# Identifikation von Wolken im Sichtfeld der Fluoreszenzdetektoren des Pierre-Auger-Observatoriums

Bachelor-Arbeit
zur Erlangung des akademischen Grades
Bachelor of Science
(B.Sc.)

der Universität Siegen


Department Physik

vorgelegt von
*Silas Rodekamp*

Januar 2018

# Inhaltsverzeichnis

# 1. Einführung

Am in Argentinien gelegenen Pierre-Auger-Observatorium werden hochenergetische kosmische Luftschauer vermessen mit dem Ziel mehr Kenntnisse über die Energie und die Zusammensetzung der primären kosmischen Strahlung und somit mehr Wissen über die Quellen dieser Teilchen zu sammeln. Zwei verschiedene Detektortypen werden benutzt, um die kosmische Strahlung auf dem Grund bzw. in der Atmosphäre zu untersuchen. Diese Bachelorarbeit wertet Daten der Detektoren aus, die in die Atmosphäre schauen. Insbesondere ist das Ziel der Bachelorarbeit den Einfluss von Wolken auf die Daten zu untersuchen. Es stellt sich die Frage, wie groß der Einfluss von sich zwischen Luftschauer und Teleskop befindlichen Wolken ist und wie möglichst geschickt die Daten der Luftschauer, in denen Wolken eine Rolle spielen, von denjenigen getrennt werden können, in denen definitiv kein Wolkeneinfluss zu erwarten ist.

Im ersten Teil werden die physikalischen Grundlagen der kosmischen Strahlung näher erläutert. Zu diesen Grundlagen gehören das Energiespektrum und die Zusammensetzung der kosmischen Strahlung. Außerdem wird erklärt, wie ausgedehnte Luftschauer in der Atmosphäre entstehen und welche von der Energie und Teilchentyp abhängigen Größen sich zur Charakterisierung der Luftschauer eignen. Im nächsten Kapitel wird dann speziell auf das Pierre-Auger-Observatorium eingegangen. Dort werden allgemeine Informationen über das Experiment genannt. Da die Fluorenszenz-Detektoren und die Wolkenkameras eine wichtige Komponente dieser Bachelorarbeit sind, sollen sie und auch die Rekonstruktion der Luftschauer aus den Fluorenszenz-Detektor Daten detaillierter erläutert werden.
Im vierten Kapitel befindet sich die eigentliche Analyse der Daten. Verschiedene Datensätze werden für verschiedene Fragestellungen gegenübergestellt, um zunächst die Frage zu beantworten, wie groß der Einfluss der Wolken auf die Daten ist, und dann eine Abschätzung zu geben, ob die Daten der Wolkenkameras allein ausreichen, um die atmosphärischen Bedingungen über dem Areal zu überwachen. Die Zusammenfassung der Ergebnisse befindet sich im fünften Kapitel.

# 2. Kosmische Strahlung

Nachdem Ende des 19. Jahrhunderts die radioaktive Strahlung entdeckt wurde und bereits erste Experimente mit radioaktiven Elementen gemacht wurden, beobachtete Victor Hess auf einem Ballonflug, dass mit zunehmender Höhe ionisierende Strahlung zunimmt. Die schon zuvor auf der Erde entdeckte radioaktive Strahlung konnte nicht die Ursache dieser Beobachtung sein, denn diese müsste mit zunehmendem Abstand vom Erdboden abnehmen. Die Quelle der von Hess beobachteten ionisierenden Strahlung musste also kosmischen Ursprungs sein. Seitdem ist die sogenannte kosmische Strahlung Gegenstand wissenschaftlicher Forschung. Durch ein genaueres Verständnis von Energiespektrum und Identität der primären kosmischen Teilchen lassen sich zuverlässigere Rückschlüsse auf die Herkunft der Teilchen und somit ein tieferes Verständnis von den Vorgängen in unserem Universum ableiten.

## 2.1. Energiespektrum

Jede Sekunde dringt eine große Zahl aus dem All kommender Teilchen in die Erdatmosphäre ein. Die Häufigkeit solcher primärer kosmischen Teilchen nimmt mit zunehmender Energie drastisch ab. In Abb. 2.1 ist der differentielle Fluss an Teilchen als Funktion von der Energie aufgetragen. Für jede Größenordnung in der Energie nimmt der Fluss um ca. drei Größenordnungen ab. Von den niederenergetischen Teilchen im GeV-Bereich ($10^9$ eV) treffen hunderte Teilchen pro Sekunde und pro Quadratmeter auf die Atmosphäre, während bei den ultrahochenergetischen Teilchen im EeV-Bereich ($10^{18}$ eV) im Schnitt nur noch ein Teilchen pro Quadratkilometer und Jahrhundert zu erwarten ist. Bei diesen Größenordnungen müssen unterschiedliche Methoden zur Messung des Spektrums angewendet werden. Für kleine Energien werden Ballons oder Satelliten für Messungen benutzt. Obwohl sie nur eine kleine Fläche abdecken können, reicht das auf Grund des großen Flusses an primären kosmischen Teilchen aus, um genügend Daten zu sammeln. Im Gegensatz dazu sind ultrahochenergetischen Teilchen so selten, dass eine viel größere Fläche benötigt wird. Die von ihnen ausgelösten Luftschauer besitzen genügend Energie, um den Erdboden zu erreichen. Das wird ausgenutzt, um auf dem Grund großflächig die in der Atmosphäre entstehenden Luftschauer zu entdecken und zu vermessen. Denn auch dadurch können dann Rückschlüsse auf das primäre Teilchen gezogen werden. Die bisher höchste Energie eines primären kosmischen Teilchens wurde 1991 von dem Fly's-Eye-Detektor in Utah (USA) mit 320 EeV gemessen. Diese Energie ist über 7 Größenordnungen größer als die Energien, die im Large Hadron Collider in Genf am CERN erreicht werden können.

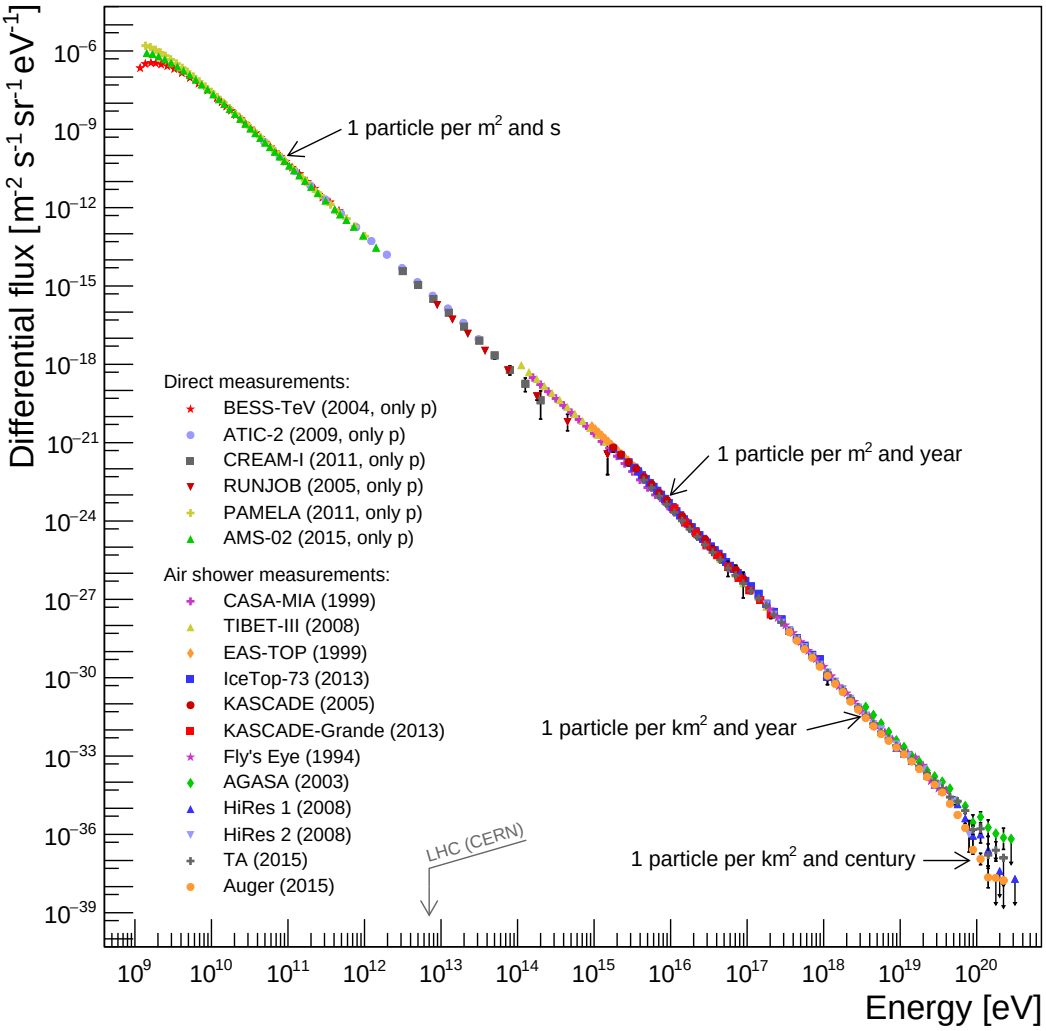



Abbildung 2.1: Das Spektrum der primären kosmischen Strahlung [1]. Der differentielle Fluss an Teilchen ist als Funktion der Energie dargestellt. Beide Achsen sind logarithmisch. Die Messwerte stammen von den verschiedensten Experimenten, die im niederenergetischen Bereich direkt den Protonenfluss und die im hochenergetischen Bereich indirekt und den ganzen Fluss an Teilchen gemessen haben. Der Übergang zwischen direkten und indirekten Messungen ist bei ca. $10^{14}$ eV.

Der Fluss an kosmischer Strahlung $\phi$ folgt einem Potenzgesetz.

$$\phi(E) \propto E^{-\gamma} \tag{2.1}$$

In der Abbildung 2.1 sind zwei Strukturen sichtbar, auf die von der Forschung ein besonderes Augenmerk gelegt wird. Zum Einen lässt sich ab einer Energie von etwa $4{\cdot}10^{15}$ eV eine leicht größere Steilheit im Spektrum beobachten. Der spektrale Index $\gamma$ ändert sich bei dieser Energie von $\gamma = 2,7$ auf $\gamma = 3$. Dieser Punkt wird mit „Knie“ bezeichnet. Ein zweiter Knick, der sogenannte „Knöchel“, ist bei einer Energie von $4{\cdot}10^{18}$ eV

vorhanden, in dieser Abbildung jedoch kaum sichtbar. Der spektrale Index wird wieder etwas kleiner als drei. Sowohl für den Ursprung des Knies, als auch für den des Knöchels gibt es eine ganze Reihe von Theorien. Genauere Informationen dazu können in [2] gefunden werden.

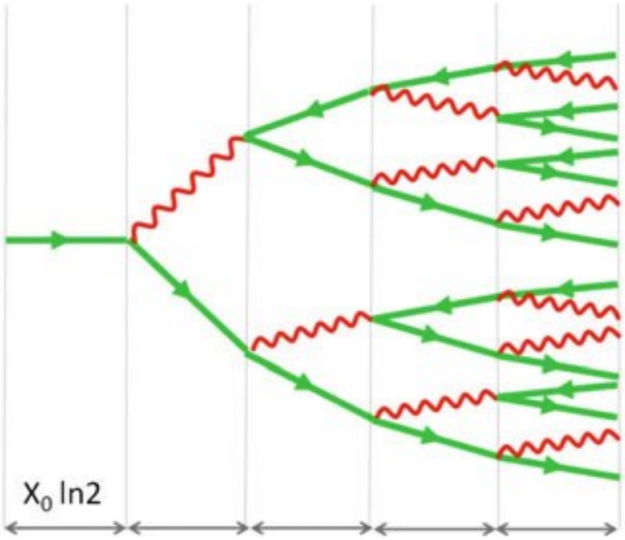
## 2.2. Zusammensetzung

Die primäre kosmische Strahlung setzt sich zu 2% aus Elektronen und zu 98% aus Atomkernen zusammen, diese wiederum setzen sich zu 86% aus Protonen, zu 12% aus alpha-Teilchen und zu 3% aus schweren Kernen zusammen [3]. Während die Zusammensetzung bei niedrigen Energien bis ca. 100 TeV der Häufigkeitsverteilung der Atomkerne ähnlich der Elementverteilung in unserem Sonnensystem ist, ist die Frage der Zusammensetzung bei hohen Energien noch ungeklärt. Die Schwierigkeit besteht darin bei indirekten Messungen den Teilchentyp des primären Teilchens zu bestimmen. Wäre dies möglich, so könnten aus diesen Informationen Rückschlüsse über die Herkunft der Teilchen gezogen werden [2].

## 2.3. Ausgedehnte Luftschauer in der Atmosphäre

Dringt ein primäres kosmisches Teilchen mit der Energie $E_0$ in die Erdatmosphäre ein, so wird es mit einem anderen Teilchen in der Erdatmosphäre wechselwirken und an dieses Teilchen einen Teil seiner Energie abgeben. Sowohl das Primär- aus auch das Sekundärteilchen werden wiederum mit neuen Teilchen in der Atmosphäre wechselwirken. Auf diese Weise entsteht eine ganze Lawine von verschiedenen Teilchen, die erst dann wieder abklingt, wenn die Energie eines einzelnen Teilchens nicht mehr für eine weitere Wechselwirkung ausreicht. Dieser Prozess wird stark vereinfacht im Heitler-Modell [4] beschrieben. Es geht davon aus, dass ein Primärteilchen die Hälfte seiner Anfangsenergie an ein Sekundärteilchen abgibt. Beide Teilchen geben wiederum die Hälfte ihrer Energie an jeweils ein neues Teilchen ab. Dieser Prozess setzt sich fort, bis das Schauermaximum erreicht ist, bei dem $N$ Teilchen im Schauer jeweils eine Energie von $1/N$ der Primärenergie haben. Dies entspricht der kritischen Energie $E_{\mathrm{crit}}$. Aus diesem einfachen Modell, dessen Prinzip in Abb. 2.2 nochmals skizziert ist, folgt eine Gesamtteilchenanzahl $N_{\max} = E_0 \cdot \ E_{\mathrm{crit}}^{-1}$ und die atmosphärische Tiefe des Schauermaximums von

$$X_{\max} = \frac{\lambda}{\ln(2)} \cdot \ln\left(\frac{E_0}{E_{\mathrm{crit}}}\right) \tag{2.2}$$




| | | | | | |
|---|---|---|---|---|---|
| k = | 1 | 2 | 3 | 4 | 5 |
| N = | 2 | 4 | 8 | 16 | 32 |
| $E / E_0$ = | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 |

Abbildung 2.2: $X_0$ ist eine Strahlungslänge, k gibt die Anzahl an Strahlungslängen an, N ist die Teilchenanzahl, E ist die Energie pro Teilchen und $E_0$ ist die Energie des Primärteilchens. Dargestellt ist ein Elektron/Positron als Primärteilchen. In dieser Skizze ist es so dargestellt, dass Photonen Elektron-Positron-Paare und Elektronen/Positronen Bremsstrahlung erzeugen. Nach jeder weiteren Strahlungslänge verdoppelt sich die Teilchenanzahl und halbiert sich die Energie pro Teilchen. [5]

Obwohl dieses Modell stark vereinfacht ist, geht aus dem Modell hervor, dass die maximale Teilchenanzahl proportional zu der Primärenergie $E_0$ und die atmosphärische Tiefe des Schauermaximums $X_{\max}$ proportional zu $\ln(E_0)$ ist. Diese Zusammenhänge sind auch experimentell bestätigt worden. Computer können die zufälligen Wechselwirkungen von Teilchen in der Atmosphäre auch mit Monte-Carlo-Methoden simulieren. Auf diese Weise können Luftschauer simuliert werden, um beispielsweise Analyseverfahren zu erstellen und zu verbessern.

# 3. Das Pierre-Auger-Observatorium

Das Pierre-Auger-Observatorium ist das weltweit größte Observatorium zur Messung von kosmischer Strahlung. Der Bau dieses Observatoriums begann 2004. Es umfasst 1600 SD Stationen (SD : surface detector) und 4 FD Standorte (FD : fluorescence detector) [6]. Beide Detektortypen werden genutzt, um Schauer zu rekonstruieren. Da für diese Arbeit die FD Standorte wichtiger sind, soll auf diese etwas genauer eingegangen werden.

## 3.1. Der Oberflächendetektor

Das Pierre-Auger-Observatorium umfasst 1600 SD Stationen. Die Stationen beinhalten jeweils einen 12 $m^3$ großen, mit reinem Wasser gefüllten Tank, in dem das von schnellen Teilchen produzierte Cherenkov-Licht mittels dreier Photomultiplier beobachtet werden kann. Um die Energieversorgung zu gewährleisten, verfügt jede SD Station über ein eigenes Solar Panel. Jede SD Station funktioniert demnach völlig autark und sie ist durchgängig aktiv. Die Stationen sind in einer hexagonalen Struktur mit einem jeweiligen Abstand von 1,5 km angeordnet (siehe Abb. 3.1). Auf diese Weise wird eine Fläche von 3000 $km^2$ abgedeckt. Der Abstand zwischen den Tanks ist so gewählt, dass Luftschauer im hochenergetischen Bereich optimal beobachtet werden können und die Fläche ist so groß, weil dann zumindest mit einigen hochenergetischen Teilen pro Jahr gerechnet werden kann. Bei diesem Abstand wird die volle Triggereffizienz für Schauer mit einem Zenitwinkel kleiner als 60° ab einer Energie von $3{\cdot}10^{18}$ eV erreicht. Wird ein Trigger an einer einzelnen Station ausgelöst, so wird das dem CDAS (central data aquisition system) zusammen mit einem Zeitstempel weitergeleitet. Bekommt das CDAS von mehreren Stationen gleichzeitig ein Signal, so kann das gemessene Ereignis als ein Luftschauer eingestuft werden und die Stationen übermitteln alle zu speichernden Daten über eine Radiowellen-Schnittstelle an das CDAS.

## 3.2. Der Fluoreszenzdetektor

Bei einem Luftschauer können geladene Teilchen Stickstoffmoleküle in der Atmosphäre anregen. Die zerfallen wieder in ihren Grundzustand unter Aussendung von ultraviolettem Licht im Wellenlängenbereich von 300-430 nm. Die Intensität des ausgesandten Lichts ist proportional zu der in der Atmosphäre deponierten Energie. Das heißt, dass die Intensität direkt mit der Anzahl der geladenen Teilchen in dem Luftschauer korreliert und somit von der Energie des Primärteilchens abhängig ist. Um dieses Licht zu beobachten, befinden sich an den Seitenrändern des Areals vier Standorte für Fluoreszenz-Detektoren, die jeweils über 6 Teleskope verfügen. Die Standorte sind so angeordnet, dass die ganze Atmosphäre über dem Areal beobachtet werden kann. Ein zentral gelegenes Ereignis kann ab einer Energie von $10^{19}$ eV von allen vier Seiten aus beobachtet werden. Jedes Teleskop deckt 30° im Azimutal- und 30° im Höhenwinkel ab. Ein Standort deckt dann insgesamt einen Azimutalwinkel von 180° ab (vgl. Abb. 3.1).

Ein Teleskop besteht aus einem $13m^2$ großen Spiegel, in dessen Brennpunkt eine Einheit mit 440 Photomultipliern ist, die jeweils einem Pixel entsprechen. Um die gesamten 30°· 30° im Sichtfeld des Teleskops abzudecken, deckt jeder Pixel einen Raumbereich im Sichtfeld von 1,5°· 1,5° ab.

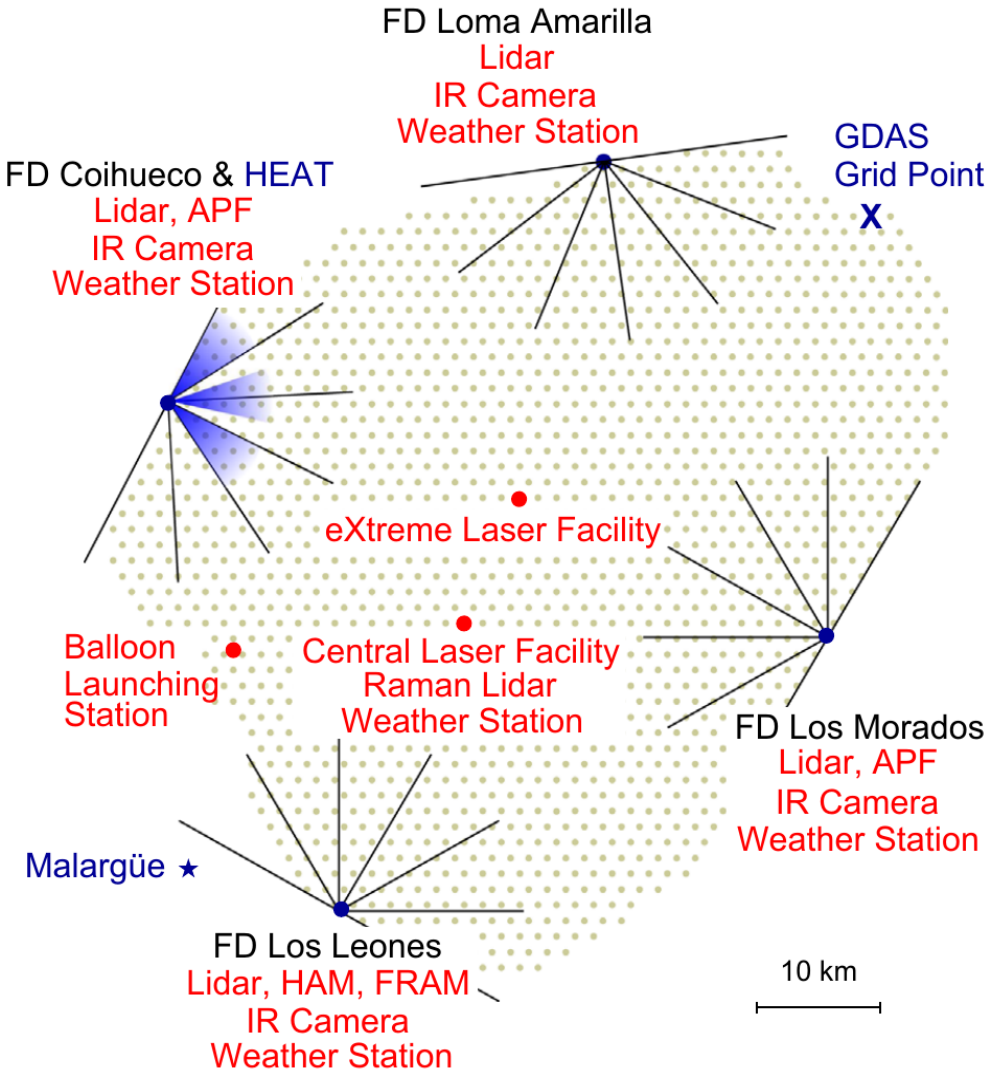



Abbildung 3.1: Stationen zur Überwachung von atmosphärischen Bedingungen [6]. Die grauen Punkte entsprechen einzelnen SD Stationen. Die komplette Atmosphäre über diesem Areal wird von den sich an den vier Seiten befindlichen Standorten der FD überwacht. Eingezeichnet sind auch die Sichtfelder der einzelnen Teleskope an jedem Standort. Außerdem befinden sich ein Lidar System, eine Wetterstation und eine Infrarot-Wolkenkamera an jedem Standort. Mitten auf dem Areal befinden sich eine Ballon-Station, die XLF und die CLF.

Die FD sind nur in dunklen und mondlosen Nächten nutzbar, da sonst das Licht aus anderen Quellen das Leuchten der Stickstoffmoleküle überdeckt. Der Duty-Cycle der FD beträgt 13%. Ein Luftschauer muss die Bedingung erfüllen, dass er die für einen Luftschauer zwingend notwendige Spur mit passender Zeitcharakteristik hat (siehe Abb. 4.3). Sind die Bedingungen für die Trigger erfüllt, werden die gemessenen Daten wie bei den

SD an das CDAS weitergeleitet. Mittels dem GPS Zeitstempel werden die Daten desselben Luftschauers von FD und SD miteinander verbunden, sodass sie als ein sogenanntes Hybrid-Ereignis für die Offline-Analyse gespeichert werden. Eine genauere Beschreibung der Trigger und der Kalibration der FD ist in [7] zu finden.
Für eine korrekte Interpretation der Intensität des gemessenen Lichts und damit einer exakten Berechnung der Energie des Primärteilchens, ist eine genaue Kenntnis der momentanen Atmosphärenbeschaffenheit erforderlich. Es können beispielsweise Aerosole oder Wolken das Fluoreszenzlicht des Schauers streuen oder absorbieren. In der Abb. 3.1 [8] sind die verschiedenen Stationen eingezeichnet, die dem Überwachen der atmosphärischen Bedingungen dienen. Wetterstationen messen regelmäßig die Temperatur, den Luftdruck und die Luftfeuchtigkeit auf dem Grund. In der Vergangenheit wurden auch regelmäßig Wetterballons gestartet, um die Temperatur etc. auch in größerer Höhe zu messen. Heute wird auf Modelle zurückgegriffen, die aus den Werten und Daten vergangener Jahre monatliche Atmosphären in der Pampa modellieren. Da diese Parameter über eine größere Fläche als konstant angesehen werden können, kann damit der Einfluss auf die Daten analytisch berechnet werden. Anders verhält es sich bei Aerosolen und Wolken, die lokal über dem Areal ganz unterschiedliche Bedingungen für das Durchqueren des Lichts schaffen können. Um die vertikale optische Tiefe von Aerosole richtig zu erfassen, existieren die Central Laser Facility (CLF) und die eXtreme Laser Facility (XLF), sowie vier Light Detection and Ranging (Lidar) Systeme. Weitere Informationen über die Aerosole misst das Aerosol Phase Function Monitor (APF) System, der Horizontal Attenuation Monitor (HAM) und der FotometricRobotic Atmospheric Monitor (FRAM). Weitere Informationen dazu sind in [8] erhältlich. Neben Aerosolen, die nicht sichtbar sind, können auch Wolken die Daten verfälschen, bzw. die Datennahme sogar ganz unmöglich machen. Dafür existiert die Möglichkeit auf Satellitendaten zurückzugreifen (GOES). Eine weitere Möglichkeit sind die Lidars, die zusätzlich auch in der Lage sind die Höhe der Wolken zu bestimmen. Die letzte Möglichkeit, die im nächsten Abschnitt etwas genauer beschrieben wird, sind die Infrarot-Wolkenkameras, die an den vier Standorten der FD installiert sind.

**Die Infrarot-Wolkenkameras** können die geringfügig wärmeren Wolken vom klaren Nachthimmel unterscheiden. Alle fünf Minuten wird ein neues Bild von dem Sichtfeld des jeweiligen FD gemacht und alle 15 Minuten wird der ganze Nachthimmel nach Wolken abgescannt. Eine Wolkenmaske wird dann erstellt, indem die Bilder, die die Wolkenkameras alle fünf Minuten schießen und die entsprechenden Pixel des FD übereinandergelegt werden, sodass anschließend gesagt werden kann, welcher Bedeckungsgrad an Wolken in jedem Pixel des FD ist. Die folgende Abb. 3.3 soll das verdeutlichen.

