPhiAddition << " " << add <<
                   " " << dAdd << endl;
1185            if (Nx == 8 && add > 0) gnu << (Ny-2) * 90./
                  nBinThetaAddition + dTheta/2. << "\t" << add  << "\t"
                   << dAdd * add << "\t" << dAdd << endl;
1186            else if (Nx == 17 && add > 0) gnu  << -(Ny-2) * 90./
                  nBinThetaAddition - dTheta/2. << "\t" << add  << "\t"
                   << dAdd * add << "\t" << dAdd<< endl;
1187            cerr << (Ny-2) * 90./nBinThetaAddition << "\t" << (Nx-1)
                   * 360./nBinPhiAddition << "\t" << add  << "\t" <<
                  dAdd << "\t" << MergedHist->GetBinContent(Nx, Ny) <<
                  endl;
1188            if (dAdd > 0.0 && dAdd < 0.1)
1189              dMergedHist->SetBinContent(Nx, Ny, dAdd);
1190            else dMergedHist->SetBinContent(Nx, Ny, 0);
1191          }
1192        }
1193      }
1194    }
1195
1196    Double_t chi;
1197    Bool_t isDone[nHist];
1198    for (Int_t i = 0; i < nHist; i++)
1199      isDone[i] = false;
1200    TH1F* test = new TH1F("","",40,-5,5);
1201
1202    for (Int_t i = 0; i < nHist; i++) {
1203      for (Int_t j = 0; j < nHist; j++) {
1204        if (dHists[i]->GetBinContent(2,2) < 10000 && dHists[j]->
              GetBinContent(2,2) < 10000 && i != j) {
1205          chi = (Hists[i]->GetBinContent(2,2) - Hists[j]->
                 GetBinContent(2,2))/
```

```
1206                sqrt(pow(dHists[i]->GetBinContent(2,2) * Hists[i]->
                       GetBinContent(2,2),2) + pow(Hists[j]->
                       GetBinContent(2,2) * dHists[j]->GetBinContent
                       (2,2),2));
1207          test->Fill(chi);
1208          if (chi > 1.4) {
1209           abc->Fill(5.0,2);
1210           //cerr << 0 << " " << 0 << endl;
1211          }
1212          //cerr << i << " " << j << " " << 2 << " " << 2 << " " <<
                 Hists[i]->GetBinContent(2,2) << " " << Hists[j]->
                 GetBinContent(2,2) << " " << sqrt(pow(dHists[i]->
                 GetBinContent(2,2) * Hists[i]->GetBinContent(2,2),2) +
                 pow(Hists[j]->GetBinContent(2,2) * dHists[j]->
                 GetBinContent(2,2),2)) << " " << dHists[i]->
                 GetBinContent(2,2) << " " << dHists[j]->GetBinContent
                 (2,2) <<  endl;
1213        }
1214        for (Int_t Ny = 3; Ny < nBinThetaAddition + 2; Ny++) { //...
               theta-bins
1215          for (Int_t Nx = 1; Nx < nBinPhiAddition + 1; Nx++) {  //
                 ...phi-bins
1216            if (isDone[j] == true)
1217              continue;
1218            if (dHists[i]->GetBinContent(Nx,Ny) < 10000 && dHists[j
                   ]->GetBinContent(Nx,Ny) < 10000 && i != j) {
1219              chi = (Hists[i]->GetBinContent(Nx,Ny) - Hists[j]->
                     GetBinContent(Nx,Ny))/
1220                  sqrt(pow(dHists[i]->GetBinContent(Nx,Ny) * Hists
                       [i]->GetBinContent(Nx,Ny),2) + pow(Hists[j]->
                       GetBinContent(Nx,Ny) * dHists[j]->
                       GetBinContent(Nx,Ny),2));
1221              test->Fill(chi);
1222             // cerr << i << " " << j << " " << Ny << " " << Nx << "
                     " << Hists[i]->GetBinContent(Nx,Ny) << " " << Hists
                     [j]->GetBinContent(Nx,Ny) << " " << sqrt(pow(dHists[
                     i]->GetBinContent(Nx,Ny) * Hists[i]->GetBinContent(
                     Nx,Ny),2) + pow(Hists[j]->GetBinContent(Nx,Ny) *
                     dHists[j]->GetBinContent(Nx,Ny),2))  << " " <<
                     dHists[i]->GetBinContent(Nx,Ny) << " " << dHists[j
                     ]->GetBinContent(Nx,Ny) <<  endl;
1223              if (chi > 1.4 || chi < -1.4) {
1224                abc->Fill((Nx-1)*360.0/nBinPhiAddition,(Ny-2)*90.0/
                       nBinThetaAddition);
1225                //cerr << (Nx-1)*360.0/nBinPhiAddition << " " << (Ny
                       -2)*90.0/nBinThetaAddition<< endl;
1226              }
1227            }
1228          }
1229        }
1230      }
1231      isDone[i] = true;
1232    }
1233    gStyle->SetOptStat(1);
1234    TCanvas* c5 = new TCanvas("cb", "Polar␣Flux", 0, 10, 1000, 1000)
```

```
        ;
1235    c5->Update();
1236    test->Draw("HIST");
1237    TF1* myfit = new TF1("myfit", "[0]*exp(-0.5*x*x/([1]*[1]))", 50,
            1);
1238    test->Fit("gaus");
1239
1240    TCanvas* c6 = new TCanvas("ca", "Polar␣Flux", 0, 10, 1000, 1000)
            ;
1241    abc->Draw("SURF1␣POL␣Z");
1242      ////------------------------------------------
1243      ////
1244      //// graphical output
1245      ////------------------------------------------
1246
1247    gStyle->SetHatchesSpacing(1000);
1248    gStyle->SetStripDecimals(kFALSE);
1249    gStyle->SetPadRightMargin(0.14);
1250    gStyle->SetPadLeftMargin(0.06);
1251    gStyle->SetPadTopMargin(0.12);
1252    gStyle->SetPadBottomMargin(0.10);
1253    gStyle->SetOptTitle(0);
1254    gStyle->SetOptStat(1);
1255    TLatex Tl;
1256    Double_t red[9] = {0./255., 61./255., 89./255., 122./255.,
            143./255.,
1257                       160./255., 185./255., 204./255., 231./255.};
1258    Double_t green[9] = {0./255., 0./255., 0./255., 0./255.,
            14./255., 37./255.,
1259                         72./255., 132./255., 235./255.};
1260    Double_t blue[9] = {0./255., 140./255., 224./255., 144./255.,
            4./255.,
1261                        5./255., 6./255., 9./255., 13./255.};
1262    Double_t stops[9] = {0.0000, 0.1250, 0.2500, 0.3750, 0.5000,
            0.6250,
1263                         0.7500, 0.8750, 1.0000};
1264    MergedHist->SetContour(99);
1265    dMergedHist->SetContour(99);
1266    TColor::CreateGradientColorTable(9, stops, red, green, blue,
            255);
1267
1268    //plot of relative flux error
1269    TCanvas* c1 = new TCanvas("c1", "Polar␣Flux", 0, 10, 1000, 1000)
            ;
1270    c1->SetFillStyle(3305);
1271    c1->SetFillColor(kBlack);
1272    c1->SetTheta(90);
1273    c1->SetPhi(0);
1274    dMergedHist->Draw("SURF1␣POL␣Z");
1275    c1->Update();
1276    TPaletteAxis* palette = (TPaletteAxis*)dMergedHist->
            GetListOfFunctions()->FindObject("palette");
1277    //dMergedHist->GetZaxis()->SetTickSize(0.025);
1278    palette->SetLabelSize(0.03);
1279    palette->SetX1NDC(0.92);
```

```
1280    palette->SetX2NDC(0.943);
1281    Tl.SetTextFont(43); Tl.SetTextSize(24); //font
1282    Tl.SetNDC(); //use ndc coordinates
1283    //36bins
1284    if (nBinPhiAddition == 36) {
1285      Tl.DrawLatex(0.561, 0.895, "15#circ");
1286      Tl.DrawLatex(0.761, 0.785, "45#circ");
1287      Tl.DrawLatex(0.872, 0.59, "75#circ");
1288      Tl.DrawLatex(0.872, 0.369, "105#circ");
1289      Tl.DrawLatex(0.754, 0.177, "135#circ");
1290      Tl.DrawLatex(0.561, 0.064, "165#circ");
1291      Tl.DrawLatex(0.323, 0.064, "195#circ");
1292      Tl.DrawLatex(0.121, 0.177, "225#circ");
1293      Tl.DrawLatex(0.007, 0.369, "255#circ");
1294      Tl.DrawLatex(0.007, 0.59, "285#circ");
1295      Tl.DrawLatex(0.121, 0.785, "315#circ");
1296      Tl.DrawLatex(0.323, 0.895, "345#circ");
1297    }
1298
1299    //18bins
1300    if (nBinPhiAddition == 18) {
1301      Tl.DrawLatex(0.595, 0.884, "20#circ");
1302      Tl.DrawLatex(0.826, 0.698, "60#circ");
1303      Tl.DrawLatex(0.878, 0.404, "100#circ");
1304      Tl.DrawLatex(0.725, 0.153, "140#circ");
1305      Tl.DrawLatex(0.153, 0.153, "220#circ");
1306      Tl.DrawLatex(0., 0.404, "260#circ");
1307      Tl.DrawLatex(0.049, 0.698, "300#circ");
1308      Tl.DrawLatex(0.290, 0.884, "340#circ");
1309    }
1310
1311
1312    //theta axis labels
1313    if (nBinThetaAddition == 9) {
1314      Tl.SetTextColor(kGray+1);
1315      Tl.DrawLatex(.446, 0.59, "25#circ");
1316      Tl.DrawLatex(.446, 0.677, "45#circ");
1317      Tl.DrawLatex(.446, 0.763, "65#circ");
1318      Tl.DrawLatex(.446, 0.851, "85#circ");
1319    }
1320
1321
1322    //~ gPad->Modified();
1323    //~ //gPad->Update();
1324
1325    //headline as latex text
1326    Tl.SetTextColor(kBlack);
1327    Tl.SetTextSize(40);
1328    Tl.DrawLatex(.326, 0.957, "Relativer␣Fehler");
1329
1330    Tl.SetTextSize(32);
1331    //~ Tl.DrawLatex(0.451, 0.907, "N");
1332    //~ Tl.DrawLatex(0.449, 0.049, "S");
1333    //~ Tl.DrawLatex(0.887, 0.48, "E");
1334    //~ Tl.DrawLatex(0.007, 0.48, "W");
```

```
1335
1336
1337    //plot of flux
1338    TCanvas* c2 = new TCanvas("c2", "Polar␣Flux", 0, 10, 1000, 1000)
            ;
1339    //MergedHist->GetZaxis()->SetRangeUser(0, 4);
1340    c2->SetFillStyle(3305);
1341    c2->SetFillColor(kBlack);
1342    c2->SetTheta(90);
1343    c2->SetPhi(0);
1344    MergedHist->Draw("SURF1␣POL␣Z");
1345    c2->Update();
1346    palette = (TPaletteAxis*)MergedHist->GetListOfFunctions()->
            FindObject("palette");
1347    palette->SetX1NDC(0.925);
1348    palette->SetX2NDC(0.951);
1349    Tl.SetTextFont(43); Tl.SetTextSize(24); //font
1350    Tl.SetNDC(); //use ndc coordinates
1351    //36bins
1352    if (nBinPhiAddition == 36) {
1353      Tl.DrawLatex(0.561, 0.895, "15#circ");
1354      Tl.DrawLatex(0.761, 0.785, "45#circ");
1355      Tl.DrawLatex(0.872, 0.59, "75#circ");
1356      Tl.DrawLatex(0.872, 0.369, "105#circ");
1357      Tl.DrawLatex(0.754, 0.177, "135#circ");
1358      Tl.DrawLatex(0.561, 0.064, "165#circ");
1359      Tl.DrawLatex(0.323, 0.064, "195#circ");
1360      Tl.DrawLatex(0.121, 0.177, "225#circ");
1361      Tl.DrawLatex(0.007, 0.369, "255#circ");
1362      Tl.DrawLatex(0.007, 0.59, "285#circ");
1363      Tl.DrawLatex(0.121, 0.785, "315#circ");
1364      Tl.DrawLatex(0.323, 0.895, "345#circ");
1365    }
1366
1367    //18bins
1368    if (nBinPhiAddition == 18) {
1369      Tl.DrawLatex(0.595, 0.884, "20#circ");
1370      Tl.DrawLatex(0.826, 0.698, "60#circ");
1371      Tl.DrawLatex(0.878, 0.404, "100#circ");
1372      Tl.DrawLatex(0.725, 0.153, "140#circ");
1373      Tl.DrawLatex(0.153, 0.153, "220#circ");
1374      Tl.DrawLatex(0., 0.404, "260#circ");
1375      Tl.DrawLatex(0.049, 0.698, "300#circ");
1376      Tl.DrawLatex(0.290, 0.884, "340#circ");
1377    }
1378
1379
1380    //theta axis labels
1381    if (nBinThetaAddition == 9) {
1382      Tl.SetTextColor(kGray+1);
1383      Tl.DrawLatex(.446, 0.59, "25#circ");
1384      Tl.DrawLatex(.446, 0.677, "45#circ");
1385      Tl.DrawLatex(.446, 0.763, "65#circ");
1386      Tl.DrawLatex(.446, 0.851, "85#circ");
1387    }
```

```
1388
1389
1390    //~ gPad->Modified();
1391    //~ //gPad->Update();
1392
1393    //headline as latex text
1394    Tl.SetTextColor(kBlack);
1395    Tl.SetTextSize(40);
1396    Tl.DrawLatex(.326, 0.957, "Flux␣in␣1/(m^{2}#upoint␣s␣#upoint␣sr)
           ");
1397
1398    Tl.SetTextSize(32);
1399    //~ Tl.DrawLatex(0.451, 0.907, "N");
1400    //~ Tl.DrawLatex(0.449, 0.049, "S");
1401    //~ Tl.DrawLatex(0.887, 0.48, "E");
1402    //~ Tl.DrawLatex(0.007, 0.48, "W");
1403
1404      //-----------------------------------------
1405      //
1406      // output file - save added histograms to .root file
1407      //-----------------------------------------
1408
1409    ostringstream filenamestream;
1410    filenamestream << "Results/Mts4Runs";
1411    for (Int_t j = 0; j < nHist; j++)
1412      filenamestream << runNumber[j] << "_";
1413    filenamestream << "added" ;
1414    string rootoutput = filenamestream.str();
1415    rootoutput.append(".root");
1416    TFile outfile (rootoutput.c_str(), "RECREATE");
1417    if (outfile.IsZombie() == true) {
1418      sprintf(buffer, "Output␣File␣could␣not␣be␣created");
1419      new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1420      return;
1421    }
1422    MergedHist->Write();
1423    dMergedHist->Write();
1424    outfile.Close();
1425
1426    //~ string pdfoutput = filenamestream.str();
1427    //~ pdfoutput.append(".png");
1428    //~ c2->SaveAs(pdfoutput.c_str());
1429  }
1430
1431  //!Calculate the integrated muon flux by multiplying the content
         of each bin with the solid angle
1432  void MtsMain::CalculateIntegratedFlux()
1433  {
1434    const Int_t nMax = 10;
1435    Int_t runNumber[nMax+1];
1436    Int_t runMin, runMax;
1437    Int_t nHist;
1438    const char *runs_char = RunNumbersT->GetText();
1439    //get run numbers
1440    if (strchr(runs_char, '-') == NULL) {
```

```
1441      nHist = sscanf(runs_char, "%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d",
1442                     &runNumber[0], &runNumber[1], &runNumber[2], &
                           runNumber[3],
1443                     &runNumber[4], &runNumber[5], &runNumber[6], &
                           runNumber[7],
1444                     &runNumber[8], &runNumber[9], &runNumber[10]);
1445      if (nHist > nMax) {
1446        sprintf(buffer, "too␣many␣runs,␣merge␣only␣first␣%d", nMax);
1447        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1448      }
1449    }
1450    else {
1451      sscanf(runs_char, "%d␣%s␣%d", &runMin, buffer, &runMax);
1452      nHist = runMax - runMin + 1;
1453      if ((runMax - runMin + 1) > nMax) {
1454        sprintf(buffer, "too␣many␣runs,␣only␣runs␣%d␣to␣%d␣will␣be␣
             merged",
1455                runMin, runMin + nMax - 1);
1456        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1457      }
1458      for (Int_t i = 0; i < nHist; i++)
1459        runNumber[i] = runMin + i;
1460    }
1461
1462    ostringstream filenamestream;
1463    filenamestream << "Results/Mts4Runs";
1464    for (Int_t j = 0; j < nHist; j++)
1465      filenamestream << runNumber[j] << "_";
1466    filenamestream << "added" ;
1467    string rootinput = filenamestream.str();
1468    rootinput.append(".root");
1469    TFile rootFile (rootinput.c_str(), "READ");
1470    if (rootFile.IsZombie() == true) {
1471      new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "Please␣
             perform␣Multiple"
1472                   "␣File␣Analysis␣first.");
1473      return;
1474    }
1475    TH2F* MergedHist = (TH2F*) rootFile.Get("Flux␣Histogram");
1476    TH2F* dMergedHist = (TH2F*) rootFile.Get("Flux␣error");
1477    Int_t nBinPhiIntegral = MergedHist->GetNbinsX();
1478    Int_t nBinThetaIntegral = MergedHist->GetNbinsY();
1479
1480    Double_t actualTheta;
1481
1482    dPhi = (phiMax - phiMin) / nBinPhiIntegral;
1483    dTheta = (thetaMax - thetaMin) / (nBinThetaIntegral - 1);
1484
1485    Double_t omega = 2 * PI * (cos(0) - cos(dTheta * PI / 180.));
1486    Double_t integratedFlux = MergedHist->GetBinContent(1, 2) *
         omega;
1487    Double_t dIntegratedFlux = dMergedHist->GetBinContent(1, 2) *
         omega;
1488
1489    for (Int_t Ny = 3; Ny < nBinThetaIntegral + 1; Ny++) {
```

```
1490      for (Int_t Nx = 1; Nx < nBinPhiIntegral + 1; Nx++) {
1491        actualTheta = (Ny - 2.) * dTheta;
1492        omega = dPhi * PI / 180. * (cos(actualTheta * PI / 180.)
1493                                    - cos((actualTheta + dTheta) *
                                           PI / 180.));
1494        integratedFlux += MergedHist->GetBinContent(Nx, Ny) * omega;
1495        dIntegratedFlux += dMergedHist->GetBinContent(Nx, Ny) *
              omega;
1496      }
1497    }
1498    sprintf(buffer, "Integrated Flux = (%lf +/- %lf) /m^2/s",
          integratedFlux,
1499              dIntegratedFlux);
1500    new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
1501    rootFile.Close();
1502  }
1503
1504  //!Show output of multiple file analysis
1505  void MtsMain::MultiplePlot()
1506  {
1507    //get filename from textbox in ui
1508    const Int_t nMax = 10;
1509    Int_t runNumber[nMax+1];
1510    Int_t runMin, runMax;
1511    Int_t nHist;
1512    const char *runs_char = RunNumbersT->GetText();
1513    if (strchr(runs_char, '-') == NULL) {
1514      nHist = sscanf(runs_char, "%d %d %d %d %d %d %d %d %d %d %d",
            &runNumber[0],
1515                     &runNumber[1], &runNumber[2], &runNumber[3], &
                         runNumber[4],
1516                     &runNumber[5], &runNumber[6], &runNumber[7], &
                         runNumber[8],
1517                     &runNumber[9], &runNumber[10]);
1518      if (nHist > nMax) {
1519        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "too many
              runs");
1520        return;
1521      }
1522    }
1523    else {
1524      sscanf(runs_char, "%d %s %d", &runMin, buffer, &runMax);
1525      nHist = runMax - runMin + 1;
1526      if (nHist > nMax) {
1527        new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, "too many
              runs");
1528      }
1529      for (Int_t i = 0; i < nHist; i++)
1530        runNumber[i] = runMin + i;
1531    }
1532
1533    ostringstream filenamestream;
1534    filenamestream << "Results/Mts4Runs";
1535    for (Int_t j = 0; j < nHist; j++)
1536      filenamestream << runNumber[j] << "_";
```