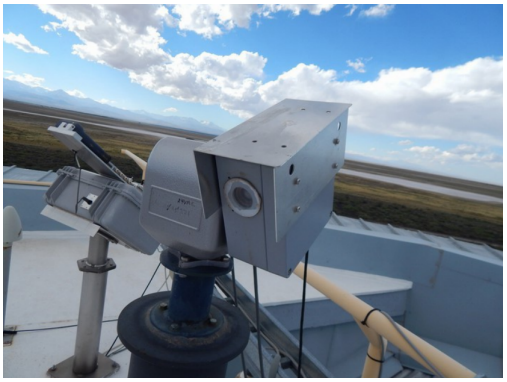


Abbildung 3.2: Die Infrarot-Wolkenkamera ist drehbar und neigbar befestigt. Sie befindet sich direkt beim FD in Los Leones. [9]

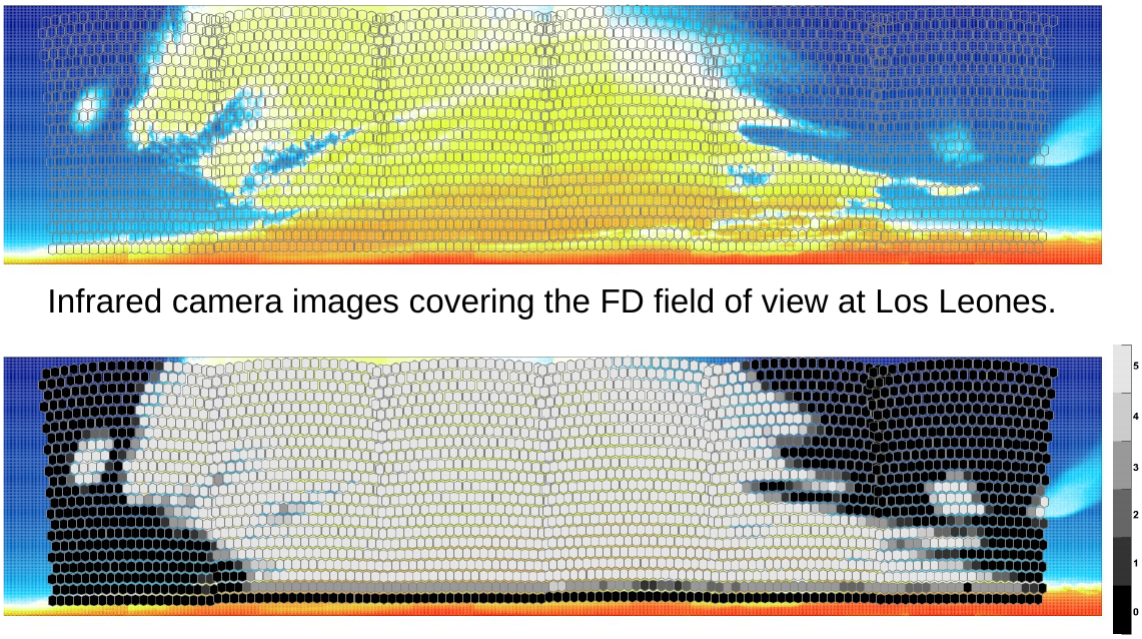



Abbildung 3.3: In dem oberen Bild ist ein Infrarotbild einer Kamera zu sehen. Die wärmeren Wolken sind in gelblichen-rötlichen Farbtönen zu sehen. Im zweiten Schritt wird nun dieses Bild, dass die Wolkenbedeckung anzeigt, mit den Pixeln, d. h. dem Sichtfeld des FD übereinander gelegt und dem Bedeckungsgrad eine Zahl zwischen 0 (klarer Himmel) und 5 (völlig bedeckt) zugeordnet[9]. Der Bedeckungsgrad 5 entspricht im weiterem Verlauf einer durchschnittlichen Wolkenbedeckung von 100% bzw. 1.

Sind zu viele Wolken im Sichtfeld des FD, der den Luftschauer beobachtet hat, oder liegen sogar die Spur des Luftschauers und eine Wolke in der 2D-Projektion aus Sicht des FD übereinander, so muss dies in der Auswertung der Ergebnisse berücksichtigt

werden. Denn es ist ein Einfluss der Wolken auf die Daten zu erwarten, solange sich die Wolke aus Sicht des FD nicht hinter dem Luftschauer befunden hat. Es kann entweder versucht werden den Einfluss der Wolken auf die Daten herauszurechnen, oder es muss versucht werden, die Daten der Ereignisse mit Wolkeneinfluss effizient von den Daten der Ereignisse ohne Wolkeneinfluss zu trennen. In dieser Arbeit wird der zweite Ansatz genutzt.

## 3.3. Rekonstruktion der FD Daten

Die vollständige Rekonstruktion eines Schauers aus den FD Daten besteht aus den folgenden drei Schritten.

1. geometrische Rekonstruktion des Schauers
2. Rekonstruktion des longitudinalen Schauerprofils
3. Rekonstruktion der Energie des primären kosmischen Teilchens

Diese drei grundlegenden Schritte sind bei allen Verfahren zur Rekonstruktion vorhanden. Das Verfahren variiert jedoch, je nachdem, ob ein SD-Ereignis, ein FD-Ereignis oder ein Hybrid-Ereignis vorliegt. Da in dieser Bachelorarbeit ausschließlich Hybrid-Ereignis zur Analyse verwendet wurden, soll sich im Folgenden darauf beschränkt werden.

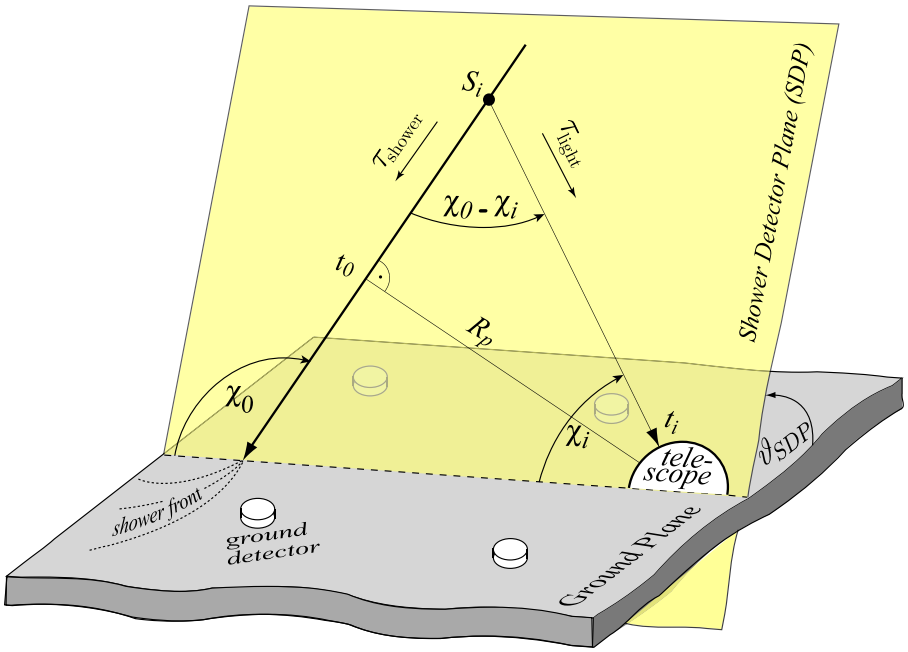



Abbildung 3.4: Dies ist eine Skizze zur Schauergeometrie [10]. Eingezeichnet sind ein Teleskop und die Achse eines Luftschauers, der den Erdboden getroffen hat. Zunächst wird die Detektor-Schauer-Ebene berechnet (SDP). Die anderen Parameter werden im Text erklärt.

Die geometrische Rekonstruktion des Schauers setzt den Auftreffpunkt des Schauers und die Schauerachse fest. Zuerst wird mittels der Spur (siehe Abb. 3.5) der getriggerten Pixel im Teleskop die Schauerachse-Detektor-Ebene (SDP) festgelegt. Die SDP legt den Winkel $\theta_{SDP}$ fest. Damit nun die genaue Position der Schauerachse in dieser Ebene bestimmt werden kann, muss die Zeitinformation der getriggerten Pixel ausgewertet werden. In diese Information fließt die Entfernung von Schauerachse zu Detektor und auch dessen Winkel zum Grund mit ein. Der Referenzpunkt auf der Schauerachse, der die kürzeste Entfernung $R_p$ zum Detektor hat, wird zum Zeitpunkt $t_0$ erreicht. Der Luftschauer bewegt sich mit nahezu Lichtgeschwindigkeit c. Da die Zeit ausgerechnet werden soll, zu dem das Licht den Detektor erreicht, muss die Zeit, die aus der Entfernung eines Punktes $S_i$ zum Teleskop resultiert ($\tau_{\text{light,i}}$), addiert werden zu der Zeit, an dem die Schauerfront den Punkt $S_i$ erreicht ($\tau_{\text{shower,i}}$), in Referenz gesetzt zu der Zeit $t_0$. Wird nun davon ausgegangen, dass die Zerfallsdauer der angeregten Stickstoffatome sehr kurz ist, kann die Ankunftszeit des Lichts am Detektor durch folgende Gleichung beschrieben werden:

$$t_i = t_0 - \tau_{\text{shower,i}} + \tau_{\text{light,i}} \tag{3.1}$$

$$= t_0 + \frac{R_p}{c} \tan\left(\frac{\chi_0 - \chi_i}{2}\right) \tag{3.2}$$

Die in dieser Gleichung benutzten Größen sind im Text bereits erklärt oder in Abb. 3.4 ersichtlich. Die exakte Lage der Schauerachse in der Schauer-Detektor-Ebene errechnet sich aus einem Fit der gemessenen Zeitpunkte. Der Winkel $\chi_0$, die Zeit $t_0$ und die Distanz $R_p$ werden so angepasst, dass die errechneten Zeitpunkte $t_i$ möglichst gut mit den gemessenen übereinstimmen. Zuletzt wird lediglich noch der Auftreffpunkt des Schauers auf dem Erdboden als Schnittpunkt zwischen Schauerachse und Erdboden bestimmt und dann ist die geometrische Rekonstruktion des Schauers vollständig.

Der nächste Schritt besteht dann darin, das longitudinale Profil des Schauers zu erstellen. Dazu werden die Signale, die in den einzelnen Pixeln der Spur ankommen, in einen Photonenfluss umgerechnet. Atmosphärische Bedingungen, sowie auch Eigenschaften der Photomultiplier und der Kamera sind in diesem Photonenfluss mit eingerechnet. Daraus ergibt sich dann ein Profil, das die momentane Änderung der Energie $^{dE}/_{dX}$ beschreibt. An dieses Profil wird nun eine sogenannte Gaisser-Hillas-Funktion gefittet (Abb. 3.5).

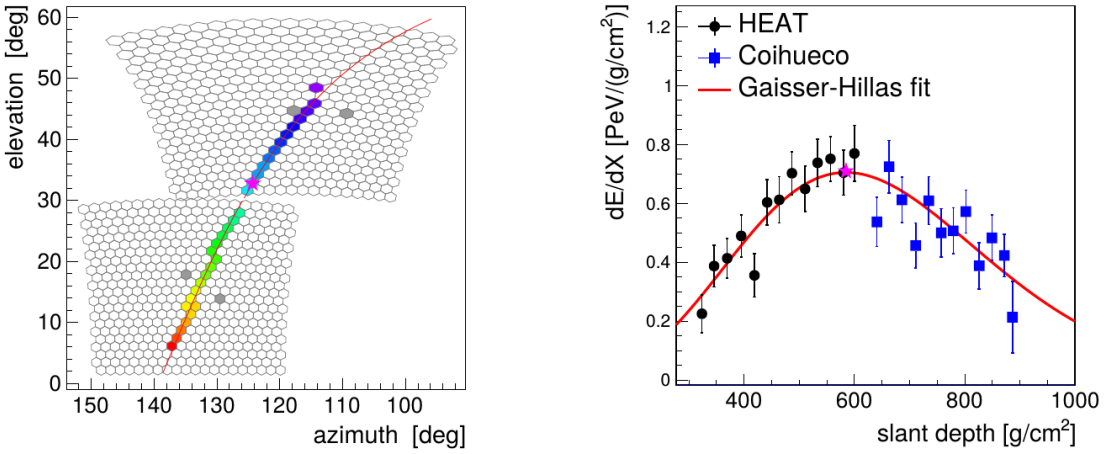



Abbildung 3.5: Im linken Bild ist das Sichtfeld eines Teleskops von Coihueco und der niederenergetischen Erweiterung HEAT (high elevation auger telescope) und die Spur eines Luftschauers zu sehen. Die unterschiedlichen Farben der Pixel stehen für die Zeitpunkte, an denen die Signale am Detektor angekommen sind (Blau - früh, Rot - spät). Die Signale von den Pixeln werden umgerechnet in einen momentanen Energieverlust als Funktion der atmosphärischen Tiefe (rechts). Es ergibt sich ein typisches Profil, an das eine Gaisser-Hillas-Funktion gefittet wird. Der Stern kennzeichnet $X_{\max}$ .[11]

Diese beschreibt die longitudinale Entwicklung eines Luftschauers, abhängig von der atmosphärischen Tiefe $X$ , der mittleren freien Weglänge $\lambda$, dem Wechselwirkungspunkt des primären kosmischen Teilchens $X_0$ und der maximalen Teilchenzahl beim Schauermaximum $N_{\max}$.

$$N(X) = N_{\max}\left(\frac{X - X_0}{X_{\max} - X_0}\right)^{\frac{X_{\max} - X_0}{\lambda}} \exp\left[\frac{X_{\max} - X_0}{\lambda}\right] \tag{3.3}$$

Aus dem Fit dieser Funktion kann dann direkt die atmosphärische Tiefe des Schauermaximums $X_{\max}$ abgelesen werden. Dieses $X_{\max}$ wird im weiteren Verlauf der Bachelorarbeit als die Variable dienen, an Hand derer der Unterschied zwischen Ereignissen mit und ohne Wolken beurteilt werden soll. Da das $X_{\max}$, d. h. die atmosphärische Tiefe des Schauermaximums in g·cm$^{-2}$, von der Energie des primären kosmischen Teilchens abhängig ist, wird auch die Energie in der Analyse dieser Arbeit berücksichtigt werden.

Die Energie des primären kosmischen Teilchens wird im letzten Schritt der Rekonstruktion des Schauers bestimmt. Die als Licht in die Atmosphäre abgegebene Energie $E_{cal}$ ist durch die Integration der Gaisser-Hillas-Funktion gegeben. Diese kalorimetrische Energie muss mit einem Korrekturfaktor multipliziert werden, der die „unsichtbare“ Energie berücksichtigt. Da Neutrinos beispielsweise fast nicht wechselwirken, kann der Detektor die Energie, die auf sie übertragen wird, nicht in Form von Licht „sehen“. Dieser Korrekturfaktor liegt im Regelfall, d. h. bei hadronischen Schauern bei etwa 10%.

Aus dem nun vollständig rekonstruierten Luftschauer können nun Rückschlüsse auf Teilchentyp und Herkunft des Teilchens gezogen werden.

Insbesondere bei Luftschauern mit höherer Energie ist es auch möglich, dass dieser von mehreren FD Standorten aus gesehen wird. Es handelt sich in solchen Fällen um sogenannte Stereoereignisse, Tripelereignisse oder falls ein Schauer von allen 4 Standorten beobachtet wurde, um ein Quadrupelereignis. In dem in dieser Analyse benutzten Datensatz sind etwa 1% der Ereignisse Stereoereignisse. Die anderen beiden Kategorien sind noch deutlich seltener. Bei solchen Mehrfachereignissen kann ein Luftschauer von mehreren Standorten aus unabhängig voneinander rekonstruiert werden. Dadurch ist es möglich die einzelnen Standorte in ihrer Kalibrierung aufeinander abzugleichen.

## 3.4. Ergebnisse bisheriger $X_{\mathrm{max}}$ -Analysen

Die $X_{\mathrm{max}}$ -Analyse wird verwendet, um die Zusammensetzung der primären kosmischen Strahlung zu erforschen. Die Unterscheidung von Protonen und Eisenkernen, d. h. schweren Kernen, spielt dabei die zentrale Rolle. In der $X_{\mathrm{max}}$ -Analyse werden für einen entsprechenden Datensatz an Ereignissen die folgenden Schritte durchgeführt. Das $X_{\mathrm{max}}$ ist energieabhängig und daher werden die Ereignisse ihrer Energie entsprechend sortiert. Die Intervallgrenzen für die Sortierung sind $10^{17,8}$ eV, $10^{17,9}$ eV, ... , $10^{19,5}$ eV. Für jedes der 18 Energiebins wird nun ein $X_{\mathrm{max}}$ Histogramm erstellt.

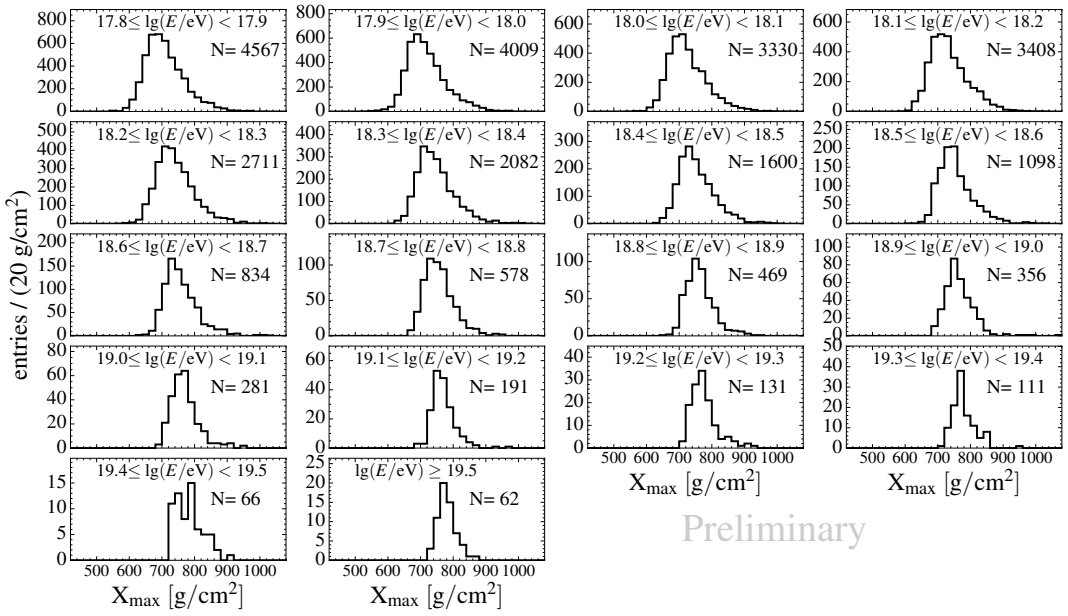



Abbildung 3.6: 18 Histogramme für $X_{\mathrm{max}}$ , je ein Energieintervall [11]. In jedem Histogramm steht das jeweilige Energieintervall und die Anzahl der Einträge in diesem Histogramm.

Der Mittelwert jedes Histogramms wird nun genutzt,um damit den linken Graph in Abb. 3.7 zu erstellen. Der „RMS“-Wert, der die Breite der Verteilung angibt, wird genutzt, um den rechten Graph zu erstellen. Auf diese Art wird $\langle X_{\mathrm{max}} \rangle$ und $\sigma$ ($X_{\mathrm{max}}$ ) abhängig

von der Energie erhalten und daraus lassen sich Rückschlüsse auf die Zusammensetzung der primären kosmischen Strahlung ziehen. Dies ist im Detail hier [11] nachzulesen.

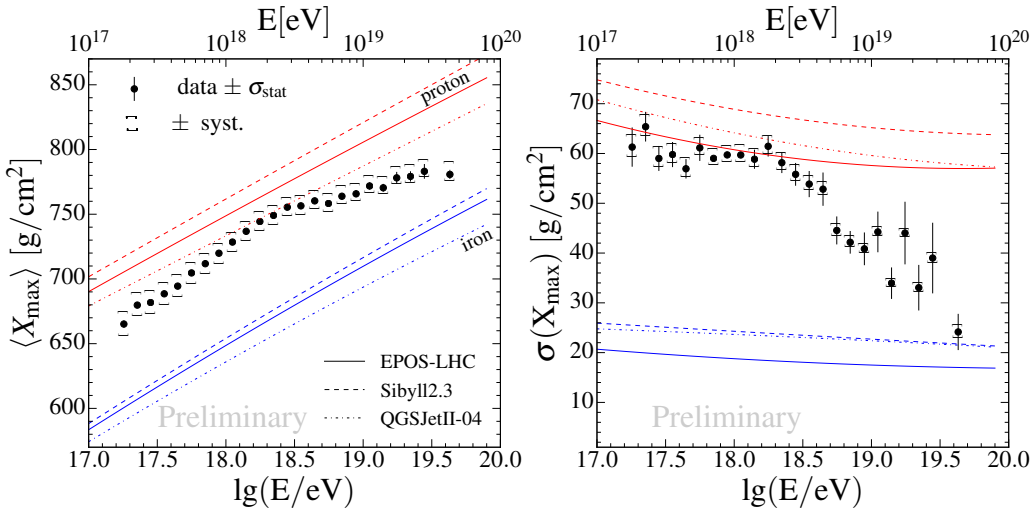



Abbildung 3.7: Das Ergebnis für $\langle X_{\max} \rangle$ (links) und das $\sigma(X_{\max})$ der Verteilung (rechts) als Funktion der Energie [11]. Die zusätzlichen Linien im Bild geben den Verlauf an, der bei Protonen bzw. Eisenkernen aus verschiedenen Modellen errechnet worden ist.

In Abb. 3.7 ist der Verlauf von $\langle X_{\max} \rangle$ (links) und $\sigma(X_{\max})$ (rechts) als Funktion der Energie zu sehen. Bis zu einer Energie von $10^{18,5}$ eV ist die Tendenz eher zu einer leichteren Zusammensetzung und bei höheren Energien deuten die Messwerte wieder in Richtung einer schwereren Zusammensetzung der primären kosmischen Strahlung.

# 4. Analyse

In diesem Kapitel werden die Ergebnisse der Analyse vorgestellt. Zunächst wird das Vorgehen für die $X_{\max}$ -Analyse beschrieben und anschließend werden einige ausgewählte Datensätze damit untersucht und verglichen. Danach soll mit weiteren Fragestellungen untersucht werden, inwiefern die von den Wolkenkameras gesammelten Informationen ausreichen, um die atmosphärischen Bedingungen zu überwachen.

## 4.1. Beschreibung der Datenextraktion aus den ADST-Dateien und der Wolkendatenbank

Entlang des Flussdiagramms in Abb. 4.1 soll in diesem Abschnitt beschrieben werden, wie aus den ADST-Dateien das Ergebnis für die $X_{\max}$ -Analyse gewonnen wird. Bei der $X_{\max}$ -Analyse ist es erforderlich, dass alle Ereignisse vollständig rekonstruiert werden können. Insbesondere das $X_{\max}$ muss gut bestimmbar sein. Der gesamte Datensatz von über 3 Millionen Ereignissen wird dadurch auf 34792 Ereignisse verkleinert. Auf diesen verkleinerten Datensatz wird nun ein Wolkenschnitt angewendet. Die Logik dieses Wolkenschnitts wird im nächsten Abschnitt genauer beschrieben. Die Ereignisse, die die Bedingungen des Wolkenschnitts erfüllen, bilden den *selected*-Datensatz, diejenigen, die heraus sortiert werden den *removed*-Datensatz. Beide zusammen bilden den *all*-Datensatz. Da für die $X_{\max}$ -Analyse nur ein Bruchteil der rekonstruierten Informationen benötigt wird, werden die erforderlichen Daten aus diesem Datensatz extrahiert. Wichtig für die weitergehende Analyse sind die Ereignis-ID, die Energie und das $X_{\max}$ des Schauers. Außerdem werden die aktiven Standorte, die Teleskope, die den Schauer registriert haben und die getriggerten Pixel benötigt, um im nächsten Schritt für die richtigen Teleskope und Pixel die Wolkenbedeckung aus der Wolkendatenbank zu extrahieren. Die genannten Variablen werden aus der ADST-Datei in eine neue „output.root“ Datei geschrieben. Sie sind jeweils ein Branch in dem Root-Tree und jedes Ereignis füllt einen Eintrag dort hinein. Im zweiten Schritt wird der Root-Tree durch die Wolkeninformationen für alle Pixel jeden aktiven Teleskops pro Ereignis ergänzt. Alle erforderlichen Informationen für die $X_{\max}$ -Analyse befinden sich nun kompakt und übersichtlich in einer .root Datei.

Diese wird für die $X_{\max}$ -Analyse zunächst eingelesen. Es kann dann ein Wolkenschnitt auf den Datensatz angewandt werden. Anschließend werden dieselben $X_{\max}$ Korrekturen wie in [11] angewandt und jedes Ereignis wird mit einem Index versehen, der das Energieintervall beschreibt, in dem sich den Schauer befindet. Zwei am Ende erstellten Textdateien fassen die Ergebnisse zusammen. Im letzten Schritt werden verschiedene Datensätze miteinander verglichen. Für jedes Energieintervall erhält man $X_{\max}$ Histogramme, aus denen das $\langle X_{\max} \rangle$ und das $\sigma(X_{\max})$ abgeleitet werden kann. $\langle X_{\max} \rangle$ und $\sigma(X_{\max})$ wird nun gegen die Energie aufgetragen. Auf diese Weise ist es möglich die Plots aus dem Paper [11] exakt zu reproduzieren und zugleich eigene Wolkenschnitte auf denselben Datensatz anzuwenden.