```
1537    filenamestream << "added" ;
1538
1539    string pdfoutput = filenamestream.str();
1540    pdfoutput.append(".pdf");
1541    sprintf(Command, "xpdf %s", pdfoutput.c_str());
1542    returnCode = system(Command);
1543  }
1544
1545
1546  //!Save settings
1547  void MtsMain::CloseWindow()
1548  {
1549    timer->TurnOff();
1550    ofstream oldSettings("temp/OldSettings.temp");
1551    if (!oldSettings.good()) {
1552      return;
1553    }
1554    oldSettings << "UpdateInterval " << UpdateIntervalT->
          GetIntNumber() << endl;
1555    oldSettings << "ProcessRunNo " << ProcessRunNumberT->
          GetIntNumber() << endl;
1556    oldSettings << "BinsizeTheta " << BinsizeThetaT->GetNumber() <<
          endl;
1557    oldSettings << "BinsizePhi " << BinsizePhiT->GetNumber() << endl
          ;
1558    oldSettings << "Timebin " << TimebinT->GetIntNumber() << endl;
1559    oldSettings << "PolarAngle " << PolarAngleT->GetNumber() << endl
          ;
1560    oldSettings << "AzimuthalAngle " << AzimuthalAngleT->GetNumber()
           << endl;
1561    if (OtherRotationB->IsOn())
1562      oldSettings << "OtherRotation " << 1 << endl;
1563    else
1564      oldSettings << "OtherRotation " << 0 << endl;
1565    oldSettings << "RunNumbers " << RunNumbersT->GetText() << endl;
1566    oldSettings.close();
1567  }
1568
1569  MtsMain::~MtsMain()
1570  {
1571    timer->TurnOff();
1572  }
1573
1574  MtsMain::MtsMain()
1575  {
1576    phiMin = 0.;
1577    phiMax = 360.;
1578    thetaMin = 0.;
1579    thetaMax = 90.;
1580    isConnected = false;
1581
1582    //open GuiSettings.ini to get IpAddress
1583    FILE* GuiSettings = fopen("ini/GuiSettings.ini", "r");
1584    if (GuiSettings == NULL) {
1585      cerr << "File named 'ini/GuiSettings.ini' cannot be read. Exit
```

```
        " << endl;
1586      exit(1);
1587    }
1588    Char_t guiSettName[100], guiSettValue[100], address[100];
1589    while (!feof(GuiSettings)) {
1590      returnCode = fscanf(GuiSettings, "%s␣%s␣", guiSettName,
            guiSettValue);
1591      if (strcmp(guiSettName, "NChambers") == 0)
1592        sscanf(guiSettValue, "%d", &nChamber);
1593      else if (strcmp(guiSettName, "IpAddress") == 0)
1594        strcpy(address, guiSettValue);
1595    }
1596    ipAddressString = address;
1597    fclose(GuiSettings);
1598
1599    //GUI components
1600
1601    ////////////////////
1602    // main frame
1603    ////////////////////
1604    MainFrame = new TGMainFrame(gClient->GetRoot(), 10, 10,
1605                                kMainFrame | kVerticalFrame);
1606    MainFrame->SetName("MainFrame");
1607    MainFrame->SetLayoutBroken(kTRUE);
1608    MainFrame->SetWindowName("Mts4GUI");
1609    MainFrame->Connect("CloseWindow()", "MtsMain", this, "
        CloseWindow()");
1610
1611    ////////////////////////////////
1612    // IP frame
1613    ////////////////////////////////
1614    TGVerticalFrame *IpFrame = new TGVerticalFrame(MainFrame, 360,
        32);
1615    IpFrame->SetName("IpFrame");
1616    IpFrame->SetLayoutBroken(kTRUE);
1617
1618    //IP label
1619    TGLabel *IpAddressL = new TGLabel(IpFrame, "IP␣address");
1620    IpFrame->AddFrame(IpAddressL);
1621    IpAddressL->Move(8, 9);
1622
1623    //IP text
1624    IpAddressT = new TGTextEntry(IpFrame);
1625    IpAddressT->SetAlignment(kTextRight);
1626    IpAddressT->SetText(address);
1627    IpFrame->AddFrame(IpAddressT);
1628    IpAddressT->MoveResize(204, 6, 126, 22);
1629    IpAddressT->Connect("TextChanged(char*)", "MtsMain",
1630                        this, "TextEdit(char*)");
1631    IpAddressT->SetToolTipText("Ip␣address␣of␣the␣muon␣telescopes␣
        raspberry␣pi,\n"
1632                               "standard␣is␣192.168.0.1");
1633
1634    MainFrame->AddFrame(IpFrame);
1635    IpFrame->MoveResize(8, 8, 360, 32);
```

```
1636
1637    ////////////////////////////////////
1638    // control run frame
1639    ////////////////////////////////////
1640    TGGroupFrame *ControlRunF = new TGGroupFrame(MainFrame, "Control
            Run");
1641    ControlRunF->SetLayoutBroken(kTRUE);
1642
1643    //run number label
1644    TGLabel *RunNumberL = new TGLabel(ControlRunF, "Run Number");
1645    ControlRunF->AddFrame(RunNumberL);
1646    RunNumberL->Move(8, 24);
1647
1648    //run number text
1649    RunNumberT = new TGNumberEntry(ControlRunF, 50, 5, -1,
1650                                   TGNumberFormat::kNESInteger,
1651                                   TGNumberFormat::kNEANonNegative);
1652    ControlRunF->AddFrame(RunNumberT);
1653    RunNumberT->MoveResize(204, 24, 126, 18);
1654    runNumberInt = RunNumberT->GetIntNumber();
1655    RunNumberT->GetNumberEntry()->SetToolTipText("run number for
            current "
1656                                                 "measurement");
1657
1658    //statistics label
1659    TGLabel *StatisticsL = new TGLabel(ControlRunF, "Statistics");
1660    ControlRunF->AddFrame(StatisticsL);
1661    StatisticsL->Move(8, 46);
1662
1663    //statistics text
1664    StatisticsT = new TGNumberEntry(ControlRunF, -1, 6, -1,
1665                                    TGNumberFormat::kNESInteger);
1666    ControlRunF->AddFrame(StatisticsT);
1667    StatisticsT->MoveResize(204, 46, 126, 18);
1668    StatisticsT->GetNumberEntry()->SetToolTipText("number of events
            till the "
1669                                                  "measurements stop
                                                      ,\n"
1670                                                  "-1 for infinite
                                                      measuring "
1671                                                  "until manual
                                                      stopping");
1672
1673    //used gas label
1674    TGLabel *UsedGasL = new TGLabel(ControlRunF, "Used Gas");
1675    ControlRunF->AddFrame(UsedGasL);
1676    UsedGasL->Move(8, 68);
1677
1678    //used gas text
1679    UsedGasT = new TGTextEntry(ControlRunF);
1680    UsedGasT->SetAlignment(kTextRight);
1681    UsedGasT->SetText("Ar:CO2 82:18");
1682    ControlRunF->AddFrame(UsedGasT);
1683    UsedGasT->MoveResize(204, 68, 126, 18);
1684    UsedGasT->SetToolTipText("information on used gas");
```

```
1685
1686    //high voltage sense wire label
1687    TGLabel *HV_SenseWireL = new TGLabel(ControlRunF, "HV:␣Sense␣
           Wire␣[V]");
1688    ControlRunF->AddFrame(HV_SenseWireL);
1689    HV_SenseWireL->Move(8,90);
1690
1691    //high voltage sense wire text
1692    HV_SenseWireT = new TGNumberEntry(ControlRunF, 1060, 6, -1,
1693                                      TGNumberFormat::kNESInteger,
1694                                      TGNumberFormat::kNEAPositive);
1695    ControlRunF->AddFrame(HV_SenseWireT);
1696    HV_SenseWireT->MoveResize(204, 90, 126, 18);
1697    HV_SenseWireT->GetNumberEntry()->SetToolTipText("High␣Voltage␣on
           ␣the␣sense␣"
1698                                                    "wires␣in␣V");
1699
1700    //high voltage sense wire current label
1701    TGLabel *HV_SW_CurrentL = new TGLabel(ControlRunF, "HV:␣SW␣
           Current␣[nA]");
1702    ControlRunF->AddFrame(HV_SW_CurrentL);
1703    HV_SW_CurrentL->Move(8, 112);
1704
1705    //high voltage sense wire current text
1706    HV_SW_CurrentT = new TGNumberEntry(ControlRunF, 2, 6, -1,
1707                                       TGNumberFormat::kNESInteger);
1708    ControlRunF->AddFrame(HV_SW_CurrentT);
1709    HV_SW_CurrentT->MoveResize(204, 112, 126, 18);
1710    HV_SW_CurrentT->GetNumberEntry()->SetToolTipText("Current␣on␣the
           ␣sense␣wires"
1711                                                     "␣in␣nA");
1712
1713    //additional information label
1714    TGLabel *AdditionalInfoL = new TGLabel(ControlRunF,
1715                                           "Additional␣Run␣
                                              Information");
1716    ControlRunF->AddFrame(AdditionalInfoL);
1717    AdditionalInfoL->Move(8, 134);
1718
1719    //additional information textbox
1720    AdditionalInformationT = new TGTextEdit(ControlRunF, 324, 80);
1721    ControlRunF->AddFrame(AdditionalInformationT);
1722    AdditionalInformationT->MoveResize(8, 156, 324, 80);
1723    Pixel_t pxl;
1724    gClient->GetColorByName("#3399ff", pxl);
1725    AdditionalInformationT->SetSelectBack(pxl);
1726    AdditionalInformationT->SetSelectFore(TGFrame::GetWhitePixel());
1727
1728    //status label
1729    StatusL = new TGLabel(ControlRunF, "Status");
1730    StatusL->SetTextJustify(kTextLeft);
1731    ControlRunF->AddFrame(StatusL);
1732    StatusL->MoveResize(8, 240, 300, 18);
1733
1734    //events label
```

```
1735    EventsL = new TGLabel(ControlRunF, "Events");
1736    EventsL->SetTextJustify(kTextLeft);
1737    ControlRunF->AddFrame(EventsL);
1738    EventsL->MoveResize(8, 262, 300, 18);
1739
1740    //Start, Update and Stop Buttons
1741    TGTextButton *StartB = new TGTextButton(ControlRunF, "Start");
1742    ControlRunF->AddFrame(StartB);
1743    StartB->MoveResize(8, 284, 80, 24);
1744    StartB->Connect("Clicked()", "MtsMain", this, "Start()");
1745    StartB->SetToolTipText("Verifies␣connection␣to␣the␣raspberry␣and
        ␣"
1746                           "non-existence␣of␣the␣specified␣run.\
                                nAfter␣that␣the␣"
1747                           "measurement␣is␣started.");
1748
1749    TGTextButton *UpdateB = new TGTextButton(ControlRunF, "Update");
1750    ControlRunF->AddFrame(UpdateB);
1751    UpdateB->MoveResize(125, 284, 80, 24);
1752    UpdateB->Connect("Clicked()", "MtsMain", this, "Update()");
1753    UpdateB->SetToolTipText("Updates␣run␣status␣labels␣with␣current␣
        measurement␣"
1754                            "status");
1755
1756    TGTextButton *StopB = new TGTextButton(ControlRunF, "Stop");
1757    ControlRunF->AddFrame(StopB);
1758    StopB->MoveResize(247, 284, 80, 24);
1759    StopB->Connect("Clicked()", "MtsMain", this, "Stop()");
1760    StopB->SetToolTipText("Stops␣the␣measurement");
1761
1762    //Update Interval label
1763    TGLabel *UpdateIntervalL = new TGLabel(ControlRunF, "Update␣
        Interval␣[min]");
1764    ControlRunF->AddFrame(UpdateIntervalL);
1765    UpdateIntervalL->Move(8,313);
1766
1767    //Update Interval text
1768    UpdateIntervalT = new TGNumberEntry(ControlRunF, 30, 6, -1,
1769                                        TGNumberFormat::kNESInteger,
1770                                        TGNumberFormat::
                                               kNEANonNegative);
1771    ControlRunF->AddFrame(UpdateIntervalT);
1772    UpdateIntervalT->MoveResize(204, 313, 126, 18);
1773    UpdateIntervalT->Connect("ValueSet(Long_t)", "MtsMain",
1774                             this, "UpdateIntervalEdit(Long_t)");
1775    UpdateIntervalT->GetNumberEntry()->SetToolTipText("time␣in␣
        minutes␣in␣between"
1776                                                      "␣updates");
1777
1778    //ControlRunF->SetLayoutManager(new TGVerticalLayout(ControlRunF
        ));
1779    MainFrame->AddFrame(ControlRunF);
1780    ControlRunF->MoveResize(8, 48, 342, 344);
1781
1782    ////////////////////////////////
```

```
1783    //process data frame
1784    ////////////////////////////////
1785
1786    TGGroupFrame *ProcessDataF = new TGGroupFrame(MainFrame, "
            Process Data");
1787    ProcessDataF->SetLayoutBroken(kTRUE);
1788
1789    //copy data and remove data button
1790    TGTextButton *CopyDataB = new TGTextButton(ProcessDataF, "Copy
            Data");
1791    ProcessDataF->AddFrame(CopyDataB);
1792    CopyDataB->MoveResize(50, 24, 100, 24);
1793    CopyDataB->Connect("Clicked()", "MtsMain", this, "CopyData()");
1794    CopyDataB->SetToolTipText("Opens a popup window, that enables
            you to copy\n"
1795                              "the measurement data from the
                                 raspberry to"
1796                              "your device");
1797
1798    TGTextButton *RemoveDataB = new TGTextButton(ProcessDataF, "
            Remove Data");
1799    ProcessDataF->AddFrame(RemoveDataB);
1800    RemoveDataB->MoveResize(192, 24, 100, 24);
1801    RemoveDataB->Connect("Clicked()", "MtsMain", this, "RemoveData()
            ");
1802    RemoveDataB->SetToolTipText("Opens a popup window, that enables
            you to "
1803                                "remove measurement data from the
                                   raspberry");
1804
1805    //single file analysis label
1806    TGLabel *SingleFileL = new TGLabel(ProcessDataF, "Single File
            Analysis");
1807    ProcessDataF->AddFrame(SingleFileL);
1808    SingleFileL->Move(12, 53);
1809
1810    //separator line
1811    TGHorizontal3DLine *SingleLine = new TGHorizontal3DLine(
            ProcessDataF, 209, 8);
1812    ProcessDataF->AddFrame(SingleLine);
1813    SingleLine->MoveResize(130, 62, 209, 7);
1814
1815    //run number for processing label
1816    TGLabel *ProcessRunNumberL = new TGLabel(ProcessDataF, "Run
            Number");
1817    ProcessDataF->AddFrame(ProcessRunNumberL);
1818    ProcessRunNumberL->Move(8, 81);
1819
1820    //run number for processing text
1821    ProcessRunNumberT = new TGNumberEntry(ProcessDataF, 53, 6, -1,
1822                                          TGNumberFormat::
                                              kNESInteger,
1823                                          TGNumberFormat::
                                              kNEAPositive);
1824    ProcessDataF->AddFrame(ProcessRunNumberT);
```

```
1825    ProcessRunNumberT->MoveResize(104, 81, 50, 18);
1826    ProcessRunNumberT->GetNumberEntry()->SetToolTipText("run␣number␣
           for␣analysing"
1827                                                        "␣a␣single␣
                                                              run");
1828
1829    //theta binsize label
1830    TGLabel *BinsizeThetaL = new TGLabel(ProcessDataF, "Binsize␣
           Theta");
1831    ProcessDataF->AddFrame(BinsizeThetaL);
1832    BinsizeThetaL->Move(8, 103);
1833
1834    //theta binsize text
1835    BinsizeThetaT = new TGNumberEntry(ProcessDataF, 5.0, 6, -1,
1836                                      TGNumberFormat::kNESReal,
1837                                      TGNumberFormat::kNEAPositive,
1838                                      TGNumberFormat::
                                             kNELLimitMinMax,
1839                                      phiMin, phiMax);
1840    ProcessDataF->AddFrame(BinsizeThetaT);
1841    BinsizeThetaT->MoveResize(104, 103, 50, 18);
1842    BinsizeThetaT->GetNumberEntry()->SetToolTipText("size␣of␣one␣
           theta␣bin␣("
1843                                                    "polar␣direction
                                                          )␣in␣degree")
                                                          ;
1844
1845    //phi binsize label
1846    TGLabel *BinsizePhiL = new TGLabel(ProcessDataF, "Binsize␣Phi");
1847    ProcessDataF->AddFrame(BinsizePhiL);
1848    BinsizePhiL->Move(8, 125);
1849
1850    //phi binsize text
1851    BinsizePhiT = new TGNumberEntry(ProcessDataF, 10.0, 6, -1,
1852                                    TGNumberFormat::kNESReal,
1853                                    TGNumberFormat::kNEAPositive,
1854                                    TGNumberFormat::kNELLimitMinMax,
1855                                    thetaMin, thetaMax);
1856    ProcessDataF->AddFrame(BinsizePhiT);
1857    BinsizePhiT->MoveResize(104, 125, 50, 18);
1858    BinsizePhiT->GetNumberEntry()->SetToolTipText("size␣of␣one␣phi␣
           bin␣(azimuthal"
1859                                                  "␣direction)␣in␣
                                                        degree");
1860
1861    //timebin label
1862    TGLabel *TimebinL = new TGLabel(ProcessDataF, "Timebins");
1863    ProcessDataF->AddFrame(TimebinL);
1864    TimebinL->Move(8, 147);
1865
1866    //timebin text
1867    TimebinT = new TGNumberEntry(ProcessDataF, 25, 6, -1,
1868                                 TGNumberFormat::kNESInteger,
1869                                 TGNumberFormat::kNEAPositive);
1870    ProcessDataF->AddFrame(TimebinT);
```