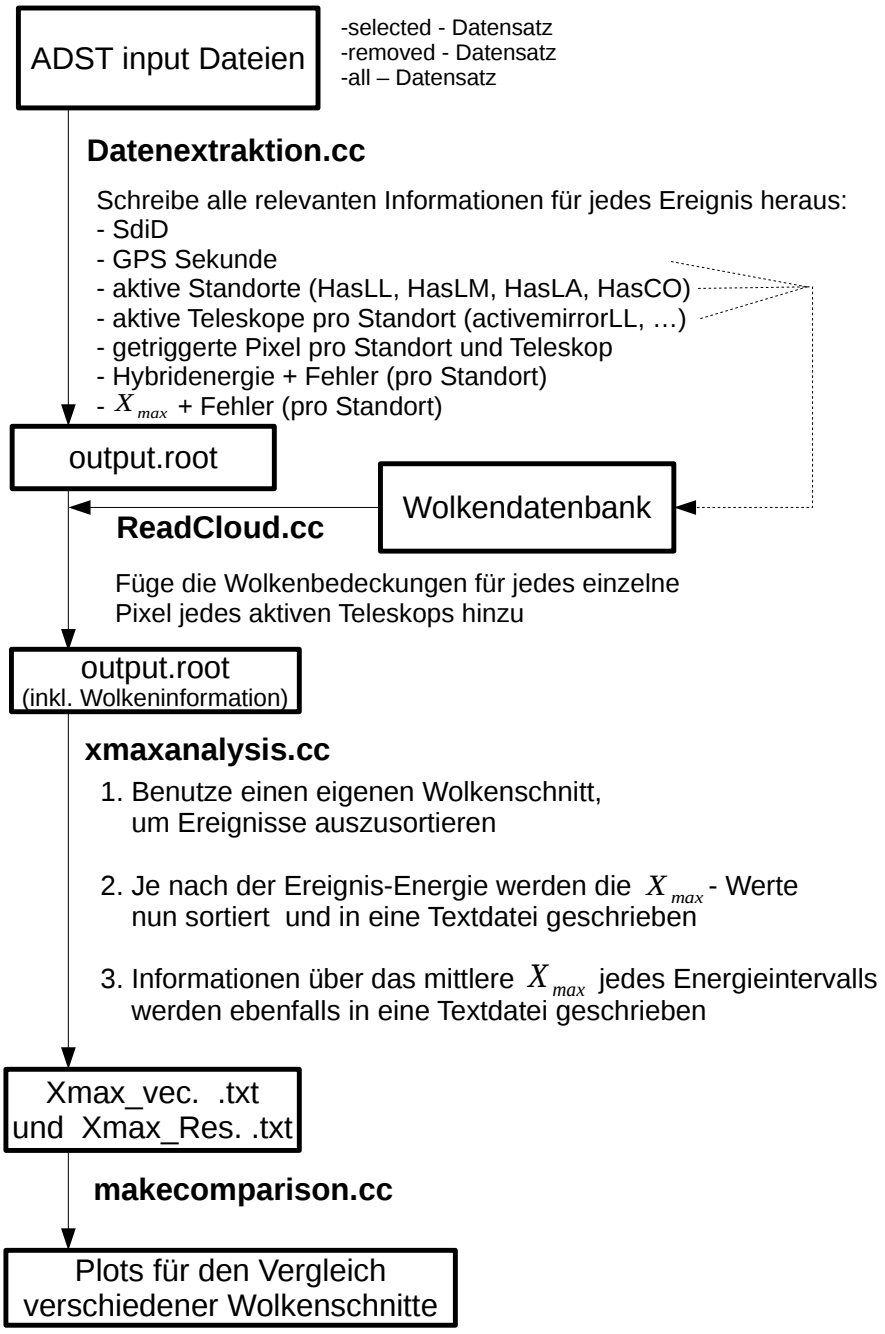



Abbildung 4.1: Dieses Flussdiagramm zeigt die notwendigen Schritte, die vom Extrahieren der richtigen Informationen aus den Datenbanken von Auger, bis zum Vergleich zweier Datensätze durch die $X_{\max}$ -Analyse getan werden müssen. Der Programmcode für die einzelnen Schritte befindet sich im Anhang (siehe Anhang B).

## 4.2. Beschreibung des bisherigen Wolkenschnitts

Erfüllen Ereignisse nicht die Bedingungen des bisherigen Wolkenschnitts, so werden diese Ereignisse in den *removed*-Datensatz aussortiert. Dieser Wolkenschnitt besitzt zusammengefasst basierend auf [12] folgende Logik.

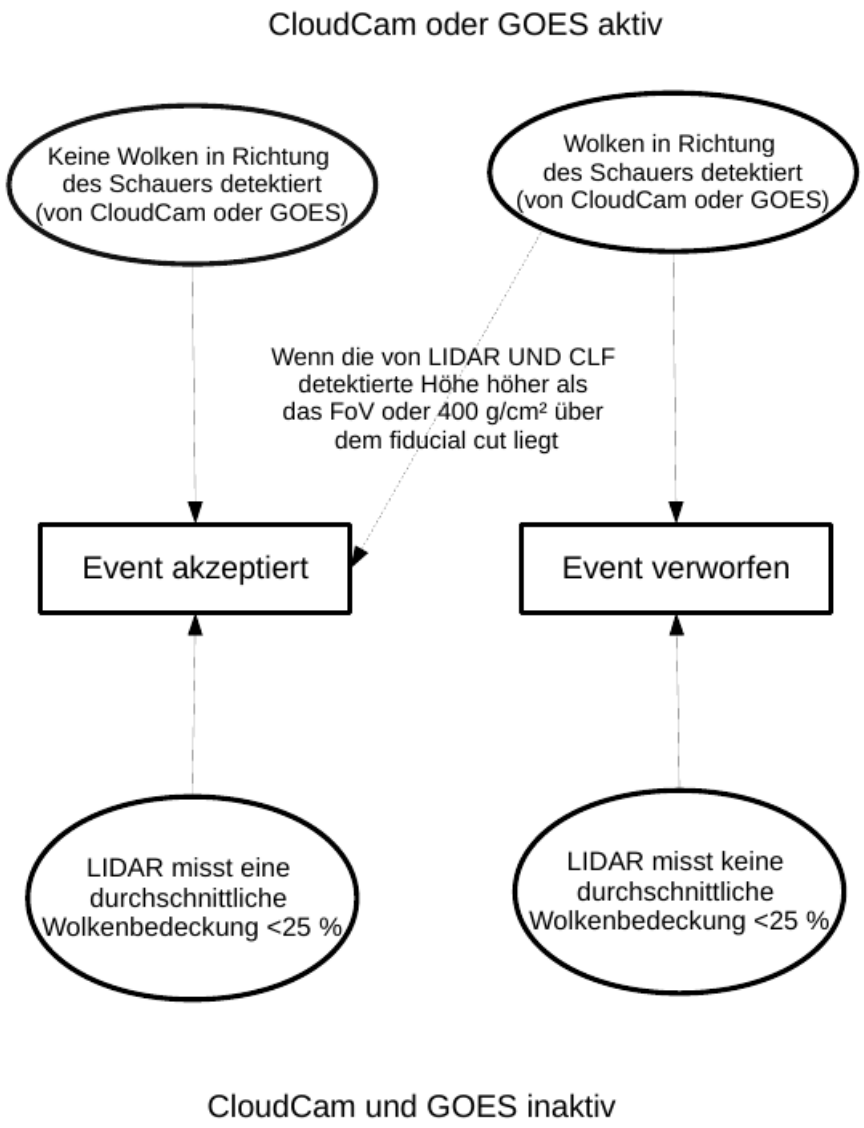



Abbildung 4.2: Logik des bisherigen Wolkenschnitts in der $X_{\max}$ -Analyse. Die Erläuterung des Flussdiagramms befindet sich im Text.

Für ein gegebenes Ereignis wird zunächst geprüft, ob die Daten von Wolkenkameras oder die Satellitendaten zur Verfügung stehen. In diesem Fall wird das Flussdiagramm von oben nach unten gelesen. Es wird dann überprüft, ob eine Wolke im Sichtfeld der Kamera oder der Satelliten gefunden wurde oder nicht. Wurde keine Wolke gefunden,

bleibt das Ereignis in dem Datensatz. Wurde eine Wolke gesichtet, so ist es immer noch möglich, dass das Ereignis nicht verworfen wird, indem die in dem Flussdiagramm angegebene Bedingung erfüllt ist. Waren sowohl die Wolkenkameras, als auch die Satelliten inaktiv, so ist das Flussdiagramm von unten nach oben zu lesen und es werden auf Daten von Lidar zurückgegriffen.

## 4.3. Möglichkeiten zur Veränderungen des Wolkenschnitts

Im Rahmen dieser Bachelorarbeit wurde die Möglichkeit entwickelt eigene Wolkenschnitte auf den *all*-Datensatz anzuwenden. Diese Wolkenschnitte basieren ausschließlich auf den Daten der Wolkenkameras. In Abb. 4.3 ist ein Beispiel einer von einem Luftschauer hinterlassenen Spur im Detektor dargestellt. Die Spur liegt im Sichtfeld eines einzelnen Teleskops. An dem Azimutalwinkel, der den Bereich von 30° - 60° abdeckt, ist ersichtlich, dass es sich um das zweite von sechs Teleskopen eines Standorts handelt.

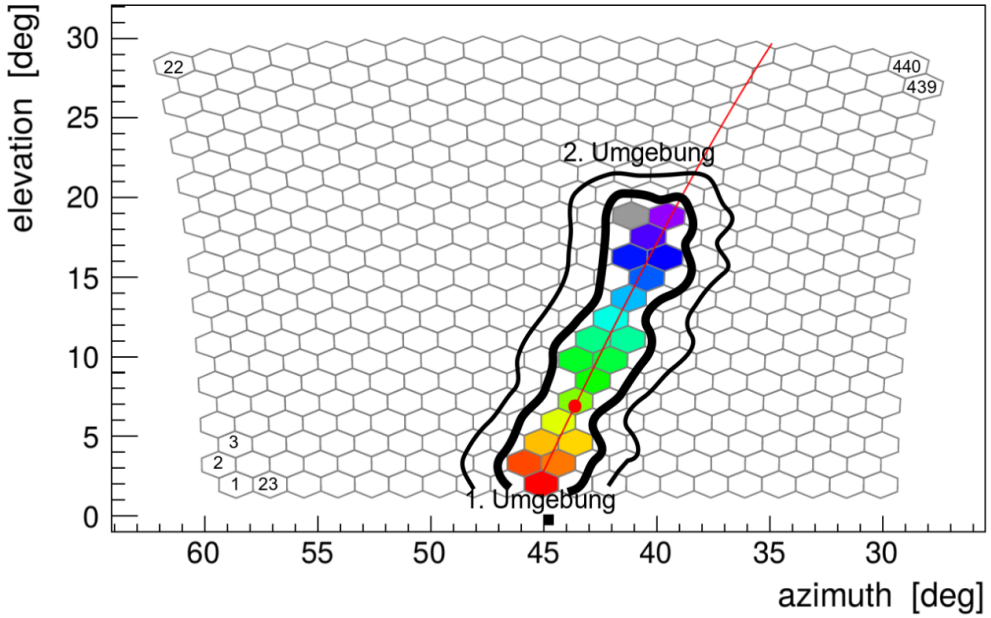



Abbildung 4.3: Das Sichtfeld des Teleskops ist aus 440 Pixeln zusammengesetzt. Die Nummerierung dieser Pixel beginnt unten links. Es werden die Spalten von links nach rechts durchgezählt, sodass sich der 440. Pixel oben rechts befindet. Die farbigen Pixel sind getriggerte Pixel und sie bilden die Spur von einem willkürlich gewähltem Ereignis. Die 1. Umgebung umfasst die getriggerten Pixel und zusätzlich die benachbarten Pixel. Die zweite Umgebung umfasst die Pixel der 1. Umgebung und zusätzlich deren benachbarten Pixel. Wolkenschnitte können gemacht werden, indem über die Wolkendeckung der getriggerten Pixel (Pixelauswahl 0), der 1. Umgebung (Pixelauswahl 1), der 2. Umgebung (Pixelauswahl 2) oder über die Wolkenbedeckung aller Pixel des Teleskops (Pixelauswahl 3) gemittelt wird.

In der Abb. 4.3 ist das Sichtfeld eines Teleskops zusammengesetzt aus 440 Pixeln zu sehen. In das Bild der Wolkenkameras wird dann eine Maske von diesen 440 Pixeln gelegt, sodass nun für jeden Pixel Wolkeninformationen zur Verfügung stehen. Über die verschiedenen in Abb. 4.3 Pixelauswahlen kann nun eine durchschnittliche Wolkenbedeckung gebildet werden. Den verschiedenen Pixelauswahlen ist ein Index zwischen null und drei zugeordnet, um im Programmcode diese Einstellungen leichter ändern zu können. Die Zuordnung ist dabei wie in Figur 4.3 beschrieben. Um anschließend zu entscheiden, welches Ereignis aussortiert wird, wurde ein weiterer Parameter in dem Programm integriert. Der Parameter „Wolkenlimit“ gibt Aufschluss darüber, ab welcher durchschnittlichen Wolkenbedeckung ein Ereignis aussortiert wird. An wenigen Stellen wurde das Wolkenlimit auch benutzt, um gerade die Ereignisse zu behalten, deren durchschnittliche Wolkenbedeckung über dem Wolkenlimit liegen. An einem Beispiel sollen diese Informationen etwas verdeutlicht werden. In Abb. 4.13 taucht folgende Bezeichnung für einen Datensatz auf: „all_CL0_CO3“
Dies bedeutet, dass der „all“ Datensatz, d. h. alle 34792 Ereignisse, als Grundlage verwendet und diejenigen aussortiert wurden, deren durchschnittliche Wolkenbedeckung über alle 440 Pixel gemittelt (Pixelauswahl 3) größer als Null ist. „CL“ ist die Abkürzung für Wolkenlimit und „CO“ ist die Abkürzung für Pixelauswahl. Die Zahl hinter dem „CL“ ist der maximale Anteil an Wolken, dass das Ereignis akzeptiert wird. Oder andersherum formuliert: In diesem Datensatz befinden sich nur Ereignisse bei denen keine einzige Wolke vom Detektor in Richtung des Schauers gesichtet wurde. Obwohl die durchschnittliche Wolkenbedeckung in Prozent angegeben maximal 100% sein kann, taucht auch an ein paar Stellen ein „CL2“ auf. Eine Wolkenbedeckung von zwei ist im Programm als unphysikalischer Wert angegeben, wenn über ein Ereignis keine Informationen über die Wolkenbedeckung vorliegen. „selected_CL2_CO3“ bedeutet demnach, dass der komplette *selected*-Datensatz behalten wird, unabhängig davon, ob es Informationen über die Wolken gibt oder nicht, und auch unabhängig von der „Pixelauswahl“.

Um einen Überblick über die Datensätze und die Größe der Statistiken zu bekommen, sind in der folgenden Abb. 4.4 die Anzahlen der in dem jeweiligen Datensatz nach dem angegebenem Wolkenschnitt verbleibenden Ereignisse aufgeführt.

| Datensatz | Anzahl Ereignisse | über alle Pixel gemittelt (Pixelauswahl 3) | | | | |
|---|---|---|---|---|---|---|
| | | Wolkenbedeckung 0 | Wolkenbedeckung 0,2 | Wolkenbedeckung 1 | Wolkenbedeckung 2 | ohne Wolkeninfos |
| selected | 25884 | 7921 | 14129 | 15423 | 25884 | 10461 |
| removed | 8908 | 20 | 1010 | 3874 | 8908 | 5034 |
| all | 34792 | 7941 | 15139 | 19297 | 34792 | 15495 |

| Datensatz | Wolkenbedeckung 0% | | | | Wolkenbedeckung > 50% |
|---|---|---|---|---|---|
| | Pixelauswahl 3 | Pixelauswahl 2 | Pixelauswahl 1 | Pixelauswahl 0 | Pixelauswahl 3 |
| all | 7941 | 10539 | 11013 | 11661 | 2376 |

Abbildung 4.4: In der oberen Tabelle werden die drei verschiedenen Datensätze miteinander verglichen. In der zweiten Spalte ist die Gesamtzahl der Ereignisse in dem jeweiligen Datensatz angegeben. Die folgenden vier Spalten geben die Anzahl der übrigbleibenden Ereignisse an, wenn die maximale Wolkenbedeckung wie angegeben sein darf. In der letzten Spalte steht die Anzahl der Ereignisse ohne Wolkeninformationen von den Kameras. In der unteren Tabelle werden verschiedene Pixelauswahlen miteinander verglichen. In der letzten Spalte ist dort die Anzahl der verbleibenden Ereignisse aus dem *all*-Datensatz angegeben, wenn bei Pixelauswahl 3 die Ereignisse in dem Datensatz bleiben, deren durchschnittliche Wolkenbedeckung größer als das Wolkenlimit von 50% sind.

Aus der oberen Tabelle in Abb. 4.4 ist ersichtlich, dass die beiden Datensätze *selected* und *removed* keine Schnittmenge haben und zusammen den *all*-Datensatz ergeben. Wie bereits erklärt, besteht der Unterschied zwischen der Spalte mit Wolkenlimit 1 und der Spalte mit Wolkenlimit 2 in denjenigen Ereignissen, die keine Wolkeninformationen besitzen (letzte Spalte). Auffällig sind in dieser Tabelle besonders die 20 Ereignisse, die wegen Wolken aussortiert wurden, obwohl aus den Informationen der Wolkenkamera eine durchschnittliche Wolkenbedeckung im Sichtfeld von genau 0% hervorgeht. Dies ist ein Anzeichen dafür, dass die bei Figur 4.2 beschriebene Logik doch komplizierter im Code verankert ist, als es in [12] beschrieben wurde. Der Datensatz für die 7941 Ereignisse ist ein wichtiger Datensatz, weil sicher behauptet werden kann, dass sich absolut keine Wolke im Sichtfeld des Teleskops befunden hat. Eine weitere wichtige Größe ist der dazu komplementäre Datensatz in der unteren Tabelle in Abb. 4.4 unten rechts. Hier gibt die Zahl gerade diejenigen Ereignisse an, die Wolkeninformationen haben und deren Wolkenbedeckung oberhalb von dem Prozentsatz von 50% liegt. Diese 2376 Ereignissen haben sicher Wolken im Sichtfeld des Teleskops und es wird auch vorkommen, dass die Wolken sich auch an der Stelle des Schauers befinden. Da die Wolkenkamera nur eine 2D-Projektion des Sichtfelds liefert, lässt sich jedoch selbst dann keine Aussage darüber treffen, ob die Wolke aus Sicht des Detektors den Luftschauer verdeckt bzw. umhüllt, was die gemessenen Daten verfälschen würde, oder ob die Wolke sich aus Sicht des Detektors hinter dem Luftschauer befindet.

In Abb. 3.6 ist deutlich erkennbar, dass mit zunehmender Energie die Anzahl der

Einträge in den Histogrammen deutlich abnimmt. Diese Beobachtung wird bei jedem Datensatz gemacht und sie stellt insofern ein Problem dar, weil statistische Schwankungen bei kleineren Zahlen viel mehr ins Gewicht fallen, als bei großen Zahlen. Daher ist eine größere Anzahl an Ereignissen in einem Datensatz vorteilhaft, weil dann in der $X_{\max}$ -Analyse die Momente der $X_{\max}$ Verteilung besser und mit kleinerem Fehler bestimmt werden können.

## 4.4. Vergleich von $X_{\max}$ -Plots für verschiedene Wolkenschnitte

### 4.4.1. Vergleich mit dem Wolkenschnitt in [11]

In [11] wurde eine $X_{\max}$ -Analyse mit dem *selected*-Datensatz durchgeführt. Im Rahmen dieser Bachelorarbeit wird derselbe Datensatz verwendet, weswegen es möglich sein muss, die Ergebnisse aus [11] in Abb. 3.7 zu reproduzieren. Eine Ausnahme stellt der niedrige Energiebereich bis $10^{18,1}$ eV dar. In [11] wird darin auf Daten von den niederenergetischen Auger-Erweiterungen zurückgegriffen, die im Rahmen dieser Bachelorarbeit nicht berücksichtigt wurden. Für den Energiebereich darüber dient der Vergleich dann jedoch als Kontrolle, dass der verwendete Programmcode fehlerfrei ist. Der in der $X_{\max}$ -Analyse verwendete Wolkenschnitt dient als Vergleich zu den eigenen Datensätzen. Die Histogramme aus Abb. 3.6 wurden vollständig reproduziert und sind identisch im Anhang (siehe A.1) zu finden. Das Histogramm in Abb. 4.5 ist lediglich ein Beispiel, das sich wegen der steilen linken Flanke gut mit dem 17. Histogramm in Abb. 3.6 vergleichen lässt.

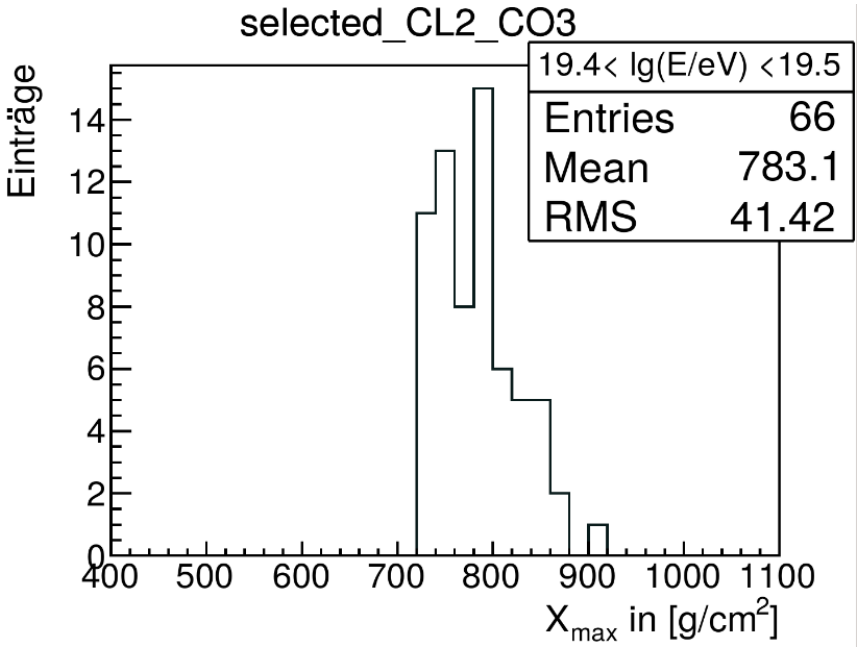



Abbildung 4.5: Auf der x-Achse ist das $X_{\max}$ aufgetragen. Jedes Ereignis aus dem *selected*-Datensatz in dem Energiebereich von $10^{19,4}$ eV - $10^{19,5}$ eV wird in das Histogramm eingetragen.

Da es in dieser Bachelorarbeit insbesondere um den von Wolken verursachten Unterschied geht, sollen beliebige Datensätze miteinander verglichen werden können. Aus dem Grund wurden die Informationen, die für die zu erstellenden Plots jedes Datensatzes nötig sind, in zwei Textdateien geschrieben, sodass im nächsten Schritt mittels dem Programm „makecomparison.cc“ zwei verschiedene Datensätze an Ereignissen miteinander verglichen werden können (vgl. Abb. 4.1).

### 4.4.2. Vergleich des Wolkenschnitts in [11] mit dem eigenen Wolkenschnitt

Nach der erfolgreichen Reproduktion der Ergebnisse soll der verwendete *selected*-Datensatz mit dem „all_CL0_CO3“-Datensatz verglichen werden. In diesem Datensatz sind nur Ereignisse enthalten, in denen alle 440 Pixel im Teleskop eine Wolkenbedeckung von Null haben. Auch wenn der letztere Datensatz weniger Statistik hat, so hat er den Vorteil, dass er allein auf Basis der Daten der Wolkenkameras funktioniert. In Abb. 4.6 und Abb. 4.7 ist das Ergebnis der $X_{\max}$ -Analyse zu sehen.
In der Abb. 4.6 ist das $\langle X_{\max} \rangle$ beider Datensätze als Funktion der Energie dargestellt. Die Tendenz eines mit der Energie zunehmenden $\langle X_{\max} \rangle$ ist in beiden Datensätzen erkennbar. In Abb. 4.7 ist die Differenz beider Datensätze als Funktion der Energie aufgetragen. Hier ist der Verlauf der Punkte zunächst konstant und dann fallend. In beiden Momenten der $X_{\max}$ -Verteilung liegen die Unterschiede beider Datensätze im statistischen Rahmen. Es kann daraus geschlussfolgert werden, dass die von einer Wolkenkamera aufzeichneten Daten ausreichen, um verlässliche Ergebnisse zu produzieren. Möglichkeiten, mit denen die Statistik des „all_CL0_CO3“-Datensatzes erhöht werden kann, werden in dem nächsten Unterkapitel näher erläutert. Zunächst soll getestet werden, ob die in Abb. 4.3 gezeigten Pixelauswahlen verschiedene Ergebnisse liefern.

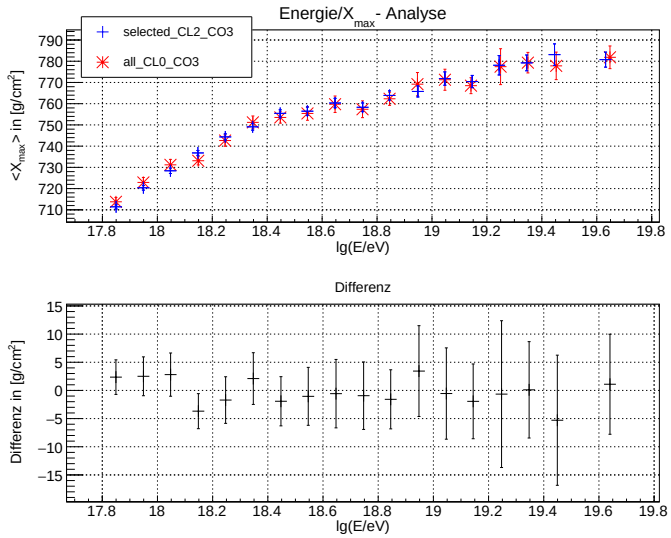



Abbildung 4.6: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\langle X_{\max} \rangle$ aufgetragen, auf der des unteren Graphen die Differenz (rot - blau) der verglichenen Datensätze in $\langle X_{\max} \rangle$. Blau gehört zu dem Datensatz in [11] und Rot zu dem „all_CL0_CO3“-Datensatz.

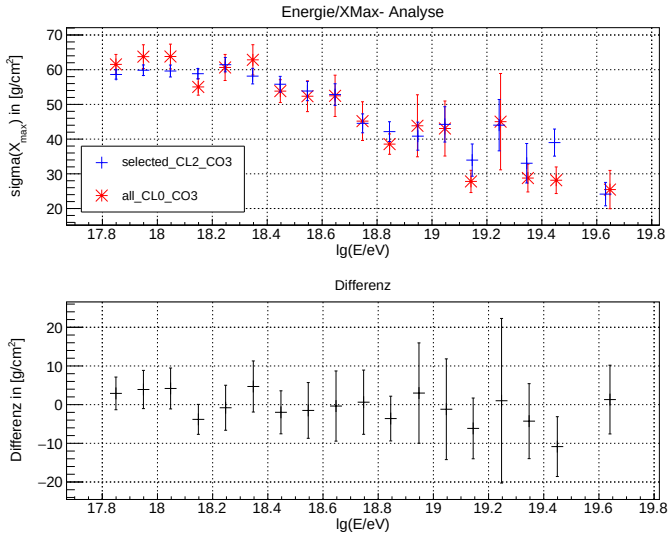



Abbildung 4.7: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\sigma(X_{\max})$ aufgetragen, auf der des unteren Graphen die Differenz (rot - blau) der verglichenen Datensätze in $\sigma(X_{\max})$. Blau gehört zu dem Datensatz in [11] und Rot zu dem „all_CL0_CO3“-Datensatz.

### 4.4.3. Vergleich verschiedener Pixelauswahlen

Um Abb. 4.3 wurde bereits erläutert, welche Pixel die verschiedenen Pixelauswahlen enthalten. Der Wolkenschnitt, der am sichersten von Wolken unbeeinflusst ist, ist der „all_CL0_CO3“-Datensatz, weil verlangt wird, dass das ganze Teleskop wolkenfrei ist. Darum wurde dieser „all_CL0_CO3“-Datensatz im letzten Abschnitt auch für den Vergleich mit dem Wolkenschnitt aus [11] verwendet. Dieser Datensatz enthält 7941 Ereignisse (siehe Abb. 4.4). Wird eine andere Pixelauswahl ausgewählt, so erhöht sich diese Zahl. Es reicht dann, wenn der Teil des Himmels um den Luftschauer herum wolkenfrei ist. Diese Tendenz ist auch deutlich beim Vergleich der verschiedenen Pixelauswahlen in der Abb. 4.4 zu sehen. Mit einer Vergrößerung des Datensatzes auf diese Art und Weise, wird ein höheres Risiko in Kauf genommen, dass die Bilder der Wolkenkamera nicht mehr die Situation zum Zeitpunkt des Schauers zeigen. In Abb. 4.8 und Abb. 4.9 werden die beiden Momente der $X_{\max}$ -Verteilung der vier wolkenfreien Datensätzen mit jeweils einer anderen Pixelauswahl miteinander verglichen.
Der Unterschied zwischen den Datensätzen ist nicht sichtbar. Bei beiden Momenten der $X_{\max}$ -Verteilung ist die Abweichung der einzelnen Datensätze zu dem „all_CL0_CO3“-Datensatz statistisch nicht signifikant. Beim ersten Moment (Abb. 4.8) ist in den meisten Energiebins das $\langle X_{\max} \rangle$ für den „all_CL0_CO3“ Datensatz etwas höher. Beim 2. Moment (Abb. 4.9) schwankt die Differenz der Datensätze symmetrischer um Null. Die absolute Größe der Schwankungen ist mit etwa 0-5 $\mathrm{g/cm^2}$ allerdings ähnlich groß, wie beim 1. Moment.
Zusammenfassend kann gesagt werden, dass der Unterschied zwischen den verschiedenen Pixelauswahlen nicht signifikant ist.

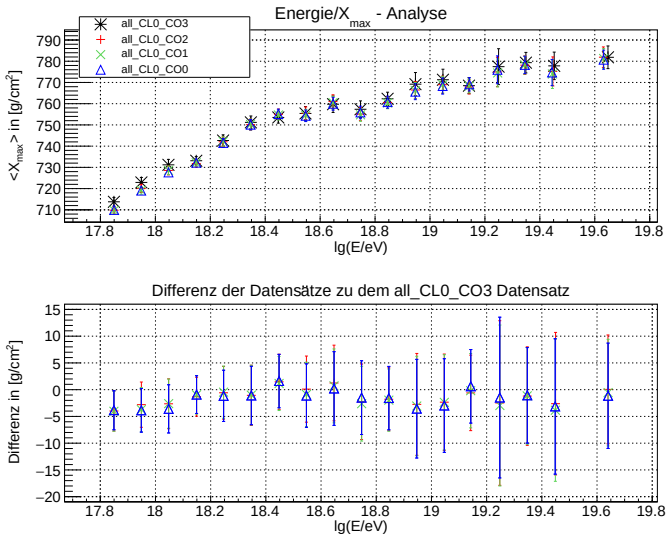



Abbildung 4.8: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\langle X_{\max} \rangle$ aufgetragen, auf der des unteren Graphen die Differenz der verglichenen Datensätze in $\langle X_{\max} \rangle$. Als Referenzdatensatz, der mit den verschiedenen Pixelauswahlen verglichen wird, dient der „all_CL0_CO3“-Datensatz.

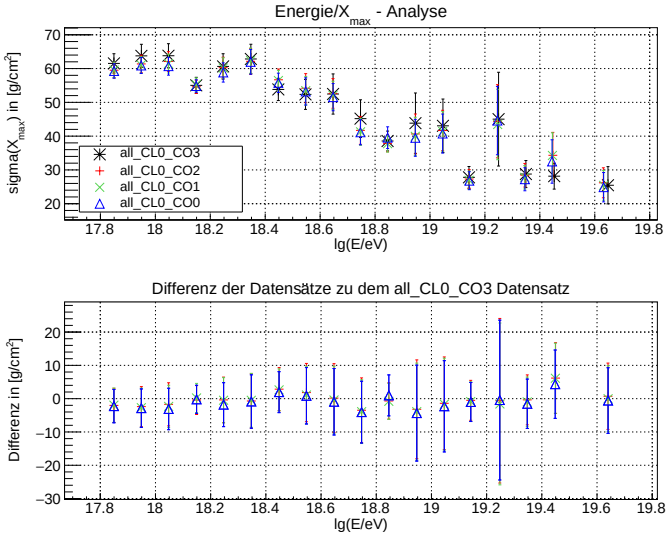



Abbildung 4.9: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\sigma(X_{\max})$ aufgetragen, auf der des unteren Graphen die Differenz der verglichenen Datensätze in $\sigma(X_{\max})$. Als Referenzdatensatz, der mit den verschiedenen Pixelauswahlen verglichen wird, dient der „all_CL0_CO3“-Datensatz.

### 4.4.4. Vergleich des *selected*- mit dem *removed*-Datensatz

Jetzt soll der *selected*-Datensatz aus [11] mit dem ihm disjunkten *removed*-Datensatz verglichen werden. Die blauen Punkte entsprechen demnach den Punkten aus Abb. 3.7. Das Ergebnis für $\langle X_{\max} \rangle$ sieht folgendermaßen aus:

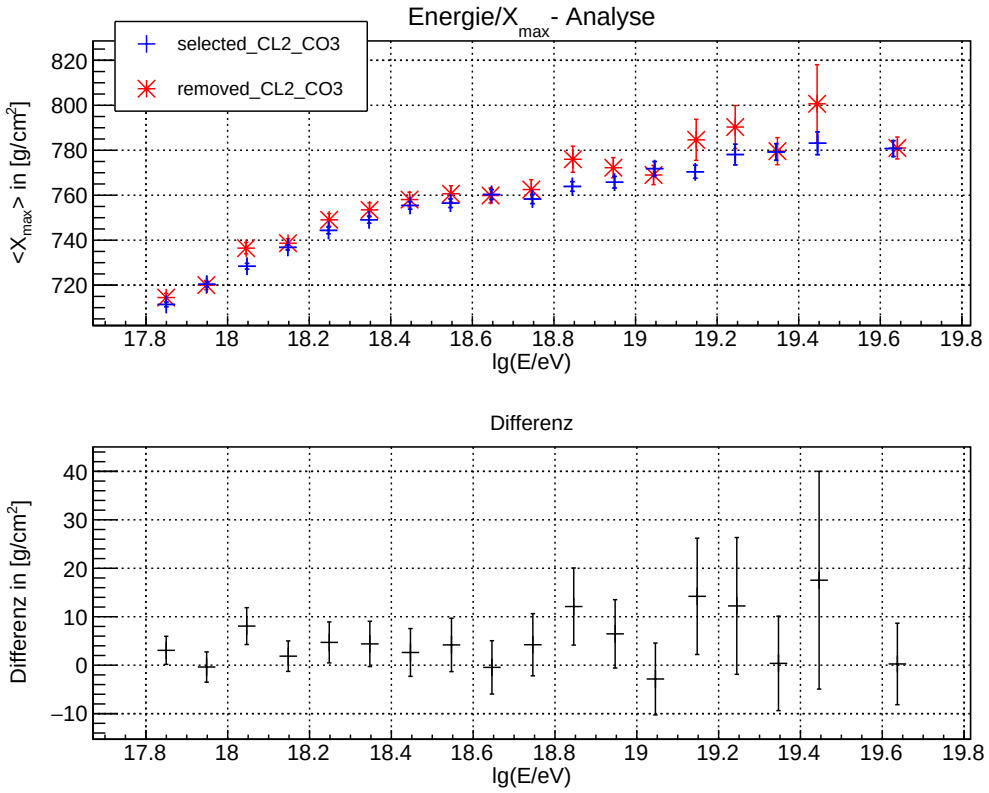



Abbildung 4.10: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\langle X_{\max} \rangle$ aufgetragen, auf der des unteren Graphen die Differenz (rot - blau) der verglichenen Datensätze in $\langle X_{\max} \rangle$. Blau gehört zu dem *selected*-Datensatz und Rot zu dem *removed*-Datensatz .

Bei dem *removed*-Datensatz ist das $\langle X_{\max} \rangle$ in den meisten Energieintervallen ein wenig höher als bei dem *selected*-Datensatz. Dies ist deutlicher an dem unteren Plot sichtbar, weil die Messpunkte nicht symmetrisch um Null verteilt sind, sondern eher über Null liegen. Dennoch sind die Abweichungen zwischen den beiden Datensätzen auch hier mit null verträglich. Ein Grund für diese geringen Unterschiede zwischen den Datensätzen könnte sein, dass schon im Vorfeld Schnitte auf den Gesamtdatensatz gemacht wurden, sodass daraus letztendlich der *all*-Datensatz entstanden ist, der im Wesentlichen schon von Wolken bereinigt ist. Zu diesen Schnitten gehört zum Beispiel, dass die Spur des Luftschauers im Detektor nicht unterbrochen sein darf, weil das ein direkter Hinweis auf eine Wolke ist.

In dem Histogramm für das Bin $10^{18,0}$ eV - $10^{18,1}$ eV (siehe Abb. 4.11) ist zu erkennen, dass tendenziell auf der rechten fallenden Flanke der Prozentsatz an Ereignissen aus dem *removed*-Datensatz etwas höher liegt, und genau umgekehrt ist es auf der linken steigenden Flanke.

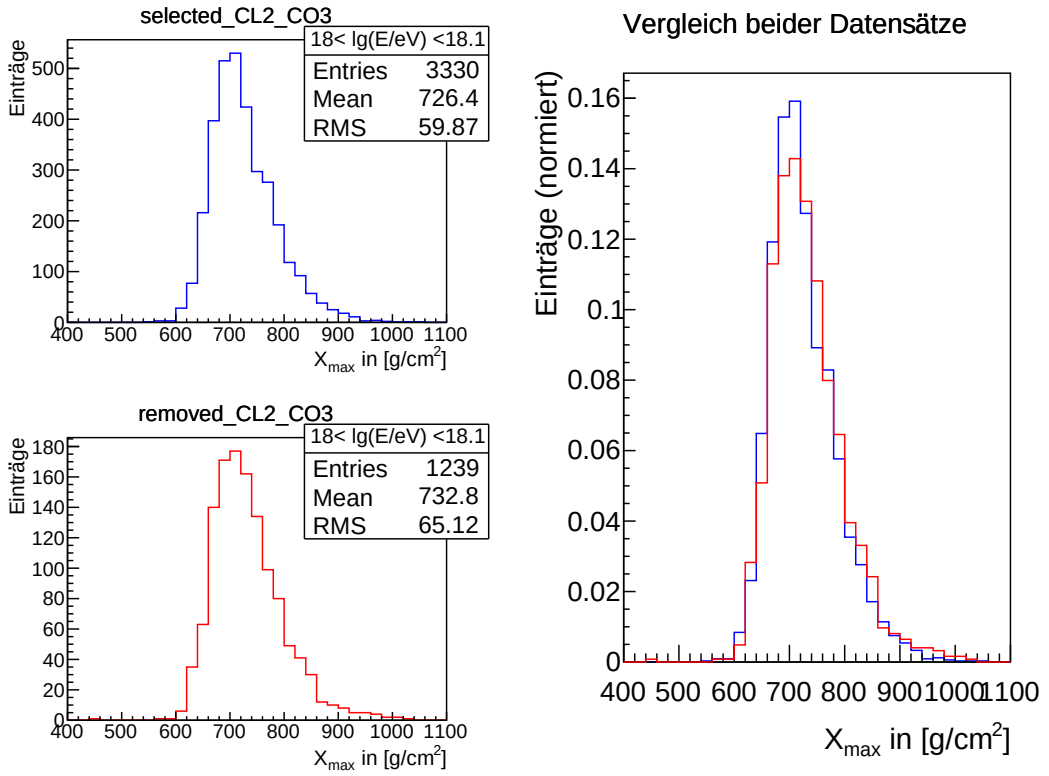



Abbildung 4.11: Bei allen drei Histogrammen ist das $X_{\max}$ jedes zu dem jeweiligen Datensatz gehörenden Ereignisses in dem Energieintervall von $10^{18}$ eV - $10^{18,1}$ eV die histogrammierte Größe. Auf der linken Seite befinden sich einzeln die beiden Histogramme für den *selected*-Datensatz und den *removed*-Datensatz . Auf der rechten Seite ist wegen der Normierung der direkte Vergleich beider Histogramme möglich.

Die Histogramme für die anderen Energieintervalle befinden sich im Anhang (siehe Abb. A.2, A.3, A.4).
In einem weiteren Graphen ist das 2. Moment der $X_{\max}$ -Verteilung gegen die Energie aufgetragen.

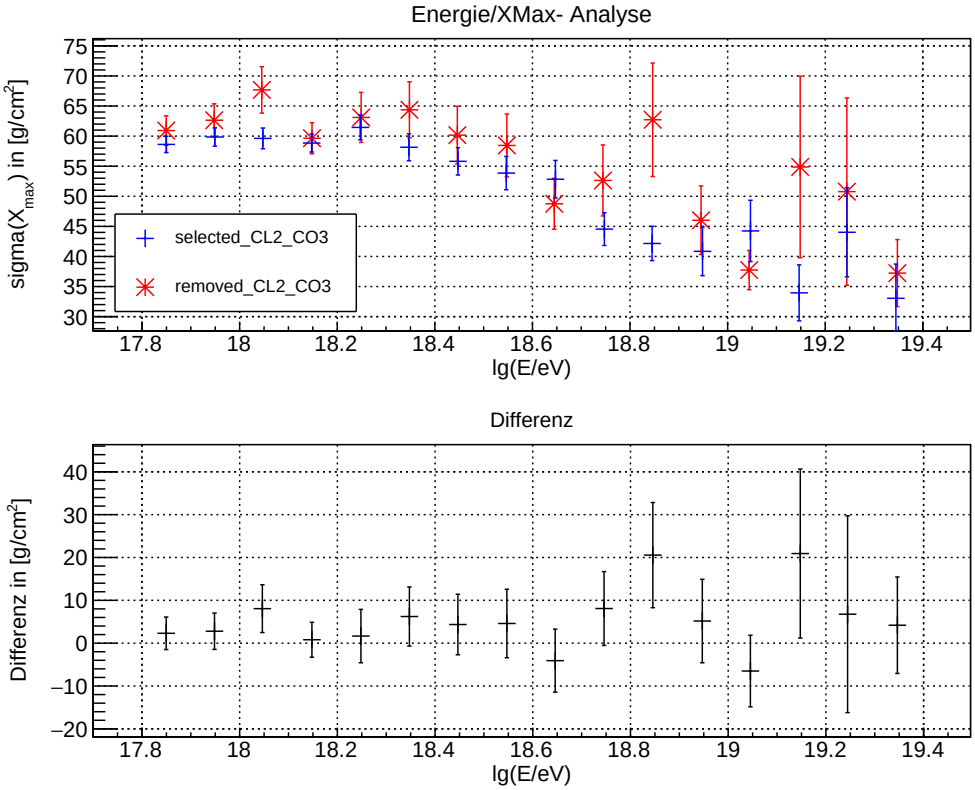



Abbildung 4.12: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\sigma(X_{\max})$ aufgetragen, auf der des unteren Graphen die Differenz (rot - blau) der verglichenen Datensätze in $\sigma(X_{\max})$. Blau gehört zu dem *selected*-Datensatz und Rot zu dem *removed*-Datensatz .

Auch hier gleicht sich bei niedrigeren Energien tendenziell das Verhalten beider Datensätze, indem die Breite der Verteilung zunächst konstant bleibt und nachher kleiner wird. Bei höheren Energien wird jedoch die Statistik zu klein und die Schwankungen werden größer. In dem Histogramm für das Bin $10^{18,8}$ eV - $10^{18,9}$ eV (siehe Anhang A.3) ist sichtbar, dass einzelne Ereignisse mit großem $X_{\max}$ dafür sorgen, dass die Breite der Verteilung so groß im Vergleich zu der des *selected*-Datensatzes bei der gleichen Energie ist. Aus den Daten für die Energiebins mit mehr Statistik scheint aber dennoch hervorzugehen, dass das $\sigma(X_{\max})$ bei dem *removed*-Datensatz ein wenig größer sein könnte als bei dem *selected*-Datensatz. Jedoch ist auch hier der Unterschied statistisch nicht signifikant.

### 4.4.5. Vergleich eines wolkenfreien mit einem nicht wolkenfreien Datensatz

Nun sollen die eigenen Wolkenschnitte verwendet werden und der wolkenfreie Datensatz mit einem Datensatz verglichen werden, in dem nur Ereignisse sind, deren durchschnittliche Bedeckung über das Teleskop gemittelt größer als 50% ist (vgl. Abb. 4.4).

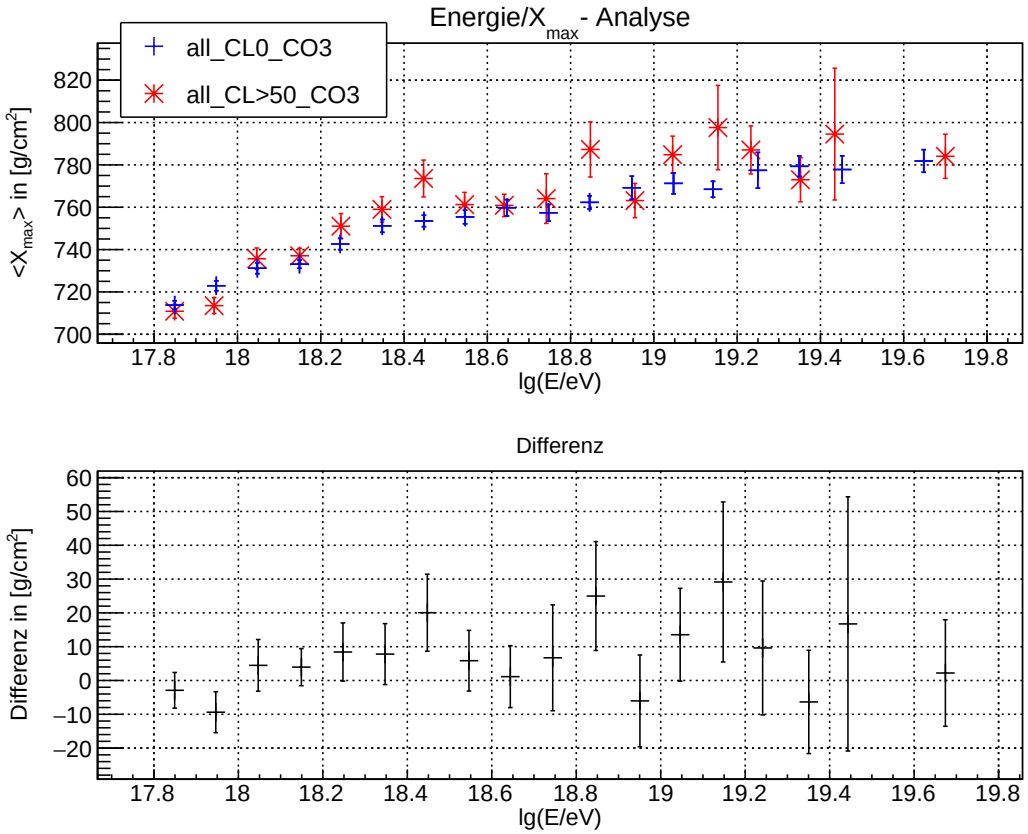



Abbildung 4.13: Auf den x-Achsen beider Graphen ist die Energie aufgetragen. Auf der y-Achse des oberen Graphs ist $\langle X_{\max} \rangle$ aufgetragen, auf der des unteren Graphen die Differenz (rot - blau) der verglichenen Datensätze in $\langle X_{\max} \rangle$. Blau gehört zu dem wolkenfreien und Rot zu dem wolkenbehafteten Datensatz

In den ersten beiden Energieintervallen ist das $\langle X_{\max} \rangle$ des wolkenbehafteten Datensatzes etwas höher. In den nachfolgenden Energieintervallen ist das $\langle X_{\max} \rangle$ des wolkenfreien Datensatzes bis auf zwei Ausnahmen niedriger. Die Differenz beider Datensätze ist allerdings bis auf drei Energiebins immer mit einem $\sigma$ mit Null verträglich. Insgesamt sieht diese Abb. 4.13 der Abb. 4.10 ähnlich. Dies ist auch zu erwarten, da derselbe Vergleich gemacht wurde, aber mit Datensätzen, die aus eigenen Wolkenschnitten erstellt worden sind. Aber andererseits ist es auch hier, wie in Abb. 4.10 nicht möglich, einen Einfluss der Wolken auf das Ergebnis der $X_{\max}$ -Analyse zu schlussfolgern. Bessere Analysen könnten gemacht werden, indem Ereignisse aus dem „bedeckten“ Datensatz herausgenommen werden, bei denen die Wolke aus Sicht des Detektors hinter der Schauerachse liegt. Wegen der einfachen 2D Projektion fallen diese Ereignisse momentan ebenfalls in diesen Datensatz hinein.

In den „all_CL>50_CO3“ Datensatz fließen auch Ereignisse mit ein, in denen die Wolkenbedeckung fälschlicherweise auf eins rekonstruiert wurde, obwohl keine Wolken

am Himmel sind. Dies soll nun diskutiert werden.

**Bilder der Wolkenkamera und falsch rekonstruierte Wolkenbedeckungen:** Auf der Internetseite [13] der Universität Adelaide kann man sich die Wolkenbedeckung bei jeder beliebigen GPS Sekunde an allen 4 Standorten bildlich anzeigen zu lassen (siehe Abb. 4.14).


Abbildung 4.14: Zu sehen ist der Himmel, aufgenommen von den Infrarotkameras an den vier Standorten der FD zu den GPS-Sekunden 1125994562 (links) und 972527414 (rechts). Ein klarer Himmel ist schwarz und weiß bedeutet, dass Wolken im Sichtfeld des Standorts waren. Dargestellt sind jeweils alle 6 Teleskope pro Standort. Auf dem rechten Bild sind von Los Leones aus wenige Wolken zu sehen gewesen, die anderen drei Standorte besitzen keine Wolkenkamerainformationen zu dieser Zeit. Auf der linken Seite ist ein wolkenfreies Ereignis zu sehen, in dem der Himmel am Standort Los Leones falsch rekonstruiert wurde. (Bild von [13])

Wie in Abb. 3.1 zu erkennen ist, überschneiden sich die Sichtfelder von den einzelnen Standorten. Es ist demnach unmöglich, dass an einem Standort ausschließlich Wolken von der Kamera detektiert werden, während keine Wolken an den anderen Standorten registriert werden. Der umgekehrte und wesentlich gefährlichere Fall, dass ein Ereignis fälschlicherweise als wolkenfrei deklariert wird, tritt nicht auf [14].

Das Ergebnis wird der Abb. 4.13 wieder unähnlicher, wenn man den „all_CL0_CO3“ mit einem Datensatz vergleicht, bei dem die durchschnittliche Wolkenbedeckung bei einem Ereignis größer als 95% ist. Dies könnte der geringeren Statistik und zufälligen Fluktuationen darin geschuldet sein (1019 Ereignisse sind in diesem Datensatz). Es könnte jedoch auch ein Hinweis darauf sein, dass die falsch rekonstruierten Ereignisse einen Einfluss auf das Ergebnis haben.

## 4.5. Zusätzliche Fragestellungen

In dem folgenden Abschnitt werden neue Fragestellungen behandelt, mit deren Hilfe neue Dinge über das Verhalten der Wolken und deren Einfluss auf Daten gewonnen werden können. Das Ziel, die Statistik zu vergrößern, soll dabei nicht außer Acht gelassen werden.

### 4.5.1. Die durchschnittliche Wolkenbedeckung in der Pampa

Um die Fragestellung zu beleuchten, inwiefern die Ergebnisse der $X_{\max}$ -Analyse verfälscht werden, wenn Ereignisse, bei denen Wolkeninformationen fehlen, für die Analyse mitbenutzt werden, soll an dieser Stelle untersucht werden, wie viele Wolken sich durchschnittlich über dem Experiment befinden.
Die durchschnittliche Wolkenbedeckung wurde in der Abb. 4.15 nur über die getriggerten Pixel, d. h. Pixelauswahl 0, berechnet. Der Unterschied zwischen den einzelnen Pixelauswahlen ist unwesentlich klein. In Abb. 4.15 ist die Häufigkeit einzelner Wolkenbedeckungen logarithmisch in ein Histogramm eingetragen. Die Anzahl der Einträge beinhaltet diejenigen Ereignisse, die keine Wolkeninformationen besitzen, deren Wolkenbedeckung also „künstlich“ auf zwei gesetzt wurde. Im Histogramm selbst sind diese nicht sichtbar, in der Statistikbox gibt der Parameter „overflow“ die Anzahl dieser Ereignisse an. In den Mittelwert in der Statistikbox fließen die „overflow“ - Einträge nicht mit ein. Die Gesamtanzahl von 35122 Einträgen (im Gegensatz zu den 34792 Ereignissen) kommt dadurch zustande, dass Mehrfachereignisse mehrfach gezählt werden.

Der Mittelwert in Abb. 4.15 für den *selected*-Datensatz liegt bei 6% und bei dem *removed*-Datensatz bei 51,8%. Dieser deutliche Unterschied wird auch bei dem direkten Vergleich sichtbar, in dem ab einer Wolkenbedeckung von 4% die Kurve des *removed*-Datensatzes über der des *selected*-Datensatzes liegt. Dennoch wird auch hier, wie in Abb. 4.4 deutlich, dass sowohl in dem *selected*-Datensatz ca. 10-13% der Ereignisse eine Wolkenbedeckung über ca. 20% haben als auch, dass in dem *removed*-Datensatz ca. 20% der Ereignisse eine Wolkenbedeckung unter 10% haben.

Eine wichtige Schlussfolgerung, die aus der Abb. 4.15 (links unten) gezogen werden kann, ist die Abschätzung, dass in dem Fall, dass ein Ereignis keine Wolkeninformationen besitzt, es in ca. 70% der Fälle eine Wolkenbedeckung weniger als 10% besitzt. Eine komplette Wolkenfreiheit im Sichtfeld des jeweils getriggerten Teleskops gab es in ca. 50% der Ereignisse. Um eine größere Sicherheit zu erhalten, ein Ereignis ohne Information als wolkenfrei einzustufen, gibt es verschiedene Möglichkeiten. Eine Möglichkeit könnte sein die Jahreszeit mitzuberücksichtigen, wenn im Sommer die Wahrscheinlichkeit für gutes Wetter höher ist, als im Winter. Zwei weitere Möglichkeiten sollen in den nächsten Kapiteln noch näher diskutiert werden.