```
1871    TimebinT->MoveResize(104, 147, 50, 18);
1872    TimebinT->GetNumberEntry()->SetToolTipText("number␣of␣timebins␣
            for␣the␣"
1873                                               "analysis");
1874
1875    //polar angle label
1876    TGLabel *PolarAngleL = new TGLabel(ProcessDataF, "Polar␣angle");
1877    ProcessDataF->AddFrame(PolarAngleL);
1878    PolarAngleL->Move(172, 90);
1879
1880    //polar angle text
1881    PolarAngleT = new TGNumberEntry(ProcessDataF, 0, 6, -1,
1882                                    TGNumberFormat::kNESReal,
1883                                    TGNumberFormat::kNEAAnyNumber);
1884    ProcessDataF->AddFrame(PolarAngleT);
1885    PolarAngleT->MoveResize(266, 88, 50, 18);
1886    PolarAngleT->GetNumberEntry()->SetToolTipText("rotation␣angle␣
            for␣the␣polar␣"
1887                                                  "angle␣theta␣in␣
                                                         degree\n"
1888                                                  "check␣
                                                         documentation␣
                                                         for␣more␣"
1889                                                  "information␣on␣
                                                         rotations");
1890
1891    //azimuthal angle label
1892    TGLabel *AzimuthalAngleL = new TGLabel(ProcessDataF, "Azimuthal␣
            angle");
1893    ProcessDataF->AddFrame(AzimuthalAngleL);
1894    AzimuthalAngleL->Move(172, 112);
1895
1896    //azimuthal angle text
1897    AzimuthalAngleT = new TGNumberEntry(ProcessDataF, 0, 6, -1,
1898                                        TGNumberFormat::kNESReal,
1899                                        TGNumberFormat::
                                               kNEAAnyNumber);
1900    ProcessDataF->AddFrame(AzimuthalAngleT);
1901    AzimuthalAngleT->MoveResize(266, 112, 50, 18);
1902    AzimuthalAngleT->GetNumberEntry()->SetToolTipText("rotation␣
            angle␣for␣the␣"
1903                                                      "azimuthal␣
                                                             angle␣phi␣"
1904                                                      "in␣degree\
                                                             ncheck␣"
1905                                                      "documentation
                                                             ␣for␣more␣"
1906                                                      "information␣
                                                             on␣
                                                             rotations")
                                                             ;
1907
1908    //other rotation label
1909    TGLabel *OtherRotationL = new TGLabel(ProcessDataF, "Other␣
            Rotation");
```

```
1910    ProcessDataF->AddFrame(OtherRotationL);
1911    OtherRotationL->Move(172, 134);
1912
1913    //other rotation check button
1914    OtherRotationB = new TGCheckButton(ProcessDataF, "");
1915    ProcessDataF->AddFrame(OtherRotationB);
1916    OtherRotationB->MoveResize(284, 134, 15, 18);
1917    OtherRotationB->SetToolTipText("check␣if␣you␣want␣to␣rotate␣
            around␣different"
1918                                   "␣axes\ncheck␣documentation␣for␣
                                       more␣"
1919                                   "information␣on␣rotations");
1920
1921    //single analysis and show plots button
1922    TGTextButton *SingleAnalysisB = new TGTextButton(ProcessDataF, "
            Analyze");
1923    ProcessDataF->AddFrame(SingleAnalysisB);
1924    SingleAnalysisB->MoveResize(50, 169, 100, 24);
1925    SingleAnalysisB->Connect("Clicked()", "MtsMain", this, "Analyze
            ()");
1926    SingleAnalysisB->SetToolTipText("Opens␣a␣popup,␣that␣provides␣
            different␣"
1927                                    "options␣for␣the␣analysis␣of␣a␣
                                        run.\n"
1928                                    "1.␣run␣analysis:␣calculates␣
                                        tracks␣from"
1929                                    "␣measurement␣data\n"
1930                                    "2.␣polar␣analysis:␣calculates␣a
                                        ␣polar␣2D"
1931                                    "plot\n"
1932                                    "3.␣full␣analysis:␣performs␣run␣
                                        analysis"
1933                                    "first␣and␣polar␣analysis␣second
                                        ");
1934
1935    TGTextButton *SingleShowPlotsB = new TGTextButton(ProcessDataF,
            "Show␣Plots");
1936    ProcessDataF->AddFrame(SingleShowPlotsB);
1937    SingleShowPlotsB->MoveResize(192, 169, 100, 24);
1938    SingleShowPlotsB->Connect("Clicked()", "MtsMain", this, "
            SinglePlot()");
1939    SingleShowPlotsB->SetToolTipText("Opens␣the␣.pdf␣created␣during␣
            run␣analysis,"
1940                                     "␣which␣contains␣several␣plots␣
                                         for␣stability"
1941                                     "␣assessment,␣a␣carthesian␣
                                         hitmap␣of␣the␣"
1942                                     "muons␣and,␣after␣executing␣
                                         polar␣analysis,␣"
1943                                     "a␣polar␣2D␣plot␣of␣the␣
                                         muonflux");
1944
1945    //multiple file analysis label
1946    TGLabel *ManyFilesL = new TGLabel(ProcessDataF, "Multiple␣File␣
            Analysis");
```

```
1947    ProcessDataF->AddFrame(ManyFilesL);
1948    ManyFilesL->Move(12, 198);
1949
1950    //separator
1951    TGHorizontal3DLine *ManyLine = new TGHorizontal3DLine(
            ProcessDataF, 201, 8);
1952    ProcessDataF->AddFrame(ManyLine);
1953    ManyLine->MoveResize(138, 207, 201, 8);
1954
1955    //run numbers label
1956    TGLabel *RunNumbersL = new TGLabel(ProcessDataF, "Run␣Numbers");
1957    ProcessDataF->AddFrame(RunNumbersL);
1958    RunNumbersL->Move(7, 224);
1959
1960    //run numbers text
1961    RunNumbersT = new TGTextEntry(ProcessDataF);
1962    RunNumbersT->SetAlignment(kTextRight);
1963    ProcessDataF->AddFrame(RunNumbersT);
1964    RunNumbersT->MoveResize(156, 224, 174, 18);
1965    RunNumbersT->SetToolTipText("compilation␣of␣runs␣for␣one␣
            telescope␣location,"
1966                                "\nthat␣are␣to␣be␣merged␣via␣a␣
                                       weighted␣mean");
1967
1968    //multiple analysis and multiple show plots buttons
1969    TGTextButton *MultipleAnalyseB = new TGTextButton(ProcessDataF,
            "Analyze");
1970    ProcessDataF->AddFrame(MultipleAnalyseB);
1971    MultipleAnalyseB->MoveResize(8, 248, 80, 24);
1972    MultipleAnalyseB->Connect("Clicked()", "MtsMain", this,
1973                              "GetWeightedHistogramMean()");
1974    MultipleAnalyseB->SetToolTipText("Performs␣the␣weighted␣mean␣
            with␣the␣polar␣"
1975                                     "histograms␣of␣the␣given␣run␣
                                            numbers");
1976
1977    TGTextButton *MultiplePlotsB = new TGTextButton(ProcessDataF, "
            Show␣Plots");
1978    ProcessDataF->AddFrame(MultiplePlotsB);
1979    MultiplePlotsB->MoveResize(125, 248, 80, 24);
1980    MultiplePlotsB->Connect("Clicked()", "MtsMain", this, "
            MultiplePlot()");
1981    MultiplePlotsB->SetToolTipText("Opens␣the␣.pdf,␣that␣contains␣
            the␣result"
1982                                   "of␣the␣weighted␣mean␣of␣the␣
                                          given␣run␣"
1983                                   "numbers");
1984
1985    TGTextButton *IntegratedFluxB = new TGTextButton(ProcessDataF,
1986                                                     "Integrate␣Flux
                                                            ");
1987    ProcessDataF->AddFrame(IntegratedFluxB);
1988    IntegratedFluxB->MoveResize(246, 248, 82, 24);
1989    IntegratedFluxB->Connect("Clicked()", "MtsMain", this,
1990                             "CalculateIntegratedFlux()");
```

```
1991    IntegratedFluxB->SetToolTipText("Calculate␣the␣Integrated␣Flux␣
            by␣adding␣up"
1992                                    "␣each␣bin␣multiplied␣with␣its␣
                                           solid␣angle.");
1993
1994    MainFrame->AddFrame(ProcessDataF);
1995    ProcessDataF->MoveResize(8, 392, 342, 290);
1996
1997    timer = new TTimer();
1998    timer->Connect("Timeout()", "MtsMain", this, "Update()");
1999    timer->Start(UpdateIntervalT->GetIntNumber() * 60 * 1000);
2000    Startup();
2001
2002    MainFrame->SetCleanup(kDeepCleanup);
2003    MainFrame->SetMWMHints( kMWMDecorAll, kMWMFuncAll,
            kMWMInputModeless);
2004    MainFrame->MapSubwindows();
2005    MainFrame->MapWindow();
2006    MainFrame->Resize(356,681);
2007  }
2008
2009  //!Calculates a polar plot starting from the slopes of the muon
        tracks
2010  /*!
2011    Takes all found tracks from the analysis, calculates the polar
           coordinates,
2012    * fills those into a 2D histogram and calculates the flux with
           its
2013    * respective error. The result is shown as a polar color plot
           and all
2014    * parameters are saved in a .txt file.
2015  */
2016  void MtsMain::AnalyzePolar()
2017  {
2018    Double_t Acceptance;
2019
2020    cerr << runToAnalyze << "␣" << time << "␣" << polarAngleDegree
           << "␣" ;
2021    cerr << azimuthalAngleDegree << "␣" << nBinPhi << "␣" <<
           nBinTheta << "␣" ;
2022    cerr << polarRotation << "␣" << azimuthalRotation << endl;
2023
2024    //height in channel units
2025    Double_t height = CHAMBER_DISTANCES[N_CHAMBER-1] -
           CHAMBER_DISTANCES[0];
2026
2027    //convert degree to radian
2028    Double_t polarAngle = polarAngleDegree * -1. * PI / 180.;
2029    Double_t azimuthalAngle = azimuthalAngleDegree * -1. * PI/180.;
2030
2031    //root tree elements
2032    UInt_t actualEvent;
2033    Double_t mPad;
2034    Double_t mFw;
2035    Double_t mPadEffTrack[N_CHAMBER];
```