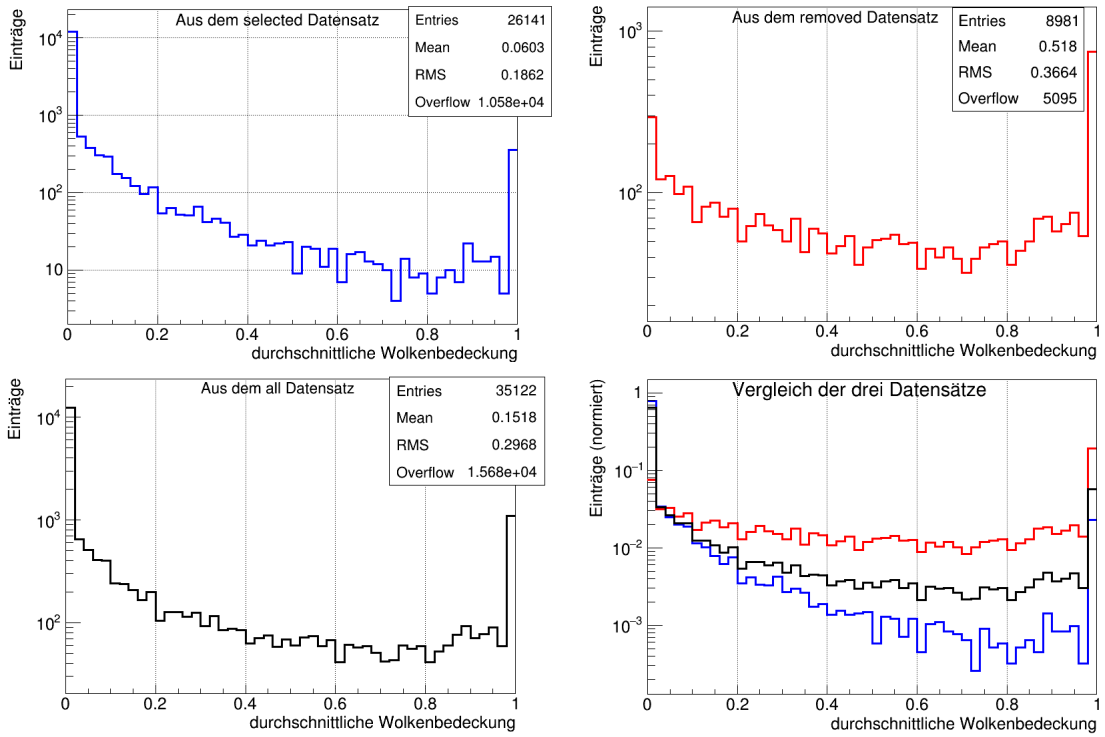



Abbildung 4.15: In allen 4 Histogrammen ist die durchschnittliche Wolkenbedeckung in der Spur des Schauers (Pixelauswahl 0) in ein Histogramm eingetragen. Der Titel des jeweiligen Histogramms gibt den Datensatz an, der ausgewertet wurde. Die Anzahl der Einträge, der Mittelwert und die Breite der Verteilung stehen in der jeweiligen Statistikbox. Die „overflow"-Einträge geben die Anzahl der Ereignisse an, für die die Wolkenkamera keine Informationen liefert.

### 4.5.2. Wie schnell ändert sich die durchschnittliche Wolkenbedeckung in der Spur?

Um herauszufinden, ob es sicher ist die Pixelauswahl 0 zu verwenden, soll die Frage betrachtet werden, wie schnell sich die Wolkenbedeckung am Himmel ändert. Das Problem ist, dass die Wolkenkameras „nur" alle fünf Minuten ein Foto von dem jeweiligen Sichtfeld schießen. Wird ein Schauer detektiert, so werden die Daten des aktuellsten Fotos für diesen Schauer verwendet. Diese können demnach jedoch schon bis zu fünf Minuten alt sein. In fünf Minuten kann eine Wolke, die bisher nicht vor dem Luftschauer zu sehen war, weiterziehen, sodass die getriggerten Pixel überlagert werden. Und andersherum natürlich ebenfalls. Je nachdem, wie schnell die Wolken ziehen, ist es vielleicht sicherer, immer die Pixelauswahl 2 oder sogar Pixelauswahl 3 zu verwenden. In der folgenden Analyse wird die durchschnittliche Wolkenbedeckung der Bilder fünf Minuten vor und nach dem aktuellen Bild verglichen, indem jeweils die durchschnittliche Wolkenbedeckung der getriggerten Pixel oder über das ganze Teleskop gebildet und die Differenzen der Wolkenbedeckung ausgerechnet werden. Diese Differenzen werden in ein Histogramm eingetragen bzw. als Funktion der Wolkenbedeckung des aktuellen Bilds aufgetragen.

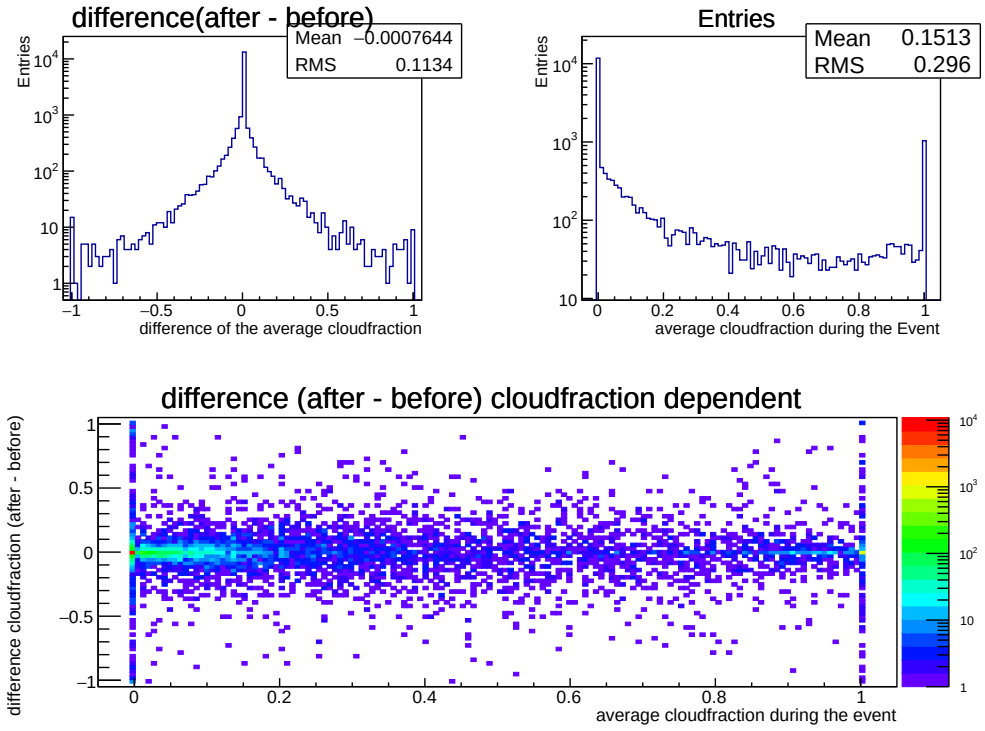



Abbildung 4.16: Das Histogramm auf der linken oberen Seite ist die y-Projektion des 2D-Histogramms unten. Das rechte obere Histogramm ist die x-Projektion des 2D-Histogramms unten. In dem 2D-Histogramm ist die Differenz der Wolkenbedeckung nach und vor dem Luftschauer (auf der y-Achse) abhängig von der Wolkenbedeckung während dem Luftschauer (auf der x-Achse) abgebildet.

Das Histogramm oben rechts in der Abb. 4.16 ist bis auf die Bingröße identisch mit dem Histogramm unten links in Abb. 4.15. Der Mittelwert im Histogramm oben links in der Abb. 4.16 ist erst in der 4. Nachkommastelle von Null verschieden. Da es sich bei diesem Histogramm um die y-Projektion des unteren 2D-Plots handelt, bedeutet dies auch, dass die Punkte im 2D-Histogramm nahezu symmetrisch um die Differenz von Null verteilt liegen. Dies ist zu erwarten, da die Existenz von Wolken nicht mit den Luftschauern korreliert ist. Aus dem oberen linken Histogramm ist ersichtlich, dass bei weniger als 3% der Ereignisse die Änderung der durchschnittlichen Wolkenbedeckung vom Betrag mehr als 50% ist. Das entspricht hier etwa 400 Ereignissen. Es sollte beachtet werden, dass einige der Ausreißer mit der Differenz von Eins mit Rekonstruktionsfehlern (siehe Abb. 4.14) zu erklären sind.
Diese Analyse wurde auch durchgeführt, um den *selected*-Datensatz mit dem *removed*-Datensatz zu vergleichen (siehe Abb. 4.17). Der Unterschied zwischen diesen beiden Datensätzen wird hauptsächlich im 2D Histogramm sichtbar. An dieser Stelle ist die durchschnittliche Wolkenbedeckung ebenfalls nur über die getriggerten Pixel gebildet worden.

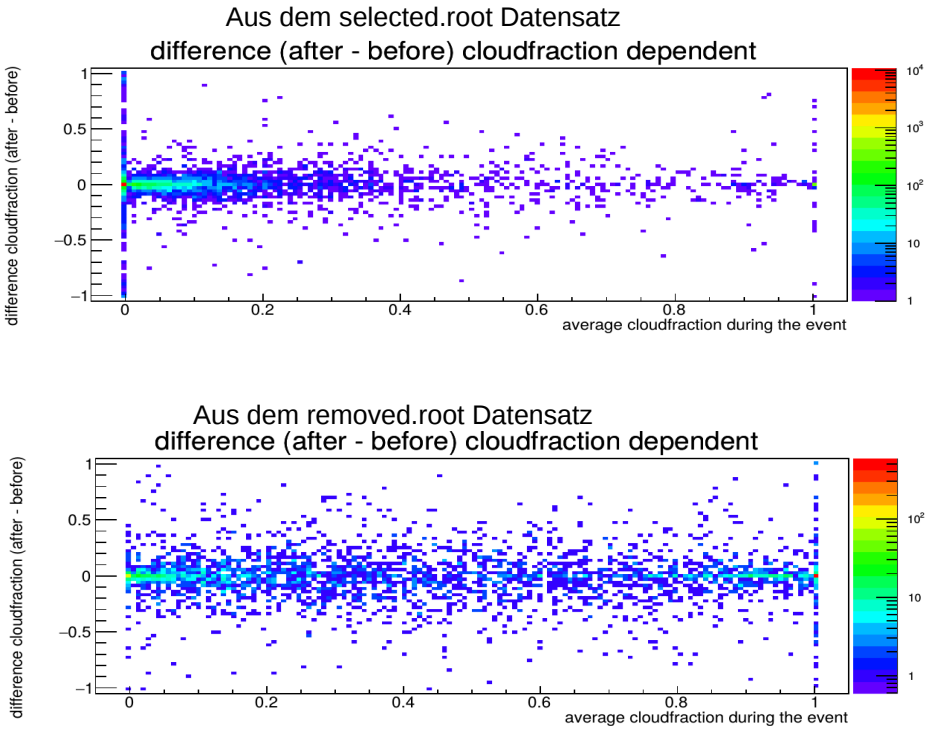



Abbildung 4.17: In beiden 2D-Histogrammen ist die Differenz der Wolkenbedeckung nach und vor dem Luftschauer (auf der y-Achse) abhängig von der Wolkenbedeckung während dem Luftschauer (auf der x-Achse) abgebildet. Für das obere Histogramm wurde der *selected*-Datensatz verwendet und für das untere Histogramm wurde der *removed*-Datensatz verwendet.

Es fällt auf, dass das obere Histogramm die meisten Einträge bei einer Differenz von 0% zwischen dem Bild vor und nach dem Schauer hat, während die Wolkenbedeckung auf dem aktuellen Bild ebenfalls Null ist. Das untere Histogramm in Abb. 4.17 hat die meisten Einträge in dem Bin, in dem die Differenz 0 ist, jedoch die Wolkenbedeckung im aktuellen Bild 1 ist. Daraus könnte geschlossen werden, dass komplett wolkenfrei bzw. komplett bedeckt ein stabiler Zustand ist. Stabil bedeutet dann in diesem Zusammenhang, dass sich die Wolkenbedeckung nur langsam verändert. Wie bereits erwähnt, können an dieser Stelle Ausreißer auch von Rekonstruktionsfehlern kommen.
Diese Schlussfolgerung betrifft auch das letzte Kapitel insofern, weil dann, wenn keine Wolkeninformationen vorliegen, eventuell korrekt rekonstruierte Daten einige Zeit früher oder später gefunden werden können und so die Wahrscheinlichkeit einen wolkenfreien Himmel richtig vorherzusagen genauer angegeben werden kann.
Ist der Himmel mittelmäßig bedeckt ist eine schnelle Änderung der Wolkenbedeckung wahrscheinlicher, im Vergleich zu einem wolkenfreien Himmel. Dies kann aus den Breiten der Verteilungen geschlussfolgert werden. Bei dem *selected*-Datensatz ist die Breite der

y-Projektion gegeben durch $\mathrm{RMS}_{\mathrm{sel}} = 0{,}0815$ im Vergleich zu $\mathrm{RMS}_{\mathrm{rem}} = 0{,}1944$ bei dem *removed*-Datensatz. Die breitere Streuung ist auch im 2D-Histogramm zu sehen (vgl. Abb. 4.17).

Eine weitere Beobachtung lässt sich machen, wenn die Breite der y-Projektion von dem Plot in Abb. 4.16 mit derjenigen verglichen wird, bei der die Wolkenbedeckung über das ganze Teleskop ermittelt wurde. Bei der Mittelung über alle Pixel (Pixelauswahl 3) wird die Breite im Vergleich zu den RMS = 0,1134 aus Abb. 4.16 (Pixelauswahl 0) um 17,8% schmaler. Die schmalere Verteilung ist gleichbedeutend damit, dass die Differenzen der durchschnittlichen Wolkenbedeckungen vor und nach dem Luftschauer näher um Null verteilt sind. Die durchschnittliche Wolkenbedeckung in der Spur ändert sich also schneller als die Wolkenbedeckung über das ganze Teleskop gemittelt. Dies wird auch erwartet, weil wenn über mehr Pixel gemittelt wird, vereinzelte „Ausreißer-Pixel“ bei der Mittelung weniger ins Gewicht fallen.

Eine Schlussfolgerung dieses Abschnitts ist, dass ein wolkenfreier und ein völlig bedeckter Himmel ein in dem Zeitraum von fünf Minuten relativ unveränderlicher Zustand ist. Ist der Himmel jedoch mittelmäßig bedeckt, so ist es gut möglich, dass sich die Situation geändert hat, ehe der Luftschauer kommt. Wird im Wolkenschnitt Pixelauswahl 3 verwendet, so ist das sicherer. Pixelauswahl 0 hat jedoch den Vorteil, dass mehr Ereignisse in den Datensatz aufgenommen werden (vgl. Abb. 4.4).

### 4.5.3. Der zeitliche Verlauf aktiver Wolkenkameras

Eine weitere Möglichkeit die Statistik zu erhöhen und gleichzeitig die Logik des Wolkenschnitts zu vereinfachen wird im Folgenden beschrieben. Das Problem, das hier umgangen werden soll, ist, dass nicht für jedes Ereignis Informationen von jedem der zur Überwachung der Atmosphäre eingesetzten Geräte zur Verfügung stehen. In der $X_{\mathrm{max}}$ -Analyse wird daher ein Wolkenschnitt mit einer komplizierten Logik, der die Daten von der Wolkenkamera, von den Satelliten und von Lidar, CLF und XLF berücksichtigt, benutzt, um einen wolkenfreien Datensatz zu erstellen. Die Wolkenkamera ist von diesen Systemen wahrscheinlich am besten geeignet, um zu behaupten, dass das Sichtfeld im Bereich des Schauers wolkenfrei ist. Aber auch die Wolkenkameras liefern nicht durchgängig Daten, sodass nicht jedes Ereignis Informationen über die Wolken von der Wolkenkamera bekommt. Aus der Präsentation [9] ist der monatliche Status der vier Wolkenkameras entnommen.

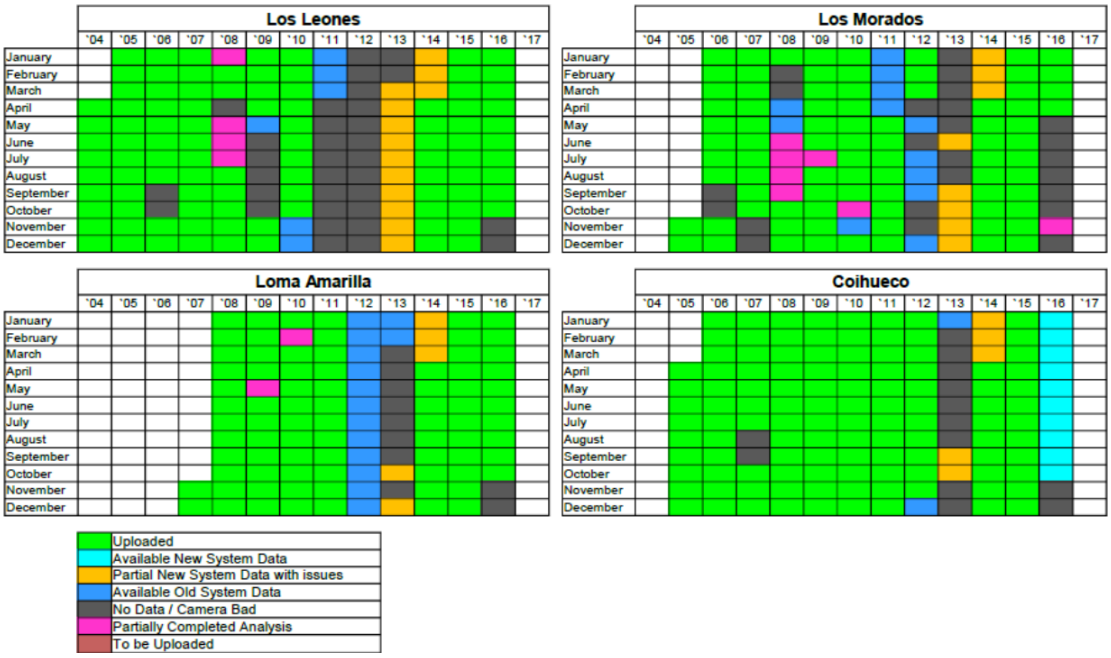



Abbildung 4.18: Dargestellt ist der monatliche Status der vier Wolkenkameras. Die grüne Farbe bedeutet, dass die Wolkendaten verfügbar sind. Blau und gelb bedeutet, dass teilweise Daten verfügbar sind, die aber noch in die Wolkendatenbank hochgeladen werden müssen. Dass keine Daten verfügbar sein werden, weil die Kamera nicht funktioniert hat, bedeutet die Farbe Grau und dass die Daten noch analysiert werden müssen, ist mit der Farbe Lila dargestellt. [9]

Aus Abb. 4.18 ist ersichtlich, dass keine der Wolkenkameras durchgängig aktiv ist. Besonders in den Jahren 2012 und 2013 fehlen wegen des Austauschs der Kameras viele Informationen für Ereignisse in diesem Zeitraum. Aber man erkennt beispielsweise auch, dass die Kamera in Coihueco im Gegensatz zu der in Los Leones funktioniert hat. Da sich, wie in Abb. 3.1 erkennbar, die Sichtfelder beider Kameras überlagern, kann die Frage gestellt werden, ob dies nicht ausgenutzt werden kann. Wenn dann ein Luftschauer von dem FD in Los Leones registriert wird, kann eventuell durch die von der Kamera in Coihueco gesammelten Wolkeninformationen die Bestätigung kommen, dass sich zwischen Luftschauer und Detektor keine Wolken befunden haben. Die folgende Abb. 4.19 ist eine Übersicht darüber, wie groß der prozentuale Anteil an Ereignissen ist, die durch diese Überlegung Wolkeninformationen von anderen Standorten bekommen können.

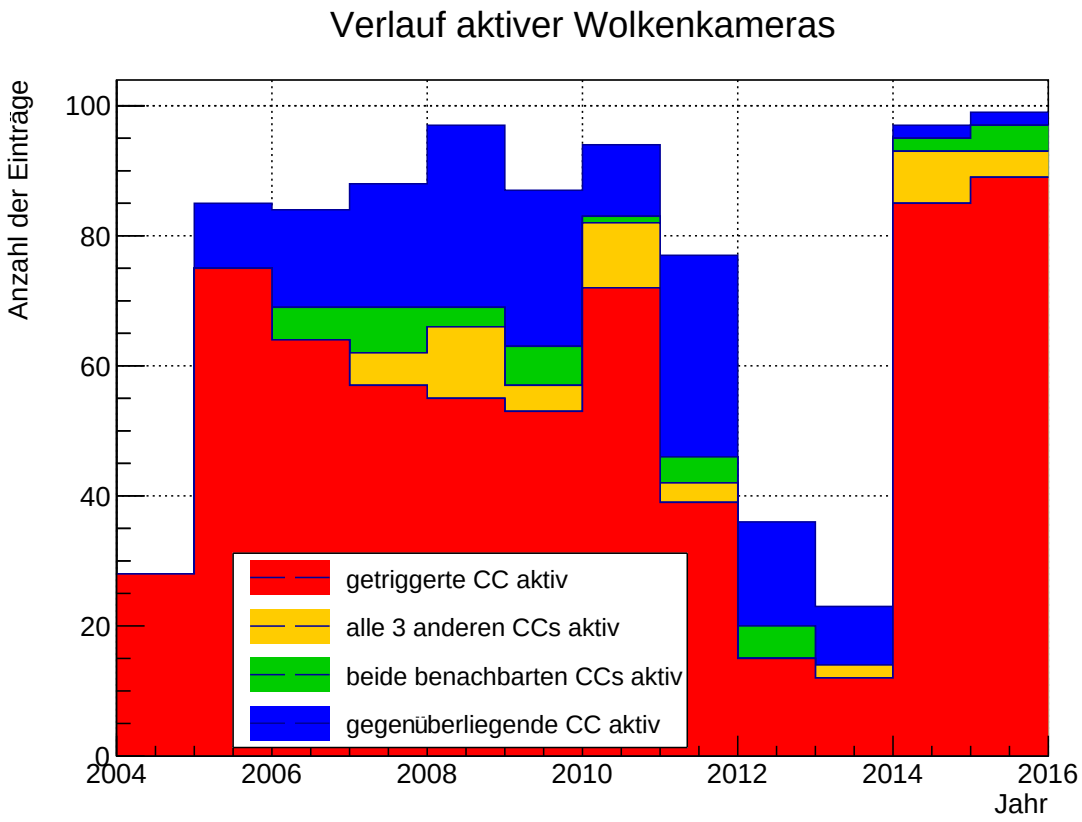



Abbildung 4.19: Der rot gefüllte Teil des Histogramms gibt den jährlichen Anteil an Ereignissen an, deren Wolkenkamera (hier CC abgekürzt) bei dem jeweils getriggerten FD aktiv war. Ist diese Kamera inaktiv, so wird gefragt, ob alle drei restlichen Kameras Wolkeninformationen aktiv waren. Dies ist der gelbe Anteil, der auf die roten Balken aufgestapelt wird. Sind nicht alle drei anderen, sondern nur die zwei benachbarten Standorte aktiv, so ist dies mit grüner Farbe dargestellt. Blau ist der Anteil der Ereignisse, an denen die gegenüberliegende Wolkenkamera aktiv war, wenn die drei anderen Bedingungen nicht erfüllt sind. Der verbleibende Anteil, der nicht ausgefüllt ist, sind somit Ereignisse, in denen nur eine benachbarte Wolkenkamera oder gar keine Wolkenkamera zum Zeitpunkt des Luftschauers aktiv war.

Im Gegensatz zu Abb. 4.18 ist die Einteilung jährlich und zwischen den vier Standorten wird nicht unterschieden. In den Jahren 2012 und 2013, in denen auf Grund des Umbaus wenig Informationen vorhanden sind, ist auch in Abb. 4.19 ein drastischer Einbruch sichtbar. Im Jahr 2012 ist in 4.18 zu sehen, dass die Kamera in Los Leones nicht in Betrieb war. Darum ist in Abb. 4.19 kein gelber Balken für dieses Jahr vorhanden. In den Jahren 2014 und 2015 ist der Anteil der Ereignisse, für die Wolkeninformationen von den Kameras zur Verfügung stehen wieder wesentlich höher. Die beiden Abb. 4.18 und 4.19 sind also konsistent miteinander.

Über den gesamten Zeitraum gesehen sind für 55,4% aller Ereignisse die Daten der Wolkenkamera verwertbar und in die Datenbank hochgeladen. Es stellt sich die Frage, ob es überhaupt lohnenswert ist, einen ausschließlich auf den Informationen der Wolkenkameras beruhenden Wolkenschnitt zu entwerfen. Dies wäre vermutlich nicht sinnvoll, wenn in der Zukunft damit zu rechnen wäre, dass z. B. jedes zweite Ereignis wegen fehlender Information über die Wolken aussortiert werden müsste. Wegen des eben beschriebenen Austauschs in den Jahren 2012 und 2013 und den guten Werten in den beiden letzten Jahren, ist aber eher davon auszugehen, dass auch in Zukunft deutlich mehr als die Hälfte der Ereignisse Wolkeninformationen von der Wolkenkamera beziehen können.
Zusätzlich sind noch weitere Farben in dem Histogramm sichtbar, die auf die einzelnen Bins aufgestapelt sind. Aus der Abb. 4.15 können Rückschlüsse darüber gezogen werden, welcher Prozentsatz der zusätzlich auswertbaren Ereignisse („gelb, grün und blau“) wolkenfrei sein werden. Insbesondere dann, wenn auch der „blaue Prozentsatz“ in Abb. 4.19 berücksichtigt wird, so wird sich eine nennenswerte Zahl zusätzlicher Ereignisse ergeben. Theoretisch könnte noch weiter gegangen und gesagt werden, dass wenn von der Kamera in Los Leones eine Wolke detektiert wurde, von der gegenüberliegenden Kamera in Loma Amarilla jedoch nicht, dass in diesem Fall die Wolke, aus Sicht des FD in Los Leones hinter dem Areal und über oder hinter Loma Amarilla liegt und somit der Bereich über dem Areal wolkenfrei ist. Demnach würde es ausreichend sein, wenn eine der vier Kameras ein komplett wolkenfreies Sichtfeld hat.
Um eine Übersicht über die Größenordnungen ohne jährliche Unterteilung zu bekommen, ist im Folgenden noch eine Übersicht abgebildet.

| In 44,7% der Ereignisse ist die Wolkenkamera des getriggerten FD inaktiv | | |
|---|---|---|
| Bei diesen 44,7% ist/sind | | rel. Anteil bezogen auf die Gesamtheit |
| keine weitere Kamera aktiv | 27,4% | 12,2% |
| 1 weitere Kamera aktiv | 31,6% | 14,1% |
| 2 weitere Kameras aktiv | 30,6% | 13,7% |
| 3 weitere Kameras aktiv | 10,4% | 4,7% |

Abbildung 4.20: Ist die Kamera des getriggerten FD nicht in Betrieb, so trifft genau eine der in den nächsten Zeilen aufgelisteten Bedingungen zu. In der zweiten Spalte steht dann jeweils der Anteil von diesen 44,7%, auf den diese Bedingung zutrifft. In der dritten Spalte ist dieser Anteil dann umgerechnet, so dass er sich auf alle Ereignisse bezieht und nicht nur auf diejenigen, deren Wolkenkamera an dem getriggerten FD inaktiv war.

Werden Wolken außer Acht gelassen, so ist in 87,8% der Ereignisse mindestens eine Wolkenkamera aktiv. Das wäre das Optimum, was erreichbar ist, wenn ausschließlich Daten der Wolkenkameras für den Wolkenschnitt benutzt werden. Es soll auch noch die Frage beantwortet werden, wie groß der Anteil der wolkenfreien Ereignisse aus den farbigen Balken in 4.19 in absoluten Zahlen ist. In 53 Ereignissen waren die jeweils 3 anderen Standorte komplett wolkenfrei, in 367 Fällen waren genau zwei andere Standorte wolkenfrei, und in 1534 Ereignissen hat genau eine andere Wolkenkamera keine Wolke

beobachten können. Wenn die eigentliche Wolkenkamera nicht in Betrieb war, ist also in insgesamt 1954 Ereignissen das Areal aus Sicht von mindestens einem anderem Standort komplett wolkenfrei. Etwa 10% dieser Ereignisse stammen aus dem *removed*-Datensatz. Man erhält sogar insgesamt 5288 Ereignisse, wenn als Voraussetzung genommen wird, dass mindestens ein Standort eine durchschnittliche Wolkenbedeckung kleiner als 1% (statt vorher 0%) haben muss. Der „all_CL2_CO3“ Datensatz mit 7941 Ereignissen (siehe Abb. 4.4), in denen definitiv keine Wolke ist, würde auf diese Weise um 70% vergrößert werden.