```
2036    Double_t mFwEffTrack[N_CHAMBER];
2037    Double_t mPadEffTrigger[N_CHAMBER];
2038    Double_t mFwEffTrigger[N_CHAMBER];
2039
2040    //define output files
2041    ostringstream rn;
2042    rn << runToAnalyze;
2043    string run_number = rn.str();
2044    string rootout = "Results/Mts4Run";
2045    rootout.append(run_number);
2046    rootout.append(".root");
2047    cerr << rootout << endl;
2048
2049    ostringstream OutputStream;
2050    OutputStream << "Results/Mts4Run" << runToAnalyze;
2051    string pdfout = OutputStream.str();
2052    pdfout.append("_polar.pdf");
2053    string txtout = OutputStream.str();
2054    txtout.append(".txt");
2055
2056    //define dummy bin so theta = 0 is plotted, else there was only
          a white
2057    //area for the bins in the middle
2058    Double_t *array = new Double_t[nBinTheta+2];
2059
2060    array[0] = 0;
2061    array[1] = 0.000000001;
2062    for (Int_t i = 2; i < nBinTheta+2; i++) {
2063      array[i] = (i - 1) * (thetaMax - thetaMin) / nBinTheta;
2064    }
2065
2066    //initializing histograms
2067    //acceptance corrected polar hitmap
2068    TH2F* PolarHist = new TH2F(Form("h2d␣%d", runToAnalyze),"polar",
           nBinPhi, phiMin,
2069                               phiMax, nBinTheta+1, array);
2070    TH2F* FluxHist = new TH2F(Form("hist␣%d", runToAnalyze),"polar␣
          result",
2071                              nBinPhi, phiMin, phiMax, nBinTheta+1,
                                     array);
2072    TH2F* dFluxHist = new TH2F(Form("error␣%d", runToAnalyze),"error
          ", nBinPhi,
2073                                  phiMin, phiMax, nBinTheta+1,
                                         array);
2074    //polar hitmap without acceptance
2075    TH2I* tracksHist = new TH2I(Form("tracks␣%d", runToAnalyze),"
          hitmap", nBinPhi,
2076                                phiMin, phiMax, nBinTheta+1, array);
2077
2078
2079    /*TH1F* cutHist = new TH1F(Form("cut %d", runToAnalyze), "cut",
          36, -90, 90);
2080    TH1F* cutHistError = new TH1F(Form("cut1 %d", runToAnalyze), "
          cut1", 36, -90, 90);
2081    TH1I* cutHistTracks = new TH1I("a", "cut2", 36, -90, 90);
```

```
2082    Double_t dAngle = 10.;
2083    for (Int_t i = 1; i < 37; i++) {
2084      cutHist->SetBinContent(i, 0);
2085      cutHistError->SetBinContent(i, 0);
2086      cutHistTracks->SetBinContent(i, 0);
2087    }*/
2088
2089    delete array;
2090
2091    //initializing efficiency class
2092    TEfficiency* efficiency[N_CHAMBER];
2093    const TH1* passedEff[N_CHAMBER];
2094    const TH1* totalEff[N_CHAMBER];
2095
2096    for (Int_t j = 0; j < N_CHAMBER; j++) {
2097      efficiency[j] = new TEfficiency("eff", "", nBinPhi, phiMin,
           phiMax,
2098                                      nBinTheta, thetaMin, thetaMax)
                                          ;
2099      passedEff[j] = efficiency[j]->GetPassedHistogram();
2100      totalEff[j] = efficiency[j]->GetTotalHistogram();
2101    }
2102
2103    Int_t nTracks = 0;
2104
2105    //initialize TTree
2106    TFile rootFile(rootout.c_str(), "READ");
2107    if (rootFile.IsZombie() == true) {
2108      sprintf(buffer, "Input␣File␣could␣not␣be␣opened");
2109      new MtsPopup(gClient->GetRoot(), MainFrame, 5, 5, buffer);
2110      return;
2111    }
2112
2113    TTree* ResultsTree = (TTree*) rootFile.Get("resultsTree");
2114
2115    ResultsTree->SetBranchAddress("actualEvent", &actualEvent);
2116    ResultsTree->SetBranchAddress("mPad", &mPad);
2117    ResultsTree->SetBranchAddress("mFw", &mFw);
2118    ResultsTree->SetBranchAddress("mPadEffTrack", &mPadEffTrack);
2119    ResultsTree->SetBranchAddress("mFwEffTrack", &mFwEffTrack);
2120    ResultsTree->SetBranchAddress("mPadEffTrigger", &mPadEffTrigger)
         ;
2121    ResultsTree->SetBranchAddress("mFwEffTrigger", &mFwEffTrigger);
2122
2123    cerr << "test2" << endl;
2124    UInt_t nEntries = ResultsTree->GetEntries();
2125    cerr << "Entries:␣" << nEntries << endl;
2126    Coord coords;
2127
2128    //loop through all tree elements
2129    for (UInt_t i = 0; i < nEntries; i++) {
2130      ResultsTree->GetEntry(i);
2131      //neglect events with no 6pt track
2132      if (mPad != 1000 && mFw != 1000) {
2133        nTracks++;
```

```
2134        coords.x = mPad;
2135        coords.y = mFw;
2136        coords.z = 1.;
2137        coords.Rotation(polarAngle, polarRotation,
2138                        azimuthalAngle, azimuthalRotation);
2139        coords.CalculatePolarCoord();
2140        //calculate geometrical detector acceptance
2141        //(projection of slope on detector area)
2142        if (N_PAD - height * fabs(mPad) > 0 && N_FW - height * fabs(
               mFw) > 0) {
2143          Acceptance = pow(cos(atan(sqrt(mFw * mFw + mPad * mPad)))
                 ,1.) *
2144                       (N_PAD - height * fabs(mPad)) / N_PAD *
2145                       (N_FW - height * fabs(mFw)) / N_FW;
2146        }
2147        else {
2148          Acceptance = 0;
2149        }
2150        //fill acceptance corrected tracks into histogram
2151        if (Acceptance > 0) {
2152          PolarHist->Fill(coords.phi, coords.theta, 1./Acceptance);
2153          tracksHist->Fill(coords.phi, coords.theta);
2154        }
2155      }
2156
2157      //efficiency calculation for each chamber
2158      for (Int_t j = 0; j < N_CHAMBER; j++) {
2159        Bool_t isTriggered = false;
2160        //check for 5point tracks, where the channel at the 6th
               point was also hit
2161        if (mPadEffTrigger[j] != 1000 && mFwEffTrigger[j] != 1000) {
2162          coords.x = mPadEffTrigger[j];
2163          coords.y = mFwEffTrigger[j];
2164          coords.z = 1.;
2165          coords.Rotation(polarAngle, polarRotation,
2166                          azimuthalAngle, azimuthalRotation);
2167          coords.CalculatePolarCoord();
2168          isTriggered = true;
2169        }
2170        //only 5point tracks (always true, if previous if was true)
2171        if (mPadEffTrack[j] != 1000 && mFwEffTrack[j] != 1000) {
2172          coords.x = mPadEffTrack[j];
2173          coords.y = mFwEffTrack[j];
2174          coords.z = 1.;
2175          coords.Rotation(polarAngle, polarRotation,
2176                          azimuthalAngle, azimuthalRotation);
2177          coords.CalculatePolarCoord();
2178          //fill the polar coordinates into the two efficiency
                 histograms
2179          //isTriggered == true: both get filled
2180          //isTriggered == false: only the totalHistogram is filled
2181          efficiency[j]->Fill(isTriggered, coords.phi, coords.theta)
                 ;
2182        }
2183      }
```

```
2184    }
2185
2186    cerr << "filling␣histograms␣finished" << endl;
2187    cerr << "TRACKS␣" << nTracks << endl;
2188
2189    Double_t omega; //solid angle
2190
2191    //bunch of variables for error calculation
2192    Double_t dChamberEff[N_CHAMBER] = {0.};
2193    Double_t ChamberEff[N_CHAMBER] = {0.};
2194    Double_t effErrorProduct[N_CHAMBER];
2195    Double_t effErrorSum[N_CHAMBER];
2196    Double_t trackingEff;
2197    Double_t dTracks = 0.;
2198    Double_t dFlux;
2199    Double_t dFluxRelative;
2200    Double_t dEffSquared;
2201    Double_t dEff;
2202
2203    //sum up all middle bins and fill dummy bins (Ny = 1)
2204    //and middle bins (Ny = 2) with it
2205    //telescope doesn't have the angular resolution, which would be
        implied
2206    //by the small theta = 0  bins, therefore they're merged to one
        circular bin
2207    Int_t a[N_CHAMBER] = {0}, b[N_CHAMBER] = {0};
2208    Int_t c = 0;
2209    Double_t d = 0;
2210
2211    for (Int_t Nx = 1; Nx < nBinPhi + 1; Nx++) {
2212      d += PolarHist->GetBinContent(Nx, 2);
2213      c += tracksHist->GetBinContent(Nx, 2);
2214      for (Int_t j = 0; j < N_CHAMBER; j++) {
2215        a[j] += totalEff[j]->GetBinContent(totalEff[j]->GetBin(Nx,
              1));
2216        b[j] += passedEff[j]->GetBinContent(passedEff[j]->GetBin(Nx,
              1));
2217      }
2218    }
2219
2220    for (Int_t Nx = 1; Nx < nBinPhi + 1; Nx++) {
2221      PolarHist->SetBinContent(Nx, 1, d);
2222      PolarHist->SetBinContent(Nx, 2, d);
2223      tracksHist->SetBinContent(Nx, 1, c);
2224      tracksHist->SetBinContent(Nx, 2, c);
2225      for (Int_t j = 0; j < N_CHAMBER; j++) {
2226        efficiency[j]->SetTotalEvents(totalEff[j]->GetBin(Nx, 1), a[
              j]);
2227        efficiency[j]->SetPassedEvents(passedEff[j]->GetBin(Nx, 1),
              b[j]);
2228      }
2229    }
2230
2231    Double_t actualTheta;
2232    Bool_t isMiddleBinCovered = true;
```

```
2233    //loop through all bins
2234    for (Int_t Ny = 2; Ny < nBinTheta + 2; Ny++) {
2235      for (Int_t Nx = 1; Nx < nBinPhi + 1; Nx++) {
2236        actualTheta = (Ny - 2.) * dTheta;
2237        Double_t actualPhi = (Nx - 1) * dPhi;
2238        Int_t currentBin = efficiency[0]->GetGlobalBin(Nx, Ny-1);
2239
2240        //check if bin can be fully filled due to detector geometry
2241        Bool_t isCovered = true;
2242        Double_t mPadTest[4];
2243        Double_t mFwTest[4];
2244
2245        //check at each corner of the bin
2246        mPadTest[0] = cos(actualPhi * PI / 180.) * tan(actualTheta *
                PI / 180.);
2247        mPadTest[1] = cos((actualPhi + dPhi) * PI / 180.) *
2248                      tan(actualTheta * PI / 180.);
2249        mPadTest[2] = cos(actualPhi * PI / 180.) *
2250                      tan((actualTheta + dTheta) * PI / 180.);
2251        mPadTest[3] = cos((actualPhi + dPhi) * PI / 180.)
2252                      * tan((actualTheta + dTheta) * PI / 180.);
2253
2254        mFwTest[0] = sin(actualPhi * PI / 180.) * tan(actualTheta *
              PI / 180.);
2255        mFwTest[1] = sin((actualPhi + dPhi) * PI / 180.) *
2256                     tan(actualTheta * PI / 180.);
2257        mFwTest[2] = sin(actualPhi * PI / 180.) *
2258                     tan((actualTheta + dTheta) * PI / 180.);
2259        mFwTest[3] = sin((actualPhi + dPhi) * PI / 180.) *
2260                     tan((actualTheta + dTheta) * PI / 180.);
2261
2262        for (Int_t i = 0; i < 4; i++) {
2263          Coord coords2;
2264          coords2.x = mPadTest[i];
2265          coords2.y = mFwTest[i];
2266          coords2.z = 1.;
2267          //rotate back to detector system
2268          coords2.Rotation(-azimuthalAngle, azimuthalRotation,
2269                           -polarAngle, polarRotation);
2270          Double_t mPadNew = coords2.x / coords2.z;
2271          Double_t mFwNew = coords2.y / coords2.z;
2272          //check if extreme slopes in this bin can be measured with
                6 points
2273          //in detector volume
2274          if (N_PAD - height * fabs(mPadNew) <= 0 ||
2275              N_FW - height * fabs(mFwNew) <= 0) {
2276            isCovered = false;
2277            //if one of the middle bins can't be filled, don't
                 fill any of
2278            //them later on
2279            if (Ny == 2)
2280              isMiddleBinCovered = false;
2281          }
2282        }
2283
```

```
2284        //set flux = 0 with high error, if the bin can't be fully
               filled
2285        if (isCovered == false) {
2286          FluxHist->SetBinContent(Nx, Ny, 0);
2287          dFluxHist->SetBinContent(Nx, Ny, 10000000000000000000.)
                 ;
2288          continue;
2289        }
2290        //solid angle for merged bin in the middle
2291        if (Ny == 2) {
2292          omega = 2 * PI * (cos(actualTheta * PI / 180.)
2293                            - cos((actualTheta + dTheta) * PI /
                                  180.));
2294        }
2295        else {
2296          //solid angle
2297          omega = dPhi * PI / 180. * (cos(actualTheta * PI / 180.)
2298                                      - cos((actualTheta + dTheta) *
                                            PI / 180.));
2299        }
2300        //tracking efficiency for track with 6 hits
2301        Double_t trackingEff6 = 1;
2302        //tracking efficiency for track with 5 hits
2303        Double_t trackingEff5Chamber[N_CHAMBER];
2304        Double_t trackingEff5 = 0;
2305        //tracking efficiency is product of chamber efficiencies
2306        for (Int_t j = 0; j < N_CHAMBER; j++) {
2307          ChamberEff[j] = efficiency[j]->GetEfficiency(currentBin);
2308        }
2309        for (Int_t j = 0; j < N_CHAMBER; j++) {
2310          //for 6 hits: multiply chamber efficiencies
2311          trackingEff6 *= ChamberEff[j];
2312          //for 5 hits: current chamber not hit: (1 - current
                 chamber efficiency)
2313          //             all others hit - multiply chamber
                 efficiencies
2314          trackingEff5Chamber[j] = (1 - ChamberEff[(j)%(N_CHAMBER)])
                  *
2315                                   ChamberEff[(j+1)%(N_CHAMBER)] *
2316                                   ChamberEff[(j+2)%(N_CHAMBER)] *
2317                                   ChamberEff[(j+3)%(N_CHAMBER)] *
2318                                   ChamberEff[(j+4)%(N_CHAMBER)] *
2319                                   ChamberEff[(j+5)%(N_CHAMBER)];
2320          //sum up all possible cases for 5 hit chambers
2321          trackingEff5 += trackingEff5Chamber[j];
2322          //if (tracksHist->GetBinContent(Nx,Ny) > 0) cerr << j << "
                 " << ChamberEff[j] << "      "; //<< trackingEff6 << "
                 " << trackingEff5Chamber[j] << " " << trackingEff5 <<
                 " " << efficiency[j]->fPassedHistogram->GetBinContent(
                 efficiency[j]->fPassedHistogram->GetBin(Nx, Ny-1)) << "
                  " << efficiency[j]->fTotalHistogram->GetBinContent(
                 efficiency[j]->fTotalHistogram->GetBin(Nx, Ny-1)) <<
                 endl;
2323        }
2324        //cerr << endl;
```

```
2325        //final efficiency = sum of 6 hits and 5 hits efficiencies
2326        trackingEff = trackingEff6 + trackingEff5;
2327        if (trackingEff == 0)
2328          trackingEff = 1;
2329        //calculate flux with corrections, acceptance is included in
                PolarHist
2330        Double_t flux = PolarHist->GetBinContent(Nx, Ny) / time /
               omega / AREA
2331                        / trackingEff;
2332        //if (tracksHist->GetBinContent(Nx,Ny) > 0) cerr << Ny << "
               " << Nx << " " << trackingEff6 << endl;//<< flux << " "
               << trackingEff << " " << PolarHist->GetBinContent(Nx, Ny)
                << " " << tracksHist->GetBinContent(Nx,Ny) << endl;
2333
2334        FluxHist->SetBinContent(Nx, Ny, flux);
2335        if (Ny == 2) {
2336          if (isMiddleBinCovered)
2337            FluxHist->SetBinContent(Nx, 1, flux);
2338          else {
2339            FluxHist->SetBinContent(Nx, 1, 0);
2340            FluxHist->SetBinContent(Nx, 2, 0);
2341            dFluxHist->SetBinContent(Nx, 1,
                   10000000000000000000.);
2342            dFluxHist->SetBinContent(Nx, 2,
                   10000000000000000000.);
2343            continue;
2344          }
2345        }
2346
2347        //error calculation
2348        //calculate tracking efficiency error
2349        for (Int_t j = 0; j < N_CHAMBER; j++) {
2350          dChamberEff[j] = efficiency[j]->GetEfficiencyErrorLow(
                 currentBin);
2351          //tracking error equals error of current chamber times
2352          //tracking efficiency of the other chambers
2353          if (ChamberEff[j] != 0) {
2354            effErrorProduct[j] = (1 - ChamberEff[(j+1)%(N_CHAMBER-1)
                   ]) *
2355                                 ChamberEff[(j+2)%(N_CHAMBER-1)] *
2356                                 ChamberEff[(j+3)%(N_CHAMBER-1)] *
2357                                 ChamberEff[(j+4)%(N_CHAMBER-1)] *
2358                                 ChamberEff[(j+5)%(N_CHAMBER-1)];
2359          }
2360        }
2361        for (Int_t l = 0; l < N_CHAMBER; l++) {
2362          effErrorSum[l] = 0;
2363          for (Int_t j = 0; j < N_CHAMBER; j++) {
2364            if (j != l)
2365              effErrorSum[l] += effErrorProduct[j];
2366          }
2367          effErrorSum[l] *= dChamberEff[l];
2368        }
2369        dEffSquared = 0;
2370        for (Int_t j = 0; j < N_CHAMBER; j++) {
```