Um dem Optimum von 87,8% noch näher zu kommen, wäre es eine Möglichkeit den Standort des Schauers, beispielsweise die GPS-Koordinate des Einschlagpunkts zu berücksichtigen. Dadurch müsste nicht ein komplettes Bild einer Kamera wolkenfrei sein, sondern nur die Teleskope, die in die Richtung des Schauers und in den Zwischenraum zwischen Schauer und getriggertem Detektor gerichtet sind.
Auch wenn nicht für jedes Ereignis der vergangenen Jahre Informationen vorliegen, so ist an Abb. 4.19 erkennbar, dass für zukünftige Jahre bei gleicher Tendenz damit zu rechnen ist, dass allein die Informationen der Wolkenkameras ausreichen werden, um Ereignisse als wolkenfrei zu klassifizieren. Durch Zuhilfennahme benachbarter Kameras wird man nahe an die 100% herankommen.

# 5. Zusammenfassung der Ergebnisse

Im ersten Teil der Analyse sind verschiedene Datensätze mittels der $X_{\max}$ -Analyse miteinander verglichen worden. Es ist möglich einen Datensatz nur auf Basis der Daten der Wolkenkameras herzustellen, der zu demselben Ergebnis für $\langle X_{\max} \rangle$ führt wie der *selected*-Datensatz aus [11]. Die verschiedenen Pixelauswahlen haben keinen Einfluss auf die Ergebnisse für die $X_{\max}$ -Analyse. Auch der Vergleich zwischen dem *selected*-Datensatz und dem *removed*-Datensatz aus [11], sowie der Vergleich zwischen einem wolkenfreien und einem wolkenbehaftetem Datensatz, die jeweils mit einem eigenen Wolkenschnitt erstellt wurden, hat keinen entscheidenden Einfluss von Wolken auf die Ergebnisse der $X_{\max}$ -Analyse ergeben. Die Unterschiede liegen alle im statistisch nicht signifikanten Bereich. Die Ursache dafür könnte sein, dass tatsächlich wolkenbehaftete Ereignisse bereits im Vorfeld durch andere Schnitte aussortiert worden sind und so in dem verbleibendem *all*-Datensatz schon zum großen Teil keine Wolken mehr zwischen Luftschauer und Detektor sind. Aus diesem Grund wurden weitere Fragestellungen untersucht. Wenn keine Informationen über Wolken vorliegen, so hat der Himmel ca. 70% der Zeit eine Wolkenbedeckung weniger als 10%. Außerdem ist ein wolkenloser Himmel ein langsam veränderlicher Zustand. Im letzten Abschnitt wurde gezeigt, dass wolkenfreie Datensätze auch durch die Hilfe anderer Wolkenkameras vergrößert werden können. Dass Ereignisse trotz inaktiver Kamera durch andere Kameras als wolkenfrei zu deklariert werden können, ist ein vielversprechender Ansatz um in Zukunft mehr zuverlässige Statistik allein aus Wolkenkameradaten zu erhalten.

# Danksagung

An dieser Stelle möchte ich noch denjenigen danken, die das Schreiben dieser Arbeit ermöglicht haben. Dazu gehören Herr Prof. Dr. Markus Risse, der mir das Angebot gemacht hat, über dieses Thema eine Bachelorarbeit zu verfassen, und Dr. Marcus Niechciol, der immer mit Beratung und Hilfestellung zur Seite stand. Außerdem danke ich denjenigen, die sich die Zeit genommen haben, diese Arbeit auf Fehler und Unverständlichkeiten zu überprüfen und somit sicherlich zur Qualität dieser Arbeit beigetragen haben.

## Literatur


[1] private Kommunikation mit Marcus Niechciol, Universität Siegen, Oktober 2017

[2] Johannes Blümer, Ralph Engel, und Jörg R. Hörandel, Cosmic Rays from the Knee to the Highest Energies, Progress in Particle and Nuclear Physics 63 (2009), 293–338

[3] Malcolm S. Longair, High Energy Astrophysics, Volume 1: Particles, photons and their detection, second edition, Cambridge University Press, Cambridge, 1992.

[4] Homij. Bhabha und Walter Heitler, The Passage of Fast Electrons and the Theory of Cosmic Showers, Proceedings of the Royal Society of London A159 (1937), 432–458

[5] Maurizio Spuriou, Particles and Astrophysics, Springer International Publishing, 2014

[6] A. Aab et al., The Pierre Auger Cosmic Ray Observatory, Nuclear Instruments and Methods in Physics Research A 798 (2015), 172–213

[7] Jorge Abraham et al. (Pierre Auger Kollaboration), The fluorescence detector of the Pierre Auger Observatory, Nuclear Instruments and Methods in Physics Research A 620 (2010), 227–251

[8] Jorge Abraham et al. (Pierre Auger Kollaboration), A study of the effect of molecular and aerosol conditions in the atmosphere on air fluorescence measurements at the Pierre Auger Observatory, Astroparticle Physics 33 (2010), 108–129

[9] Patrick van Bodegom, Trent Grubb, Roger Clay, Bruce Dawson, Cloud Camera Analysis and Database Update, The University of Adelaide, März 2017,

[10] Daniel Kuempel, Karl-Heinz Kampert, Markus Risse, Geometry reconstruction of fluorescence detectors revisited, Astroparticle Physics 30 (2008), 167–174

[11] Jose Bellido et al. (Pierre Auger Kollaboration), Depth of maximum of air-shower profiles at the Pierre Auger Observatory: Measurements above 1017.2 eV and Composition Implications, PoS(ICRC2017)

[12] https://web.ikp.kit.edu/munger/Xmax/xmaxPaper2014/suppl_01_09_2014.pdf, interne Publikation der Pierre-Auger-Kollaboration

[13] http://www.physics.adelaide.edu.au/astrophysics/Auger/CloudCams

[14] private Kommunikation mit Patrick van Bodegom, University of Adelaide, Oktober 2017

# A. Plots

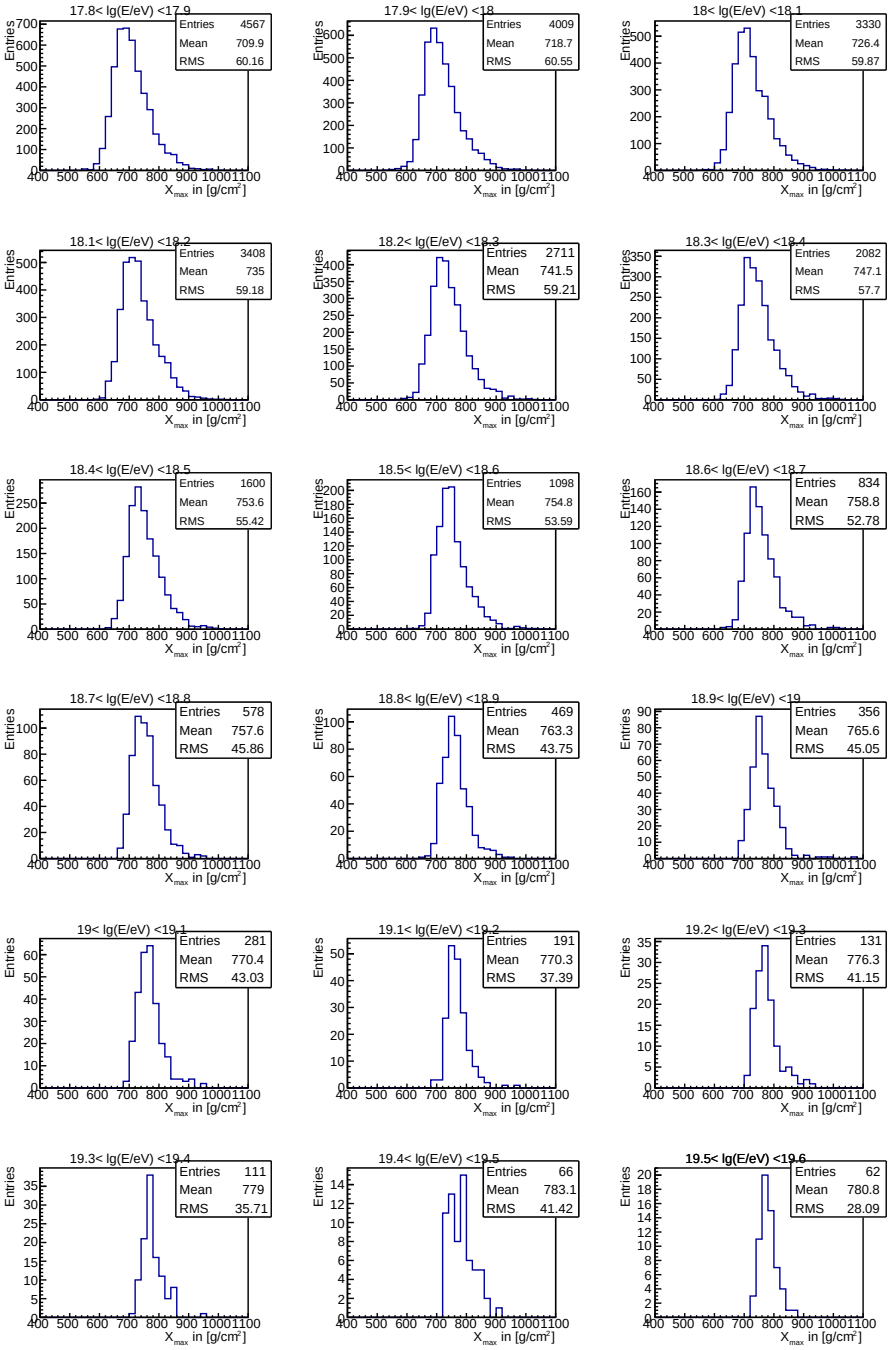



Abbildung A.1: 18 $X_{\max}$ -Histogramme, die identisch mit denen in [11] sind. In den Statistikboxen stehen jeweils die Anzahl der Einträge, der Mittelwert und die Breite der Verteilung.

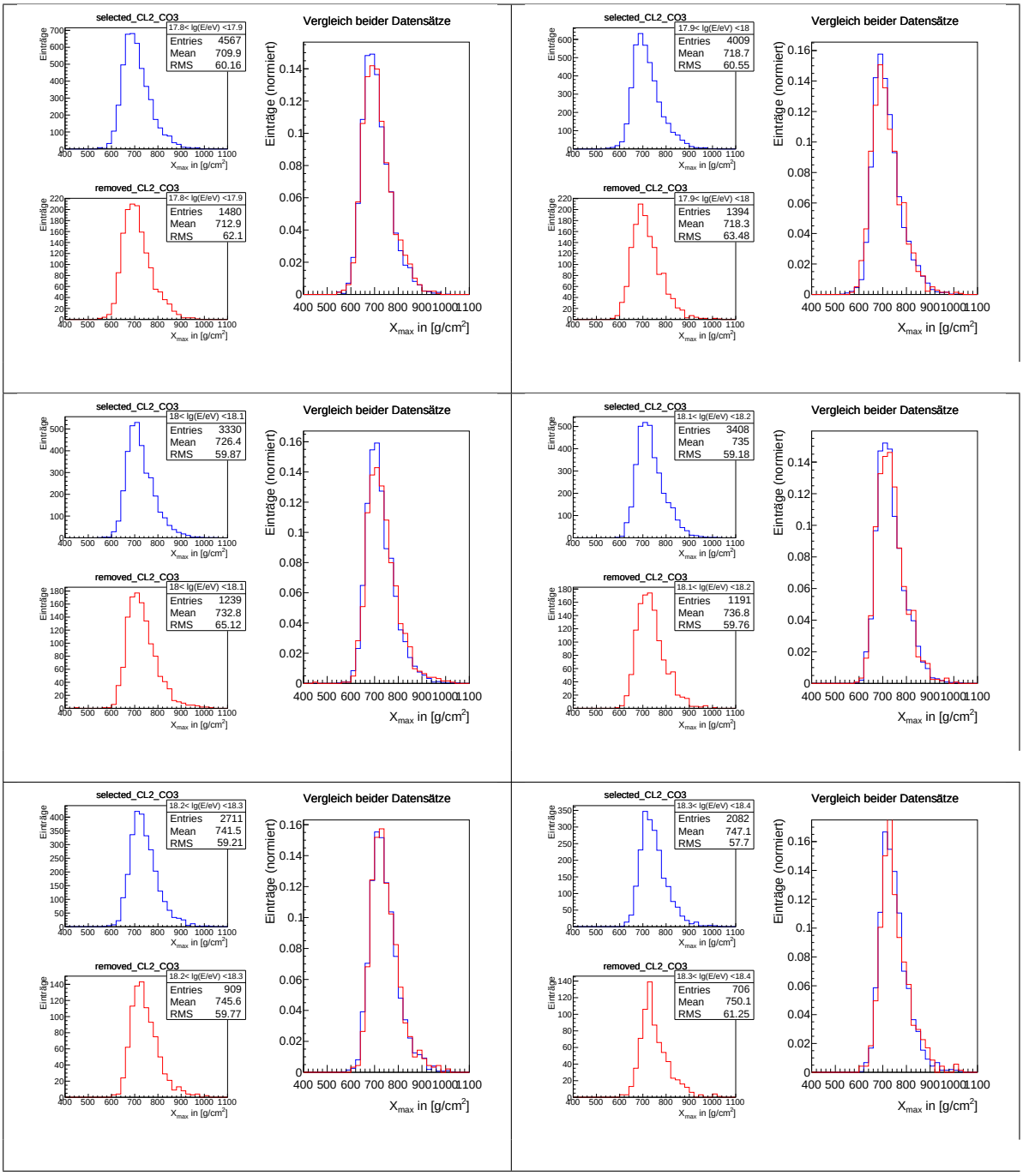



Abbildung A.2: Vergleich zwischen dem *selected*-Datensatz aus [11] mit dem *removed*-Datensatz . Zu sehen sind die $X_{\max}$ -Histogramme der Energieintervalle von $10^{17,8}$ eV - $10^{18,4}$ eV. Auf der jeweils linken Seite sind die beiden Histogramme des *selected*-Datensatzes und des *removed*-Datensatzes einzeln zu sehen. Auf der jeweils rechten Seite sind die beiden Histogramme dann überlagert.

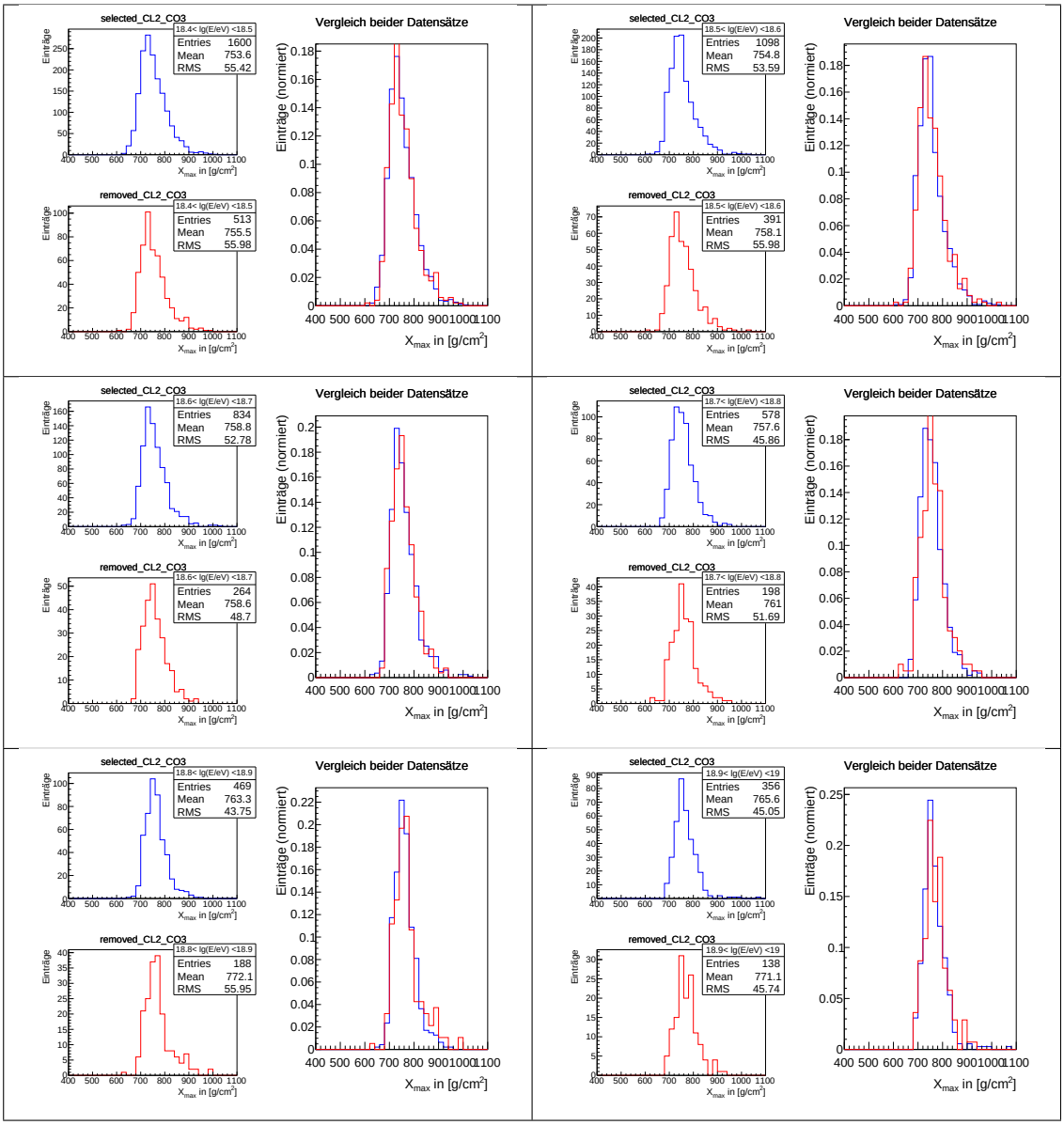



Abbildung A.3: Vergleich zwischen dem *selected*-Datensatz aus [11] mit dem *removed*-Datensatz . Zu sehen sind die $X_{\max}$ -Histogramme der Energieintervalle von $10^{18,4}$ eV - $10^{19,0}$eV. Auf der jeweils linken Seite sind die beiden Histogramme des *selected*-Datensatzes und des *removed*-Datensatzes einzeln zu sehen. Auf der jeweils rechten Seite sind die beiden Histogramme dann überlagert.

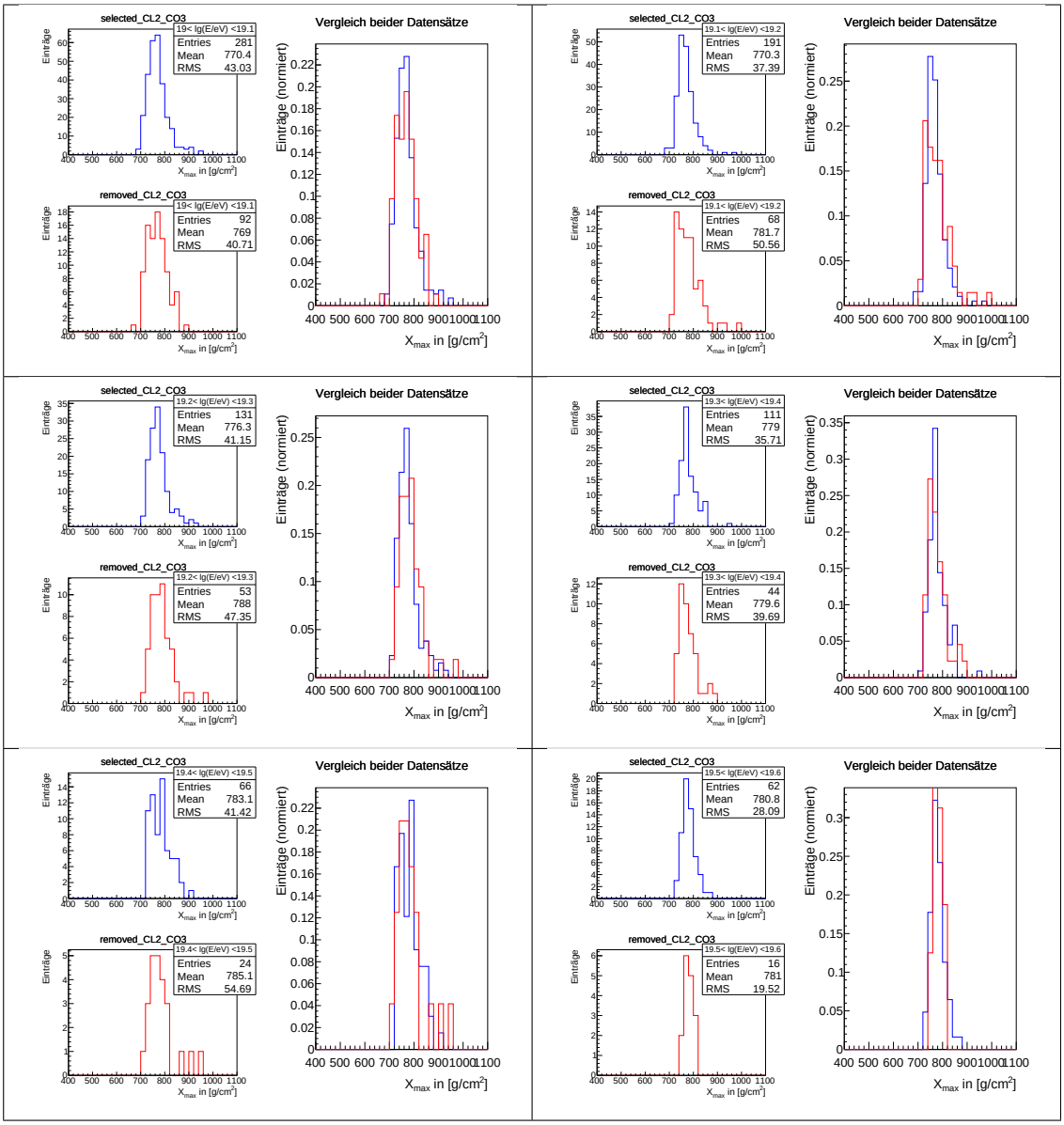



Abbildung A.4: Vergleich zwischen dem *selected*-Datensatz aus [11] mit dem *removed*-Datensatz . Zu sehen sind die $X_{\max}$ -Histogramme der Energieintervalle mit einer Energie größer als $10^{19,0}$ eV. Auf der jeweils linken Seite sind die beiden Histogramme des *selected*-Datensatzes und des *removed*-Datensatzes einzeln zu sehen. Auf der jeweils rechten Seite sind die beiden Histogramme dann überlagert.