```
2371          dEffSquared += pow(effErrorSum[j],2);
2372        }
2373        dEff = sqrt(dEffSquared);
2374        //calculate error of tracks
2375        if (tracksHist->GetBinContent(Nx, Ny) > 0)
2376          dTracks = sqrt(tracksHist->GetBinContent(Nx, Ny));
2377        //get acceptance for bins
2378        Double_t acceptance = tracksHist->GetBinContent(Nx, Ny) /
2379                              PolarHist->GetBinContent(Nx, Ny);
2380        dFlux = sqrt(pow(dTracks/acceptance/time/omega/AREA/
               trackingEff,2) +
2381                pow(dEff * PolarHist->GetBinContent(Nx, Ny) / time /
                    omega / AREA
2382                / pow(trackingEff,2),2));
2383
2384
2385        //high error for bins, that were not filled
2386        if (PolarHist->GetBinContent(Nx, Ny) == 0)
2387          dFluxRelative = 10000000000000000000.;
2388        else {
2389          dFluxRelative = dFlux / (PolarHist->GetBinContent(Nx, Ny)
               / time
2390                          / omega / AREA / trackingEff);
2391        }
2392        dFluxHist->SetBinContent(Nx, Ny, dFluxRelative);
2393        if (Ny == 2)
2394          dFluxHist->SetBinContent(Nx, 1, dFluxRelative);
2395      }
2396    }
2397
2398    if (isMiddleBinCovered == false) {
2399      for (Int_t Nx = 1; Nx < nBinPhi + 1; Nx++) {
2400        dFluxHist->SetBinContent(Nx, 1, 10000000000000000000.);
2401        dFluxHist->SetBinContent(Nx, 2, 10000000000000000000.);
2402      }
2403    }
2404    cerr << "flux␣and␣error␣calculation␣finished" << endl;
2405
2406    rootFile.Close();
2407
2408  /*
2409    for (Int_t Ny = 2; Ny < nBinTheta + 2; Ny++) {
2410      for (Int_t Nx = 1; Nx < nBinPhi + 1; Nx++) {
2411        if (dFluxHist->GetBinContent(Nx, Ny) > 3.5)
2412          dFluxHist->SetBinContent(Nx, Ny, 0);
2413      }
2414    }
2415    */
2416    ////////////////////////
2417    //graphical output
2418    ////////////////////////
2419
2420    gStyle->SetOptStat(1); //no statistic box
2421
2422    TCanvas* c1 = new TCanvas("c1", "gauss", 0, 10, 1600, 900);
```

```
2423    //custom colormap
2424    Double_t red[9] = {0./255., 61./255., 89./255., 122./255.,
           143./255.,
2425                       160./255., 185./255., 204./255., 231./255.};
2426    Double_t green[9] = {0./255., 0./255., 0./255., 0./255.,
           14./255., 37./255.,
2427                         72./255., 132./255., 235./255.};
2428    Double_t blue[9] = {0./255., 140./255., 224./255., 144./255.,
           4./255.,
2429                        5./255., 6./255., 9./255., 13./255.};
2430    Double_t stops[9] = {0.0000, 0.1250, 0.2500, 0.3750, 0.5000,
           0.6250, 0.7500,
2431                         0.8750, 1.0000};
2432    TColor::CreateGradientColorTable(9, stops, red, green, blue, 99)
           ;
2433    FluxHist->SetContour(99);
2434    dFluxHist->SetContour(99);
2435
2436
2437    c1->Divide(2,1);
2438    c1->cd(1);
2439    c1->cd(1)->SetTheta(90);
2440    c1->cd(1)->SetPhi(-10);
2441    dFluxHist->Draw("SURF1␣POL␣Z");
2442    //cutHist->Draw();
2443    //FluxHist->Draw("SURF1 POL Z");
2444    c1->cd(2);
2445    c1->cd(2)->SetTheta(90);
2446    c1->cd(2)->SetPhi(-10);
2447    //~ c1->SetTheta(90);
2448    //~ c1->SetPhi(-5);
2449    //~ FluxHist->SetLineWidth(0.7);
2450
2451    //sometimes it doesn't fill outer bins with color, this fixes
           this problem
2452    c1->SetFillStyle(3305);
2453    c1->SetFillColor(kBlack);
2454    //gStyle->SetHatchesLineWidth(0.01);
2455    gStyle->SetHatchesSpacing(1000);
2456    gPad->Modified();
2457    gPad->Update();
2458
2459    FluxHist->Draw("SURF1␣POL␣Z");
2460    //tracksHist->Draw("Surf1 Pol Z");
2461    //save .pdf
2462    c1->SaveAs(pdfout.c_str());
2463    //combine the analysis pdf with the polar plot one
2464    sprintf(Command,
2465            "pdfunite␣Results/Mts4Run%d.pdf␣Results/Mts4Run%d_polar.
                  pdf␣temp.pdf",
2466            runToAnalyze, runToAnalyze);
2467    returnCode = system(Command);
2468    sprintf(Command,"mv␣temp.pdf␣Results/Mts4Run%d_complete.pdf",
           runToAnalyze);
2469    returnCode = system(Command);
```

```
2470    sprintf(Command,"rm␣Results/Mts4Run%d_polar.pdf", runToAnalyze);
2471    returnCode = system(Command);
2472    if (returnCode != 0) {
2473     cerr << "temporary␣file␣Results/Mts4Run" << runToAnalyze;
2474     cerr << "_polar.pdf␣couldn't␣be␣removed." << endl;
2475    }
2476
2477    //save histograms to TFile
2478    TFile* outfile = new TFile(rootout.c_str(), "UPDATE");
2479    if (outfile->IsZombie() == true) {
2480      cerr << "ERROR␣OPENING␣OUTPUT␣FILE" << endl;
2481      return;
2482    }
2483
2484    FluxHist->Write();
2485    dFluxHist->Write();
2486    outfile->Close();
2487
2488    //~ ofstream cut("cut.txt");
2489    //~ for (Int_t i = 1; i < 37; i++) {
2490      //~ cut << (-87.5 + ((i-1)*5)) << "\t" << cutHist->
           GetBinContent(i) << "\t" << cutHistError->GetBinContent(i)
           << endl;
2491    //~ }
2492    //~ cut.close();
2493
2494    //save parameters to .txt file
2495    ofstream txt(txtout.c_str());
2496    txt << "Run" << "\t" << runToAnalyze << endl;
2497    txt << "Time" << "\t" << time  << endl;
2498    if (polarRotation == 0) txt << "First␣Rotation␣Axis\tx"  << endl
          ;
2499    if (polarRotation == 1) txt << "First␣Rotation␣Axis\ty"  << endl
          ;
2500    if (polarRotation == 2) txt << "First␣Rotation␣Axis␣\tz"  <<
          endl;
2501    txt << "First␣Rotation␣Angle\t" << polarAngleDegree << endl;
2502    if (azimuthalRotation == 0) txt << "Second␣Rotation␣Axis\tx" <<
          endl;
2503    if (azimuthalRotation == 1) txt << "Second␣Rotation␣Axis\ty" <<
          endl;
2504    if (azimuthalRotation == 2) txt << "Second␣Rotation␣Axis\tz" <<
          endl;
2505    txt << "Second␣Rotation␣Angle\t" << azimuthalAngleDegree << endl
          ;
2506    txt << "nBinPhi\t\t" << nBinPhi << endl;
2507    txt << "nBinTheta\t" << nBinTheta << endl;
2508    txt << "phimin\t\t" << phiMin << "\tphimax\t\t" << phiMax <<
          endl;
2509    txt << "thetamin\t" << thetaMin << "\tthetamax\t" << thetaMax <<
           endl;
2510    txt.close();
2511  }
2512
2513
```

```
2514 void GUI()
2515 //int main(Int_t argc, char *argv [])
2516 {
2517   //TApplication theApp("App", &argc, argv);
2518   mtsMain = new MtsMain();
2519   //theApp.Run();
2520   //return 0;
2521 }
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *https://iktp.tu-dresden.de [Stand: 09.09.2016]*

[2] Büchner, Adam: *Untersuchung der Richtungsabhängigkeit des Myonenflusses im Nierniveaumesslabor Felsenkeller.* 2015

[3] Gergely Gàbor Barnafölfi, Hunor Gergely Melegh Làszlò Olàh Gergely Surànyi Dezsö V. Gergö Hamar H. Gergö Hamar: *Cosmic Background Measurements at a Proposed Underground Laboratory by the REGARD Muontomograph.* 2012. – paper of muon-tomograph

[4] *https://www.hephy.at [Stand: 24.04.2016]*

[5] Brandt, Matthias: *Aufbau und Test eines Wasser-Cherenkov-Detektors zur Demonstration kosmischer Strahlung.* 2010

[6] Sperling, Artur: *Aufbau und Test eines Plastikszintillatordetektors zur Demonstration kosmischer Strahlung.* 2010

[7] Allkofer: *Cosmic Rays on Earth.* Fachinformationszentrum Karlsruhe, 1984

[8] Thiel, Christopher: *Wechselwirkung von Strahlung mit Materie.* 2010

[9] *http://physik.uibk.ac.at [Stand: 20.06.2016]*

[10] *http://www.spektrum.de [Stand: 17.09.2016]*

[11] *https://de.wikipedia.org [Stand: 02.06.2016]*

[12] *https://web-docs.gsi.de [Stand: 15.05.2016]*

[13] *http://images.slideplayer.org [Stand: 07.07.2016]*

[14] László Oláh, Gergy Hamar Hunor Gergely Melegh Gergely Surányi Dezsy V. Gergely Gábor Barnaföldi B. Gergely Gábor Barnaföldi: Cosmic Muon Detection for Geophysical Applications. In: *Hindawi Publishing Corporation* (2013)

[15] *http://physik-begreifen-zeuthen.desy.de [Stand: 17.09.2016]*

[16] Group, REGARD: *User's Manual for the Regard MuonTomograph MTS4*. 2015. – official User's Manual for the Teleskope

[17] D. Varga, G. K. G. Hamar H. G. Hamar: Asymmetric Multi-Wire Proportional Chamber with reduced requirements to mechanical precision. In: *ELSEVIER* (2011)

[18] Gergely Gàbor Barnafölfi, Hunor Gergely Melegh Làszlò Olàh Gergely Surànyi Dezsö V. Gergö Hamar H. Gergö Hamar: Portable cosmic muon telescope for environmental applications. In: *ELSEVIER* (2012)

Selbstständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Dresden, 12. Dezember 2016