# B. Programmcode

Dies ist der Code, der verwendet wurde, um die nötigen Informationen zu den Teilchenschauern aus den Auger-Datenbanken zu ziehen.

```
/*=========================================================
=========================================================
   File: Datenextraktion.cc

   Author: Marcus Niechciol und
           Silas Rodekamp

   Usage: analysis -c [config file]
=========================================================
=========================================================*/

#include <vector>
#include <string>
#include <iostream>
#include <map>

#include "ConfigFile.h"

#include <FdCloudCameraData.h>
#include <DetectorGeometry.h>
#include <RecEvent.h>
#include <RecEventFile.h>
#include <FdRecShower.h>
#include <FDEvent.h>
#include <FdRecLevel.h>
#include <FileInfo.h>

#include <TFile.h>
#include <TTree.h>

using namespace std;

/*=========================================================

   Declaration of variables and functions

=========================================================*/

string gConfigFileName, gOutputFileName;

vector<string> inputFileNames;

int SDiD ;
int counterTotal(0), counterGood(0), TelID;

double  gEHybrid, gXmax ;
vector<int>  pixel_vec;

bool HasLL,HasLM,HasLA,HasCO;  // Los Leones, ...
int gpssecondCO,gpssecondLA,gpssecondLL,gpssecondLM;

double gXmaxCO, gXmaxLA,gXmaxLM,gXmaxLL;
double gEHybridCO, gEHybridLA, gEHybridLM, gEHybridLL;
double gEHybridCOError, gEHybridLAError, gEHybridLMError, gEHybridLLError;
double gXmaxCOError, gXmaxLAError,gXmaxLMError,gXmaxLLError;

vector<int> activemirrorLL,activemirrorLM,activemirrorLA,activemirrorCO;
vector<int> pixelTel1LL, pixelTel2LL, pixelTel3LL, pixelTel4LL, pixelTel5LL,
    pixelTel6LL;
vector<int> pixelTel1LM, pixelTel2LM, pixelTel3LM, pixelTel4LM, pixelTel5LM,
    pixelTel6LM;
vector<int> pixelTel1LA, pixelTel2LA, pixelTel3LA, pixelTel4LA, pixelTel5LA,
    pixelTel6LA;
vector<int> pixelTel1CO, pixelTel2CO, pixelTel3CO, pixelTel4CO, pixelTel5CO,
    pixelTel6CO;
```



```
int getOptions(int argc, char** argv);
void setIntVectorsToDefault (vector<int> defaultvec, vector<int> &v1, vector<int>
    &v2, vector<int> &v3, vector<int> &v4, vector<int> &v5, vector<int> &v6);
void createActiveMirrorVec (vector<int> &activemirror, vector<int> &v1, vector<int>
    &v2, vector<int> &v3, vector<int> &v4, vector<int> &v5, vector<int> &v6);


/*=========================================================
   Main function
=========================================================*/

int main(int argc, char **argv)
{
  const int nOptions = getOptions(argc, argv);
  ConfigFile config(gConfigFileName.c_str());
  string gInputListFileName;
  config.readInto(gInputListFileName, "InputListFileName");
  string line;
  ifstream inputStream(gInputListFileName.c_str());
  while(!inputStream.eof())  {
    getline(inputStream, line);
    inputFileNames.push_back(line);
  }
  inputStream.close();

  RecEventFile inputFile(inputFileNames);
  RecEvent *theRecEvent = new RecEvent();
  DetectorGeometry theGeo;
  inputFile.ReadDetectorGeometry(theGeo);
  FileInfo theInfo;
  inputFile.ReadFileInfo(theInfo);
  inputFile.SetBuffers(&theRecEvent);
  config.readInto(gOutputFileName, "OutputFileName");

  TFile *outputFile = new TFile(gOutputFileName.c_str(), "RECREATE");
  TTree *tree = new TTree("Tree", "Tree");
  tree->Branch("SDID", &SDiD );
  tree->Branch("HasLL", &HasLL);  tree->Branch("HasLM", &HasLM);
  tree->Branch("HasLA", &HasLA);  tree->Branch("HasCO", &HasCO);
  tree->Branch("gpssecondLL",&gpssecondLL);  tree->Branch("gpssecondLM",&gpssecondLM);
  tree->Branch("gpssecondLA",&gpssecondLA);  tree->Branch("gpssecondCO",&gpssecondCO);
  tree->Branch("gEHybridLL", &gEHybridLL);  tree->Branch("gEHybridLM", &gEHybridLM);
  tree->Branch("gEHybridLA", &gEHybridLA);  tree->Branch("gEHybridCO", &gEHybridCO);
  tree->Branch("gEHybridLLError", &gEHybridLLError);
  tree->Branch("gEHybridLMError", &gEHybridLMError);
  tree->Branch("gEHybridLAError", &gEHybridLAError);
  tree->Branch("gEHybridCOError", &gEHybridCOError);
  tree->Branch("gXmaxLL",&gXmaxLL);  tree->Branch("gXmaxLM",&gXmaxLM);
  tree->Branch("gXmaxLA",&gXmaxLA);  tree->Branch("gXmaxCO",&gXmaxCO);
  tree->Branch("gXmaxLLError",&gXmaxLLError);
  tree->Branch("gXmaxLMError",&gXmaxLMError);
  tree->Branch("gXmaxLAError",&gXmaxLAError);
  tree->Branch("gXmaxCOError",&gXmaxCOError);
  tree->Branch("activemirrorLL",&activemirrorLL);
  tree->Branch("activemirrorLM",&activemirrorLM);
  tree->Branch("activemirrorLA",&activemirrorLA);
  tree->Branch("activemirrorCO",&activemirrorCO);
  // In these branches the pixelnumber of each active pixel of each telescope of
    each site is saved
  tree->Branch("pixelTel1LL",&pixelTel1LL); tree->Branch("pixelTel2LL",&pixelTel2LL);
  tree->Branch("pixelTel3LL",&pixelTel3LL); tree->Branch("pixelTel4LL",&pixelTel4LL);
  tree->Branch("pixelTel5LL",&pixelTel5LL); tree->Branch("pixelTel6LL",&pixelTel6LL);
  tree->Branch("pixelTel1LM",&pixelTel1LM); tree->Branch("pixelTel2LM",&pixelTel2LM);
  tree->Branch("pixelTel3LM",&pixelTel3LM); tree->Branch("pixelTel4LM",&pixelTel4LM);
  tree->Branch("pixelTel5LM",&pixelTel5LM); tree->Branch("pixelTel6LM",&pixelTel6LM);
  tree->Branch("pixelTel1LA",&pixelTel1LA); tree->Branch("pixelTel2LA",&pixelTel2LA);
  tree->Branch("pixelTel3LA",&pixelTel3LA); tree->Branch("pixelTel4LA",&pixelTel4LA);
  tree->Branch("pixelTel5LA",&pixelTel5LA); tree->Branch("pixelTel6LA",&pixelTel6LA);
  tree->Branch("pixelTel1CO",&pixelTel1CO); tree->Branch("pixelTel2CO",&pixelTel2CO);
```



```
  tree->Branch("pixelTel3CO",&pixelTel3CO); tree->Branch("pixelTel4CO",&pixelTel4CO);
  tree->Branch("pixelTel5CO",&pixelTel5CO); tree->Branch("pixelTel6CO",&pixelTel6CO);


  while (inputFile.ReadNextEvent() == RecEventFile::eSuccess)
  {
    counterTotal++;
    cout << endl << "====================" << endl;
    cout <<  "Reading Event Number: " << counterTotal << " of " <<
    inputFile.GetNEvents() << endl << endl;
    cout << "  Event ID: " << theRecEvent->GetSDEvent().GetEventId() << "  YYMMDD: "
    << theRecEvent->GetYYMMDD()  << endl;
    SDiD = theRecEvent->GetSDEvent().GetEventId();

    //Define the start/default values of each event
    HasLL = false;    HasLM = false;    HasLA = false;    HasCO = false;
    gpssecondLL = -100; gpssecondLM = -100; gpssecondLA = -100; gpssecondCO = -100;
    gEHybridLL = -100; gEHybridLM = -100; gEHybridLA = -100; gEHybridCO = -100;
    gEHybridLLError = -100;    gEHybridLMError = -100;
    gEHybridLAError = -100;    gEHybridCOError = -100;
    gXmaxLL = -100; gXmaxLM = -100; gXmaxLA = -100; gXmaxCO = -100;
    gXmaxLLError = -100; gXmaxLMError = -100; gXmaxLAError = -100; gXmaxCOError = -100;
    vector<int> activemirrorstart (6,0);
    activemirrorLL = activemirrorstart;    activemirrorLM = activemirrorstart;
    activemirrorLA = activemirrorstart;    activemirrorCO = activemirrorstart;

    pixelTel1LL.clear();    pixelTel2LL.clear();
    pixelTel3LL.clear();    pixelTel4LL.clear();
    pixelTel5LL.clear();    pixelTel6LL.clear();
    pixelTel1LM.clear();    pixelTel2LM.clear();
    pixelTel3LM.clear();    pixelTel4LM.clear();
    pixelTel5LM.clear();    pixelTel6LM.clear();
    pixelTel1LA.clear();    pixelTel2LA.clear();
    pixelTel3LA.clear();    pixelTel4LA.clear();
    pixelTel5LA.clear();    pixelTel6LA.clear();
    pixelTel1CO.clear();    pixelTel2CO.clear();
    pixelTel3CO.clear();    pixelTel4CO.clear();
    pixelTel5CO.clear();    pixelTel6CO.clear();

    vector<FDEvent> FDEvents = theRecEvent->GetFDEvents();

    for(int i=0; i< FDEvents.size(); ++i)             // loop over all active
    sides/Eyes / FD Stations
    {
      int EyeID = FDEvents.at(i).GetEyeId();
      if(EyeID == 1) {
        HasLL = true;  // Los Leones is ACTIVE
        // get information about the shower
        gpssecondLL = FDEvents.at(i).GetGPSSecond();
        gEHybridLL = FDEvents.at(i).GetFdRecShower().GetEnergy();
        gXmaxLL = FDEvents.at(i).GetFdRecShower().GetXmax();
        gEHybridLLError = FDEvents.at(i).GetFdRecShower().GetEnergyError();
        gXmaxLLError = FDEvents.at(i).GetFdRecShower().GetXmaxError();

      // get the active pixelnumbers and convert them a the pixelnumber from 1-440
        vector<UShort_t> pixelLL = FDEvents.at(i).GetFdRecPixel().GetID();
        for (int k=0 ; k < pixelLL.size(); ++k){
          if ( FDEvents.at(i).GetFdRecPixel().GetStatus(k) == 4) // 4 is the
    status for the triggered timepixels (colored in the EventBrowser)
            {
              TelID = FDEvents.at(i).GetFdRecPixel().GetTelescopeId(k);
              if (TelID==1)
    {pixelTel1LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
              if (TelID==2)
    {pixelTel2LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
              if (TelID==3)
    {pixelTel3LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
              if (TelID==4)
    {pixelTel4LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
```



```
              if (TelID==5)
    {pixelTel5LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
              if (TelID==6)
    {pixelTel6LL.push_back((pixelLL.at(k)+1)-(440*(TelID-1)));}
            }
          }
          createActiveMirrorVec (activemirrorLL, pixelTel1LL, pixelTel2LL,
    pixelTel3LL, pixelTel4LL, pixelTel5LL, pixelTel6LL);
      }

      if(EyeID == 2)  {
    HasLM = true;
    gpssecondLM = FDEvents.at(i).GetGPSSecond();
    gEHybridLM = FDEvents.at(i).GetFdRecShower().GetEnergy();
    gXmaxLM = FDEvents.at(i).GetFdRecShower().GetXmax();
    gEHybridLMError = FDEvents.at(i).GetFdRecShower().GetEnergyError();
    gXmaxLMError = FDEvents.at(i).GetFdRecShower().GetXmaxError();
    vector<UShort_t> pixelLM = FDEvents.at(i).GetFdRecPixel().GetID();
    for (int k=0 ; k < pixelLM.size(); ++k) {
        if ( FDEvents.at(i).GetFdRecPixel().GetStatus(k) == 4  )  {
        TelID = FDEvents.at(i).GetFdRecPixel().GetTelescopeId(k);
        if (TelID == 1) {pixelTel1LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 2) {pixelTel2LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 3) {pixelTel3LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 4) {pixelTel4LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 5) {pixelTel5LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 6) {pixelTel6LM.push_back((pixelLM.at(k)+1)-(440*(TelID-1)));}
        }
    }
    createActiveMirrorVec (activemirrorLM, pixelTel1LM, pixelTel2LM, pixelTel3LM,
    pixelTel4LM, pixelTel5LM, pixelTel6LM);
      }


      if(EyeID == 3)  {
    HasLA = true;
    gpssecondLA = FDEvents.at(i).GetGPSSecond();
    gEHybridLA = FDEvents.at(i).GetFdRecShower().GetEnergy();
    gXmaxLA = FDEvents.at(i).GetFdRecShower().GetXmax();
    gEHybridLAError = FDEvents.at(i).GetFdRecShower().GetEnergyError();
    gXmaxLAError = FDEvents.at(i).GetFdRecShower().GetXmaxError();

    vector<UShort_t> pixelLA = FDEvents.at(i).GetFdRecPixel().GetID();
    for (int k=0 ; k < pixelLA.size(); ++k) {
        if ( FDEvents.at(i).GetFdRecPixel().GetStatus(k) == 4  )  {
        TelID = FDEvents.at(i).GetFdRecPixel().GetTelescopeId(k);
        if (TelID == 1) {pixelTel1LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 2) {pixelTel2LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 3) {pixelTel3LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 4) {pixelTel4LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 5) {pixelTel5LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 6) {pixelTel6LA.push_back((pixelLA.at(k)+1)-(440*(TelID-1)));}
        }
    }
    createActiveMirrorVec (activemirrorLA, pixelTel1LA, pixelTel2LA, pixelTel3LA,
    pixelTel4LA, pixelTel5LA, pixelTel6LA);
      }


      if(EyeID == 4)  {
    HasCO = true;
    gpssecondCO = FDEvents.at(i).GetGPSSecond();
    gEHybridCO = FDEvents.at(i).GetFdRecShower().GetEnergy();
    gXmaxCO = FDEvents.at(i).GetFdRecShower().GetXmax();
    gEHybridCOError = FDEvents.at(i).GetFdRecShower().GetEnergyError();
    gXmaxCOError = FDEvents.at(i).GetFdRecShower().GetXmaxError();

    vector<UShort_t> pixelCO = FDEvents.at(i).GetFdRecPixel().GetID();
    for (int k=0 ; k < pixelCO.size(); ++k) {
```

```
        if ( FDEvents.at(i).GetFdRecPixel().GetStatus(k) == 4  ) {
        TelID = FDEvents.at(i).GetFdRecPixel().GetTelescopeId(k);
        if (TelID == 1) {pixelTel1CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 2) {pixelTel2CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 3) {pixelTel3CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 4) {pixelTel4CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 5) {pixelTel5CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        if (TelID == 6) {pixelTel6CO.push_back((pixelCO.at(k)+1)-(440*(TelID-1)));}
        }
    }
    createActiveMirrorVec (activemirrorCO, pixelTel1CO, pixelTel2CO, pixelTel3CO,
    pixelTel4CO, pixelTel5CO, pixelTel6CO);
      }
    }

    // Set all the default values if a site/telescope is not active
    vector<int> activemirrordefault (6,-100);
    if (activemirrorLL == activemirrorstart){activemirrorLL = activemirrordefault;}
    if (activemirrorLM == activemirrorstart){activemirrorLM = activemirrordefault;}
    if (activemirrorLA == activemirrorstart){activemirrorLA = activemirrordefault;}
    if (activemirrorCO == activemirrorstart){activemirrorCO = activemirrordefault;}
    vector<int> pixelteldefault (1,-100);  // Set vectors to default if they are empty
    setIntVectorsToDefault(pixelteldefault, pixelTel1LL, pixelTel2LL, pixelTel3LL,
    pixelTel4LL, pixelTel5LL, pixelTel6LL);
    setIntVectorsToDefault(pixelteldefault, pixelTel1LM, pixelTel2LA, pixelTel3LA,
    pixelTel4LM, pixelTel5LM, pixelTel6LM);
    setIntVectorsToDefault(pixelteldefault, pixelTel1LA, pixelTel2LM, pixelTel3LM,
    pixelTel4LA, pixelTel5LA, pixelTel6LA);
    setIntVectorsToDefault(pixelteldefault, pixelTel1CO, pixelTel2CO, pixelTel3CO,
    pixelTel4CO, pixelTel5CO, pixelTel6CO);

    counterGood++;
    tree->Fill();
  }  // the program starts reading the next event

  inputFile.Close();

  cout << endl;
  cout << "************************************************** " << endl;
  cout << "                    STATISTICS                    " << endl;
  cout << "************************************************** " << endl << endl;
  cout << "Total Number of Events in Input Files:    " <<inputFile.GetNEvents() << endl;
  cout << "Total Number of Events read:              " <<counterTotal << endl;
  cout << "Good Events:                              " <<counterGood<< endl<<endl;
  cout << "************************************************** " << endl<<endl;

  outputFile->cd();
  tree->Write();
  outputFile->Close();
  return 0;
}
// ================== DEFINE FUNCTIONS =====================

/*============================================================
   Reads command line options
============================================================*/
int getOptions(int argc, char **argv)
{
  int a;
  while ((a = getopt(argc, argv, "c:h")) != -1) {
    switch (a)
    {
    case 'c':
      gConfigFileName = string(optarg);
      cout << " use config file:\n  " << gConfigFileName << endl;
      break;
    case 'h':
      return -2;
    default:
```



```
      return -2;
    }
  }
  cout << endl;
  return optind;
}

void setIntVectorsToDefault (vector<int> defaultvec, vector<int> &v1, vector<int>
    &v2, vector<int> &v3, vector<int> &v4, vector<int> &v5, vector<int> &v6)
{
    if (v1.empty()){v1 = defaultvec;}    if (v2.empty()){v2 = defaultvec;}
    if (v3.empty()){v3 = defaultvec;}    if (v4.empty()){v4 = defaultvec;}
    if (v5.empty()){v5 = defaultvec;}    if (v6.empty()){v6 = defaultvec;}
}

void createActiveMirrorVec (vector<int> &activemirror, vector<int> &v1, vector<int>
    &v2, vector<int> &v3, vector<int> &v4, vector<int> &v5, vector<int> &v6)
{
  if (!v1.empty()){activemirror.at(0) = 1;}   if (!v2.empty()){activemirror.at(1) = 1;}
  if (!v3.empty()){activemirror.at(2) = 1;}   if (!v4.empty()){activemirror.at(3) = 1;}
  if (!v5.empty()){activemirror.at(4) = 1;}   if (!v6.empty()){activemirror.at(5) = 1;}
  // if there is an entry (here a pixelnumber) in the vector, then the mirror is active
}
```



Dieses Programm schreibt die Wolkeninformationen zu den aktiven Pixeln aus der Wolkendatenbank heraus.

```
#include <iomanip>
#include <sstream>
using namespace std;

#include <evt/Event.h>
using namespace evt;

#include <det/Detector.h>
using namespace det;

#include <fdet/FDetector.h>
#include <fdet/Eye.h>
#include <fdet/Telescope.h>
#include <fdet/Pixel.h>
using namespace fdet;

#include <utl/ErrorLogger.h>
#include <utl/TimeStamp.h>
using namespace utl;

#include "ReadCloud.h"
using namespace ReadCloudNS;
using namespace fwk;

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

ReadCloud::ReadCloud()
{
}


ReadCloud::~ReadCloud()
{
}


VModule::ResultFlag
ReadCloud::Init()
{
  INFO("ReadCloud::Init()");
  return eSuccess;
}


VModule::ResultFlag
ReadCloud::Run(evt::Event& /*event*/)
{
  INFO("ReadCloud::Run()");

  // Initialize variables

  bool HasLL , HasLM, HasLA, HasCO ;  // Los Leones, ...
  bool noPixelStatus;

  int gpssecondCO,gpssecondLA,gpssecondLL,gpssecondLM;

  vector<int> * activemirrorLL = new vector<int>();
  vector<int> * activemirrorLM = new vector<int>();
  vector<int> * activemirrorLA = new vector<int>();
  vector<int> * activemirrorCO = new vector<int>();

  vector<int> *pixelTel1LL=new vector<int>();vector<int> *pixelTel2LL=new vector<int>();
```



```
  vector<int> *pixelTel3LL=new vector<int>();vector<int> *pixelTel4LL=new vector<int>();
  vector<int> *pixelTel5LL=new vector<int>();vector<int> *pixelTel6LL=new vector<int>();
  vector<int> *pixelTel1LM=new vector<int>();vector<int> *pixelTel2LM=new vector<int>();
  vector<int> *pixelTel3LM=new vector<int>();vector<int> *pixelTel4LM=new vector<int>();
  vector<int> *pixelTel5LM=new vector<int>();vector<int> *pixelTel6LM=new vector<int>();
  vector<int> *pixelTel1LA=new vector<int>();vector<int> *pixelTel2LA=new vector<int>();
  vector<int> *pixelTel3LA=new vector<int>();vector<int> *pixelTel4LA=new vector<int>();
  vector<int> *pixelTel5LA=new vector<int>();vector<int> *pixelTel6LA=new vector<int>();
  vector<int> *pixelTel1CO=new vector<int>();vector<int> *pixelTel2CO=new vector<int>();
  vector<int> *pixelTel3CO=new vector<int>();vector<int> *pixelTel4CO=new vector<int>();
  vector<int> *pixelTel5CO=new vector<int>();vector<int> *pixelTel6CO=new vector<int>();

  // define new variables
  vector<double> cfcoveragedefault (440,2);
  vector<double> pixelcfTel1LL, pixelcfTel2LL, pixelcfTel3LL, pixelcfTel4LL,
    pixelcfTel5LL, pixelcfTel6LL;
  vector<double> pixelcfTel1LM, pixelcfTel2LM, pixelcfTel3LM, pixelcfTel4LM,
    pixelcfTel5LM, pixelcfTel6LM;
  vector<double> pixelcfTel1LA, pixelcfTel2LA, pixelcfTel3LA, pixelcfTel4LA,
    pixelcfTel5LA, pixelcfTel6LA;
  vector<double> pixelcfTel1CO, pixelcfTel2CO, pixelcfTel3CO, pixelcfTel4CO,
    pixelcfTel5CO, pixelcfTel6CO;

  int counterTotal(0),SDiD;

  // Read the input-root File   (produced from analysis.cc)
  TFile *file = new TFile("output.root", "UPDATE");
  TTree *tree = (TTree*)file->Get("Tree");

  if (!file || file->IsZombie())  //gives an error, If the root file can't be opened
  {
    cout <<"----  The ROOT - file could not be opened! ---  "<< endl;;
    delete file;
    return eFailure;
  }
  // Set branch adresses to use these data as the input for the cloud data extraction
  tree->SetBranchAddress("SDID", &SDiD );
  tree->SetBranchAddress("HasLL", &HasLL);  tree->SetBranchAddress("HasLM", &HasLM);
  tree->SetBranchAddress("HasLA", &HasLA);  tree->SetBranchAddress("HasCO", &HasCO);
  tree->SetBranchAddress("gpssecondLL",&gpssecondLL);
  tree->SetBranchAddress("gpssecondLM",&gpssecondLM);
  tree->SetBranchAddress("gpssecondLA",&gpssecondLA);
  tree->SetBranchAddress("gpssecondCO",&gpssecondCO);

  tree->SetBranchAddress("activemirrorLL",&activemirrorLL);
  tree->SetBranchAddress("activemirrorLM",&activemirrorLM);
  tree->SetBranchAddress("activemirrorLA",&activemirrorLA);
  tree->SetBranchAddress("activemirrorCO",&activemirrorCO);

  tree->SetBranchAddress("pixelTel1LL",&pixelTel1LL);
  tree->SetBranchAddress("pixelTel2LL",&pixelTel2LL);
  tree->SetBranchAddress("pixelTel3LL",&pixelTel3LL);
  tree->SetBranchAddress("pixelTel4LL",&pixelTel4LL);
  tree->SetBranchAddress("pixelTel5LL",&pixelTel5LL);
  tree->SetBranchAddress("pixelTel6LL",&pixelTel6LL);
  tree->SetBranchAddress("pixelTel1LM",&pixelTel1LM);
  tree->SetBranchAddress("pixelTel2LM",&pixelTel2LM);
  tree->SetBranchAddress("pixelTel3LM",&pixelTel3LM);
  tree->SetBranchAddress("pixelTel4LM",&pixelTel4LM);
  tree->SetBranchAddress("pixelTel5LM",&pixelTel5LM);
  tree->SetBranchAddress("pixelTel6LM",&pixelTel6LM);
  tree->SetBranchAddress("pixelTel1LA",&pixelTel1LA);
  tree->SetBranchAddress("pixelTel2LA",&pixelTel2LA);
  tree->SetBranchAddress("pixelTel3LA",&pixelTel3LA);
  tree->SetBranchAddress("pixelTel4LA",&pixelTel4LA);
  tree->SetBranchAddress("pixelTel5LA",&pixelTel5LA);
  tree->SetBranchAddress("pixelTel6LA",&pixelTel6LA);
  tree->SetBranchAddress("pixelTel1CO",&pixelTel1CO);
  tree->SetBranchAddress("pixelTel2CO",&pixelTel2CO);
```

```
  tree->SetBranchAddress("pixelTel3CO",&pixelTel3CO);
  tree->SetBranchAddress("pixelTel4CO",&pixelTel4CO);
  tree->SetBranchAddress("pixelTel5CO",&pixelTel5CO);
  tree->SetBranchAddress("pixelTel6CO",&pixelTel6CO);

  // each vector has 440 entries (the cloud fraction for each pixel)
  TBranch *pcfTel1LL = tree->Branch("pixelcfTel1LL", &pixelcfTel1LL );
  TBranch *pcfTel2LL = tree->Branch("pixelcfTel2LL", &pixelcfTel2LL );
  TBranch *pcfTel3LL = tree->Branch("pixelcfTel3LL", &pixelcfTel3LL );
  TBranch *pcfTel4LL = tree->Branch("pixelcfTel4LL", &pixelcfTel4LL );
  TBranch *pcfTel5LL = tree->Branch("pixelcfTel5LL", &pixelcfTel5LL );
  TBranch *pcfTel6LL = tree->Branch("pixelcfTel6LL", &pixelcfTel6LL );
  TBranch *pcfTel1LM = tree->Branch("pixelcfTel1LM", &pixelcfTel1LM );
  TBranch *pcfTel2LM = tree->Branch("pixelcfTel2LM", &pixelcfTel2LM );
  TBranch *pcfTel3LM = tree->Branch("pixelcfTel3LM", &pixelcfTel3LM );
  TBranch *pcfTel4LM = tree->Branch("pixelcfTel4LM", &pixelcfTel4LM );
  TBranch *pcfTel5LM = tree->Branch("pixelcfTel5LM", &pixelcfTel5LM );
  TBranch *pcfTel6LM = tree->Branch("pixelcfTel6LM", &pixelcfTel6LM );
  TBranch *pcfTel1LA = tree->Branch("pixelcfTel1LA", &pixelcfTel1LA );
  TBranch *pcfTel2LA = tree->Branch("pixelcfTel2LA", &pixelcfTel2LA );
  TBranch *pcfTel3LA = tree->Branch("pixelcfTel3LA", &pixelcfTel3LA );
  TBranch *pcfTel4LA = tree->Branch("pixelcfTel4LA", &pixelcfTel4LA );
  TBranch *pcfTel5LA = tree->Branch("pixelcfTel5LA", &pixelcfTel5LA );
  TBranch *pcfTel6LA = tree->Branch("pixelcfTel6LA", &pixelcfTel6LA );
  TBranch *pcfTel1CO = tree->Branch("pixelcfTel1CO", &pixelcfTel1CO );
  TBranch *pcfTel2CO = tree->Branch("pixelcfTel2CO", &pixelcfTel2CO );
  TBranch *pcfTel3CO = tree->Branch("pixelcfTel3CO", &pixelcfTel3CO );
  TBranch *pcfTel4CO = tree->Branch("pixelcfTel4CO", &pixelcfTel4CO );
  TBranch *pcfTel5CO = tree->Branch("pixelcfTel5CO", &pixelcfTel5CO );
  TBranch *pcfTel6CO = tree->Branch("pixelcfTel6CO", &pixelcfTel6CO );

  int nentries = tree->GetEntries();

  try
  {
    // loop over all events
    for(int i=0; i<nentries; ++i) {
      counterTotal++;
      cout << "  \n\nReading event " << (i+1) << " of " << nentries ;
      tree->GetEntry(i);
      cout << "   :     Event-SDiD: " << SDiD << "  .... " << endl;   ;

      //define a vector<vector>> containing the data for all telescopes for easier
    data handling and fill them with the vectors of active pixels
      vector<vector<int> > pixelTel1LL;
      pixelTel1LL.push_back(*pixelTel1LL);   pixelTel1LL.push_back(*pixelTel2LL);
      pixelTel1LL.push_back(*pixelTel3LL);   pixelTel1LL.push_back(*pixelTel4LL);
      pixelTel1LL.push_back(*pixelTel5LL);   pixelTel1LL.push_back(*pixelTel6LL);
      vector<vector<int> > pixelTel1LM;
      pixelTel1LM.push_back(*pixelTel1LM);   pixelTel1LM.push_back(*pixelTel2LM);
      pixelTel1LM.push_back(*pixelTel3LM);   pixelTel1LM.push_back(*pixelTel4LM);
      pixelTel1LM.push_back(*pixelTel5LM);   pixelTel1LM.push_back(*pixelTel6LM);
      vector<vector<int> > pixelTel1LA;
      pixelTel1LA.push_back(*pixelTel1LA);   pixelTel1LA.push_back(*pixelTel2LA);
      pixelTel1LA.push_back(*pixelTel3LA);   pixelTel1LA.push_back(*pixelTel4LA);
      pixelTel1LA.push_back(*pixelTel5LA);   pixelTel1LA.push_back(*pixelTel6LA);
      vector<vector<int> > pixelTel1CO;
      pixelTel1CO.push_back(*pixelTel1CO);   pixelTel1CO.push_back(*pixelTel2CO);
      pixelTel1CO.push_back(*pixelTel3CO);   pixelTel1CO.push_back(*pixelTel4CO);
      pixelTel1CO.push_back(*pixelTel5CO);   pixelTel1CO.push_back(*pixelTel6CO);

      pixelcfTel1LL.clear(); pixelcfTel2LL.clear(); pixelcfTel3LL.clear();
      pixelcfTel4LL.clear(); pixelcfTel5LL.clear(); pixelcfTel6LL.clear();
      pixelcfTel1LM.clear(); pixelcfTel2LM.clear(); pixelcfTel3LM.clear();
      pixelcfTel4LM.clear(); pixelcfTel5LM.clear(); pixelcfTel6LM.clear();
      pixelcfTel1LA.clear(); pixelcfTel2LA.clear(); pixelcfTel3LA.clear();
      pixelcfTel4LA.clear(); pixelcfTel5LA.clear(); pixelcfTel6LA.clear();
      pixelcfTel1CO.clear(); pixelcfTel2CO.clear(); pixelcfTel3CO.clear();
      pixelcfTel4CO.clear(); pixelcfTel5CO.clear(); pixelcfTel6CO.clear();
```



```

      if (HasLL == 1)  {
    int EyeID = 1;
    Detector& det = Detector::GetInstance();
    TimeStamp time(gpssecondLL);
    det.Update(time);   //Set detector time to the gpssecondLL
    const Eye& eye = det.GetFDetector().GetEye(EyeID);

    for (unsigned int l=0 ; l< activemirrorLL->size(); ++l) {
    if (activemirrorLL->at(l) )   // we only need the data from the mirror used,
    activemirrorLL looks e.g. (0,0,1,1,0,0)
      {
        noPixelStatus = false;
        for (unsigned int m = 1; m < 441; ++m) {
          float coverage = 0;
          const Pixel& pixel = eye.GetTelescope(l+1).GetPixel(m);
          if ( ! pixel.HasCloudFraction() ) {  // gives an error, if cloud fraction
    not available
        ostringstream err;
        err << " no cloud fraction for pixel " << pixel.GetId() << " eye: " <<
    pixel.GetEyeId()<< " tel:" << pixel.GetTelescopeId();
        ERROR(err.str()); noPixelStatus = true;
        break;
          }
          else {
        coverage = pixel.GetCloudFraction();
        if (l==0) {pixelcfTel1LL.push_back(coverage);}
        if (l==1) {pixelcfTel2LL.push_back(coverage);}
        if (l==2) {pixelcfTel3LL.push_back(coverage);}
        if (l==3) {pixelcfTel4LL.push_back(coverage);}
        if (l==4) {pixelcfTel5LL.push_back(coverage);}
        if (l==5) {pixelcfTel6LL.push_back(coverage);}
          }
        }
        if (noPixelStatus == true)  {
          if (l==0) {pixelcfTel1LL = cfcoveragedefault;}
          if (l==1) {pixelcfTel2LL = cfcoveragedefault;}
          if (l==2) {pixelcfTel3LL = cfcoveragedefault;}
          if (l==3) {pixelcfTel4LL = cfcoveragedefault;}
          if (l==4) {pixelcfTel5LL = cfcoveragedefault;}
          if (l==5) {pixelcfTel6LL = cfcoveragedefault;}
        }
      }   // end of the "activeMirror" loop
    }   // end of the mirror loop
      }  // end of the SiteLL-loop, same process for the other sites

      if (HasLM == 1) {
    int EyeID = 2;
    Detector& det = Detector::GetInstance();
    TimeStamp time(gpssecondLM);
    det.Update(time);   //Set detector time to the gpssecondLL
    const Eye& eye = det.GetFDetector().GetEye(EyeID);
    for (unsigned int l=0 ; l< activemirrorLM->size(); ++l) {
    if (activemirrorLM->at(l) ) {
        noPixelStatus = false;
        for (unsigned int m = 1; m < 441; ++m){
          float coverage = 0;
          const Pixel& pixel = eye.GetTelescope(l+1).GetPixel(m);
          if ( ! pixel.HasCloudFraction() )  {
        ostringstream err;
        err << " no cloud fraction for pixel " << pixel.GetId() << " eye: " <<
    pixel.GetEyeId()<< " tel:" << pixel.GetTelescopeId();
        ERROR(err.str()); noPixelStatus = true;
        break;
          }
          else {
        coverage = pixel.GetCloudFraction();
        if (l==0) {pixelcfTel1LM.push_back(coverage);}
        if (l==1) {pixelcfTel2LM.push_back(coverage);}
```



```
        if (l==2) {pixelcfTel3LM.push_back(coverage);}
        if (l==3) {pixelcfTel4LM.push_back(coverage);}
        if (l==4) {pixelcfTel5LM.push_back(coverage);}
        if (l==5) {pixelcfTel6LM.push_back(coverage);}
          }
        }
        if (noPixelStatus == true) {
          if (l==0) {pixelcfTel1LM = cfcoveragedefault;}
          if (l==1) {pixelcfTel2LM = cfcoveragedefault;}
          if (l==2) {pixelcfTel3LM = cfcoveragedefault;}
          if (l==3) {pixelcfTel4LM = cfcoveragedefault;}
          if (l==4) {pixelcfTel5LM = cfcoveragedefault;}
          if (l==5) {pixelcfTel6LM = cfcoveragedefault;}
        }
      }
    }
      }

      if (HasLA == 1){
    int EyeID = 3;
    Detector& det = Detector::GetInstance();
    TimeStamp time(gpssecondLA);
    det.Update(time);
    const Eye& eye = det.GetFDetector().GetEye(EyeID);
    for (unsigned int l=0 ; l< activemirrorLA->size(); ++l) {
    if (activemirrorLA->at(1) )  {
        noPixelStatus = false;
        for (unsigned int m = 1; m < 441; ++m) {
          float coverage = 0;
          const Pixel& pixel = eye.GetTelescope(l+1).GetPixel(m);
          if ( ! pixel.HasCloudFraction() ) {
        ostringstream err;
        err << " no cloud fraction for pixel " << pixel.GetId() << " eye: " <<
    pixel.GetEyeId()<< " tel:" << pixel.GetTelescopeId();
        ERROR(err.str()); noPixelStatus = true;
        break;
          }
          else {
        coverage = pixel.GetCloudFraction();
        if (l==0) {pixelcfTel1LA.push_back(coverage);}
        if (l==1) {pixelcfTel2LA.push_back(coverage);}
        if (l==2) {pixelcfTel3LA.push_back(coverage);}
        if (l==3) {pixelcfTel4LA.push_back(coverage);}
        if (l==4) {pixelcfTel5LA.push_back(coverage);}
        if (l==5) {pixelcfTel6LA.push_back(coverage);}
          }
        }
        if (noPixelStatus == true){
          if (l==0) {pixelcfTel1LA = cfcoveragedefault;}
          if (l==1) {pixelcfTel2LA = cfcoveragedefault;}
          if (l==2) {pixelcfTel3LA = cfcoveragedefault;}
          if (l==3) {pixelcfTel4LA = cfcoveragedefault;}
          if (l==4) {pixelcfTel5LA = cfcoveragedefault;}
          if (l==5) {pixelcfTel6LA = cfcoveragedefault;}
        }
      }
    }
      }

      if (HasCO == 1) {
    int EyeID = 4;
    Detector& det = Detector::GetInstance();
    TimeStamp time(gpssecondCO);
    det.Update(time);
    const Eye& eye = det.GetFDetector().GetEye(EyeID);
    for (unsigned int l=0 ; l< activemirrorCO->size(); ++l)   {
    if (activemirrorCO->at(1) )  {
        noPixelStatus = false;
        for (unsigned int m = 1; m < 441; ++m) {
```



```
          float coverage = 0;
          const Pixel& pixel = eye.GetTelescope(l+1).GetPixel(m);
          if ( ! pixel.HasCloudFraction() )  {
        ostringstream err;
        err << " no cloud fraction for pixel " << pixel.GetId() << " eye: " <<
    pixel.GetEyeId()<< " tel:" << pixel.GetTelescopeId();
        ERROR(err.str()); noPixelStatus = true;
        break;
          }
          else {
        coverage = pixel.GetCloudFraction();
        if (l==0) {pixelcfTel1CO.push_back(coverage);}
        if (l==1) {pixelcfTel2CO.push_back(coverage);}
        if (l==2) {pixelcfTel3CO.push_back(coverage);}
        if (l==3) {pixelcfTel4CO.push_back(coverage);}
        if (l==4) {pixelcfTel5CO.push_back(coverage);}
        if (l==5) {pixelcfTel6CO.push_back(coverage);}
          }
        }
        if (noPixelStatus == true) {
          if (l==0) {pixelcfTel1CO = cfcoveragedefault;}
          if (l==1) {pixelcfTel2CO = cfcoveragedefault;}
          if (l==2) {pixelcfTel3CO = cfcoveragedefault;}
          if (l==3) {pixelcfTel4CO = cfcoveragedefault;}
          if (l==4) {pixelcfTel5CO = cfcoveragedefault;}
          if (l==5) {pixelcfTel6CO = cfcoveragedefault;}
        }
      }
    }
      }

      // If the telescope is not used, set the default to -100
      if (pixelcfTel1LL.empty()){pixelcfTel1LL.push_back(-100);}
      if (pixelcfTel2LL.empty()){pixelcfTel2LL.push_back(-100);}
      if (pixelcfTel3LL.empty()){pixelcfTel3LL.push_back(-100);}
      if (pixelcfTel4LL.empty()){pixelcfTel4LL.push_back(-100);}
      if (pixelcfTel5LL.empty()){pixelcfTel5LL.push_back(-100);}
      if (pixelcfTel6LL.empty()){pixelcfTel6LL.push_back(-100);}
      if (pixelcfTel1LM.empty()){pixelcfTel1LM.push_back(-100);}
      if (pixelcfTel2LM.empty()){pixelcfTel2LM.push_back(-100);}
      if (pixelcfTel3LM.empty()){pixelcfTel3LM.push_back(-100);}
      if (pixelcfTel4LM.empty()){pixelcfTel4LM.push_back(-100);}
      if (pixelcfTel5LM.empty()){pixelcfTel5LM.push_back(-100);}
      if (pixelcfTel6LM.empty()){pixelcfTel6LM.push_back(-100);}
      if (pixelcfTel1LA.empty()){pixelcfTel1LA.push_back(-100);}
      if (pixelcfTel2LA.empty()){pixelcfTel2LA.push_back(-100);}
      if (pixelcfTel3LA.empty()){pixelcfTel3LA.push_back(-100);}
      if (pixelcfTel4LA.empty()){pixelcfTel4LA.push_back(-100);}
      if (pixelcfTel5LA.empty()){pixelcfTel5LA.push_back(-100);}
      if (pixelcfTel6LA.empty()){pixelcfTel6LA.push_back(-100);}
      if (pixelcfTel1CO.empty()){pixelcfTel1CO.push_back(-100);}
      if (pixelcfTel2CO.empty()){pixelcfTel2CO.push_back(-100);}
      if (pixelcfTel3CO.empty()){pixelcfTel3CO.push_back(-100);}
      if (pixelcfTel4CO.empty()){pixelcfTel4CO.push_back(-100);}
      if (pixelcfTel5CO.empty()){pixelcfTel5CO.push_back(-100);}
      if (pixelcfTel6CO.empty()){pixelcfTel6CO.push_back(-100);}

      // Write necessary data in the new branches
      pcfTel1LL->Fill(); pcfTel2LL->Fill(); pcfTel3LL->Fill();
      pcfTel4LL->Fill(); pcfTel5LL->Fill(); pcfTel6LL->Fill();
      pcfTel1LM->Fill(); pcfTel2LM->Fill(); pcfTel3LM->Fill();
      pcfTel4LM->Fill(); pcfTel5LM->Fill(); pcfTel6LM->Fill();
      pcfTel1LA->Fill(); pcfTel2LA->Fill(); pcfTel3LA->Fill();
      pcfTel4LA->Fill(); pcfTel5LA->Fill(); pcfTel6LA->Fill();
      pcfTel1CO->Fill(); pcfTel2CO->Fill(); pcfTel3CO->Fill();
      pcfTel4CO->Fill(); pcfTel5CO->Fill(); pcfTel6CO->Fill();
    }

    // end of the event loop
```

```
    file->cd();
    tree->Write("", TObject::kOverwrite);
    file->Close();
    }

  catch (const AugerException& e) {
    ERROR(e.GetMessage());
  }
  return eSuccess;
}
VModule::ResultFlag
ReadCloud::Finish()
{
  INFO("ReadCloud::Finish()");
  return eSuccess;
}
```



Mit diesem Programm wurde die $X_{\max}$-Analyse gemacht. Das Ergebnis wird in zwei verschiedene Output-Dateien geschrieben.

```
#include <vector>
#include <string>
#include <iostream>
#include <math.h>
#include <sstream>
#include <numeric>
#include <fstream>
#include <iomanip>

#include <TFile.h>
#include <TMath.h>
#include <TTree.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TROOT.h>
#include <TLatex.h>
#include <TH1D.h>
#include <TGraphErrors.h>
#include <TRandom3.h>

using namespace std;

  string ReadRootFile = "all";   // options: selected / removed / all
  double CloudLimitdefault = 2;  // (Wolkenlimit)
  int CutOrder = 3;    // (Pixelauswahl)  -1 = noCut , 0 timepixelaverage , 1 =
   first crown, 2 = second crown, 3 = all pixel average
  int nwuerfe = 10000; // higher number -> better error values
  // Initialize functions
  vector<vector<int> > FindNeighborPixel(vector<int> pixel_vec, int order = 2);
  bool CloudCut(vector<vector<int> > Pixelmatrix, vector<double> cloudfraction, int
   defineCut = CutOrder , double CloudLimit = CloudLimitdefault);
  bool HasToCutOutTheEvent(vector<vector<int> > Pixelmatrix, vector<int> mirror,
   bool HasCloud, int CutOrder, vector<int> pT1,
                vector<int> pT2,  vector<int> pT3,  vector<int> pT4,  vector<int>
   pT5,  vector<int> pT6,
                vector<double> pcfT1, vector<double> pcfT2, vector<double> pcfT3,
   vector<double> pcfT4, vector<double> pcfT5, vector<double> pcfT6);
  vector<double> CreateBinBoundaryVec();
  // functions for the xmax-corrections
  double GetWeightedAverage(vector<double> values, vector<double> errors);
  double GetCorrectedXmax(double xmax, double ecal);
  void GetXmaxMoments(vector<double> xmaxvalues_input, vector<double>
   logenergyvalues_input, vector<double> binboundaries, vector<double> &mean,
   vector<double> &sigma, vector<double> &meanenergy, vector<int> &nevents);
  void GetXmaxMomentsUncertainties(vector<double> xmaxvalues_input, vector<double>
   logenergyvalues_input, vector<double> binboundaries, vector<double> &meanerror,
   vector<double> &sigmaerror);

  double CalculateX1(double lgE);
  double CalculateX2(double lgE);
  double CalculateLambda1(double lgE);
  double CalculateLambda2(double lgE);
  double CalculateK(double Lambda_eta, double z0, double lambda);
  double CalculateRisingEtaDepth(vector<double> xmaxValues, double eta);
  double CalculateFallingEtaDepth(vector<double> xmaxValues, double eta);
  double LambdaEtaLLH(double lambda_eta, double x0, double lambda, double etaDepth,
   bool rising, vector<double> xmaxValues);
  double FitLambdaEta(double x0, double lambda, double etaDepth, bool rising,
   vector<double> xmaxValues);
  double CalculateFDTotalResolutionICRC17(double lgE);
  double RoundFixedPrecision(double value);
  vector<double> GetMoments(vector<double> v);

  void FillEnergyXmaxVecMono(vector<double> &EHybrid_vec, vector<double>
   &LogEHybrid_vec, vector<double> &Xmax_vec, vector<string> &eventClass_vec, double
   EHybrid, double Xmax );
```



```
  vector<double> pushbackRecEnergyXmaxData(double EHybrid1, double EHybrid2, double
   EHybrid1err, double EHybrid2err, double Xmax1, double Xmax2, double Xmax1err,
   double Xmax2err,
                    double EHybrid3 = 0,  double EHybrid3err = 0, double Xmax3 =
  0, double Xmax3err = 0, double EHybrid4 = 0,  double EHybrid4err = 0, double
   Xmax4 = 0, double Xmax4err = 0);
  void FillEnergyXmaxVec(int stqevent, vector<double> RecData, vector<double>
   &energytemp, vector<double> &energyerrortemp, vector<double> &xmaxtemp,
   vector<double> &xmaxerrortemp, vector<double> &EHybrid_vec, vector<double>
   &LogEHybrid_vec, vector<double> &Xmax_vec, vector<string> &eventClass_vec);

  //Initialize variables
  vector<vector<int> > Pixelmatrix;
  vector<int> CutOutSites;
  bool HasLL , HasLM, HasLA, HasCO ;  // Los Leones, ...
  bool HasCloudLL, HasCloudLM, HasCloudLA, HasCloudCO;

  int gpssecondCO,gpssecondLA,gpssecondLL,gpssecondLM;

  double EHybridLL, EHybridLM, EHybridLA, EHybridCO;
  double EHybridLLError, EHybridLMError, EHybridLAError, EHybridCOError;
  double XmaxLL, XmaxLM, XmaxLA, XmaxCO;
  double XmaxLLError, XmaxLMError, XmaxLAError, XmaxCOError;

  vector<int> SDID_vec;
  vector<double> EHybrid_vec, LogEHybrid_vec, Xmax_vec ;
  vector<string> eventClass_vec;
  vector<vector<double> > Xmax_vec_vec ;
  vector<double> empty_vec;

  vector<int> * activemirrorLL = new vector<int>();
  vector<int> * activemirrorLM = new vector<int>();
  vector<int> * activemirrorLA = new vector<int>();
  vector<int> * activemirrorCO = new vector<int>();

  vector<int> *pixelTel1LL=new vector<int>();vector<int> *pixelTel2LL=new vector<int>();
  vector<int> *pixelTel3LL=new vector<int>();vector<int> *pixelTel4LL=new vector<int>();
  vector<int> *pixelTel5LL=new vector<int>();vector<int> *pixelTel6LL=new vector<int>();
  vector<int> *pixelTel1LM=new vector<int>();vector<int> *pixelTel2LM=new vector<int>();
  vector<int> *pixelTel3LM=new vector<int>();vector<int> *pixelTel4LM=new vector<int>();
  vector<int> *pixelTel5LM=new vector<int>();vector<int> *pixelTel6LM=new vector<int>();
  vector<int> *pixelTel1LA=new vector<int>();vector<int> *pixelTel2LA=new vector<int>();
  vector<int> *pixelTel3LA=new vector<int>();vector<int> *pixelTel4LA=new vector<int>();
  vector<int> *pixelTel5LA=new vector<int>();vector<int> *pixelTel6LA=new vector<int>();
  vector<int> *pixelTel1CO=new vector<int>();vector<int> *pixelTel2CO=new vector<int>();
  vector<int> *pixelTel3CO=new vector<int>();vector<int> *pixelTel4CO=new vector<int>();
  vector<int> *pixelTel5CO=new vector<int>();vector<int> *pixelTel6CO=new vector<int>();

  vector<double> *pixelcfTel1LL = new vector<double>();
  vector<double> *pixelcfTel2LL = new vector<double>();
  vector<double> *pixelcfTel3LL = new vector<double>();
  vector<double> *pixelcfTel4LL = new vector<double>();
  vector<double> *pixelcfTel5LL = new vector<double>();
  vector<double> *pixelcfTel6LL = new vector<double>();
  vector<double> *pixelcfTel1LM = new vector<double>();
  vector<double> *pixelcfTel2LM = new vector<double>();
  vector<double> *pixelcfTel3LM = new vector<double>();
  vector<double> *pixelcfTel4LM = new vector<double>();
  vector<double> *pixelcfTel5LM = new vector<double>();
  vector<double> *pixelcfTel6LM = new vector<double>();
  vector<double> *pixelcfTel1LA = new vector<double>();
  vector<double> *pixelcfTel2LA = new vector<double>();
  vector<double> *pixelcfTel3LA = new vector<double>();
  vector<double> *pixelcfTel4LA = new vector<double>();
  vector<double> *pixelcfTel5LA = new vector<double>();
  vector<double> *pixelcfTel6LA = new vector<double>();
  vector<double> *pixelcfTel1CO = new vector<double>();
  vector<double> *pixelcfTel2CO = new vector<double>();
  vector<double> *pixelcfTel3CO = new vector<double>();
```



```
  vector<double> *pixelcfTel4CO = new vector<double>();
  vector<double> *pixelcfTel5CO = new vector<double>();
  vector<double> *pixelcfTel6CO = new vector<double>();

  int counterTotal(0),SDiD;
  string ReadFile;

int main() {
  if (!ReadRootFile.compare("all"))         ReadFile = "./Root-Files/outputallcd.root";
  else if (!ReadRootFile.compare("selected"))   ReadFile =
   "./Root-Files/outputselectedcd.root";
  else if (!ReadRootFile.compare("removed"))    ReadFile =
   "./Root-Files/outputremovedcd.root";
  else {cout << "Ungueltige Eingabe (kein Root.File vorhanden)" << endl;
    return 0;
  }

  ostringstream oss;
  oss << ReadRootFile.c_str() << "_CL" << CloudLimitdefault <<  "_CO" << CutOrder ;
  string XmaxVecName = "XMaxVec_" + oss.str()    + ".txt" ;
  string XmaxResultsName = "XMaxRes_" + oss.str()    + ".txt";

  // Read the input-root File   (produced from analysis.cc)
  TFile *file = new TFile(ReadFile.c_str(), "READ");
  TTree *tree = (TTree*)file->Get("Tree");

  if (!file || file->IsZombie()) { //gives an error, If the root file can't be opened
    cout <<"----  The ROOT - file could not be opened! ---  "<< endl;;
    delete file;
  }

  // Set Branch Addresses to use these data as the input for the cloud data extraction
  tree->SetBranchAddress("SDID", &SDiD );
  tree->SetBranchAddress("HasLL", &HasLL);  tree->SetBranchAddress("HasLM", &HasLM);
  tree->SetBranchAddress("HasLA", &HasLA);  tree->SetBranchAddress("HasCO", &HasCO);
  tree->SetBranchAddress("gpssecondLL",&gpssecondLL);
  tree->SetBranchAddress("gpssecondLM",&gpssecondLM);
  tree->SetBranchAddress("gpssecondLA",&gpssecondLA);
  tree->SetBranchAddress("gpssecondCO",&gpssecondCO);
  tree->SetBranchAddress("gEHybridLL", &EHybridLL);
  tree->SetBranchAddress("gEHybridLM", &EHybridLM);
  tree->SetBranchAddress("gEHybridLA", &EHybridLA);
  tree->SetBranchAddress("gEHybridCO", &EHybridCO);
  tree->SetBranchAddress("gEHybridLLError", &EHybridLLError);
  tree->SetBranchAddress("gEHybridLMError", &EHybridLMError);
  tree->SetBranchAddress("gEHybridLAError", &EHybridLAError);
  tree->SetBranchAddress("gEHybridCOError", &EHybridCOError);
  tree->SetBranchAddress("gXmaxLL", &XmaxLL);
  tree->SetBranchAddress("gXmaxLM", &XmaxLM);
  tree->SetBranchAddress("gXmaxLA", &XmaxLA);
  tree->SetBranchAddress("gXmaxCO", &XmaxCO);
  tree->SetBranchAddress("gXmaxLLError", &XmaxLLError);
  tree->SetBranchAddress("gXmaxLMError", &XmaxLMError);
  tree->SetBranchAddress("gXmaxLAError", &XmaxLAError);
  tree->SetBranchAddress("gXmaxCOError", &XmaxCOError);
  tree->SetBranchAddress("activemirrorLL",&activemirrorLL);
  tree->SetBranchAddress("activemirrorLM",&activemirrorLM);
  tree->SetBranchAddress("activemirrorLA",&activemirrorLA);
  tree->SetBranchAddress("activemirrorCO",&activemirrorCO);
  tree->SetBranchAddress("pixelTel1LL",&pixelTel1LL);
  tree->SetBranchAddress("pixelTel2LL",&pixelTel2LL);
  tree->SetBranchAddress("pixelTel3LL",&pixelTel3LL);
  tree->SetBranchAddress("pixelTel4LL",&pixelTel4LL);
  tree->SetBranchAddress("pixelTel5LL",&pixelTel5LL);
  tree->SetBranchAddress("pixelTel6LL",&pixelTel6LL);
  tree->SetBranchAddress("pixelTel1LM",&pixelTel1LM);
  tree->SetBranchAddress("pixelTel2LM",&pixelTel2LM);
  tree->SetBranchAddress("pixelTel3LM",&pixelTel3LM);
  tree->SetBranchAddress("pixelTel4LM",&pixelTel4LM);
```

```
tree->SetBranchAddress("pixelTel5LM",&pixelTel5LM);
tree->SetBranchAddress("pixelTel6LM",&pixelTel6LM);
tree->SetBranchAddress("pixelTel1LA",&pixelTel1LA);
tree->SetBranchAddress("pixelTel2LA",&pixelTel2LA);
tree->SetBranchAddress("pixelTel3LA",&pixelTel3LA);
tree->SetBranchAddress("pixelTel4LA",&pixelTel4LA);
tree->SetBranchAddress("pixelTel5LA",&pixelTel5LA);
tree->SetBranchAddress("pixelTel6LA",&pixelTel6LA);
tree->SetBranchAddress("pixelTel1CO",&pixelTel1CO);
tree->SetBranchAddress("pixelTel2CO",&pixelTel2CO);
tree->SetBranchAddress("pixelTel3CO",&pixelTel3CO);
tree->SetBranchAddress("pixelTel4CO",&pixelTel4CO);
tree->SetBranchAddress("pixelTel5CO",&pixelTel5CO);
tree->SetBranchAddress("pixelTel6CO",&pixelTel6CO);
tree->SetBranchAddress("pixelcfTel1LL",&pixelcfTel1LL);
tree->SetBranchAddress("pixelcfTel2LL",&pixelcfTel2LL);
tree->SetBranchAddress("pixelcfTel3LL",&pixelcfTel3LL);
tree->SetBranchAddress("pixelcfTel4LL",&pixelcfTel4LL);
tree->SetBranchAddress("pixelcfTel5LL",&pixelcfTel5LL);
tree->SetBranchAddress("pixelcfTel6LL",&pixelcfTel6LL);
tree->SetBranchAddress("pixelcfTel1LM",&pixelcfTel1LM);
tree->SetBranchAddress("pixelcfTel2LM",&pixelcfTel2LM);
tree->SetBranchAddress("pixelcfTel3LM",&pixelcfTel3LM);
tree->SetBranchAddress("pixelcfTel4LM",&pixelcfTel4LM);
tree->SetBranchAddress("pixelcfTel5LM",&pixelcfTel5LM);
tree->SetBranchAddress("pixelcfTel6LM",&pixelcfTel6LM);
tree->SetBranchAddress("pixelcfTel1LA",&pixelcfTel1LA);
tree->SetBranchAddress("pixelcfTel2LA",&pixelcfTel2LA);
tree->SetBranchAddress("pixelcfTel3LA",&pixelcfTel3LA);
tree->SetBranchAddress("pixelcfTel4LA",&pixelcfTel4LA);
tree->SetBranchAddress("pixelcfTel5LA",&pixelcfTel5LA);
tree->SetBranchAddress("pixelcfTel6LA",&pixelcfTel6LA);
tree->SetBranchAddress("pixelcfTel1CO",&pixelcfTel1CO);
tree->SetBranchAddress("pixelcfTel2CO",&pixelcfTel2CO);
tree->SetBranchAddress("pixelcfTel3CO",&pixelcfTel3CO);
tree->SetBranchAddress("pixelcfTel4CO",&pixelcfTel4CO);
tree->SetBranchAddress("pixelcfTel5CO",&pixelcfTel5CO);
tree->SetBranchAddress("pixelcfTel6CO",&pixelcfTel6CO);

int gecutteteSites = 0;
int gecutteteEvents = 0;
int nentries = tree->GetEntries();

// Sort out all telescopes with a high cloudfraction
for(int i=0; i<nentries; ++i)  {// loop over all events
  HasCloudLL = false;    HasCloudLM = false;
  HasCloudLA = false;    HasCloudCO = false;
  tree->GetEntry(i);
  counterTotal++;
  cout << "   :     Event-SDiD: " << SDiD << "  ....  and " << gpssecondLL << " ,
  " << gpssecondLM << " , " << gpssecondLA << " , " << gpssecondCO ;

  // Cut out all Cloud Sites
  if (HasLL == 1 ){
    HasCloudLL = HasToCutOutTheEvent(Pixelmatrix, *activemirrorLL, &HasCloudLL,
  CutOrder, *pixelTel1LL, *pixelTel2LL, *pixelTel3LL, *pixelTel4LL, *pixelTel5LL,
  *pixelTel6LL,
            *pixelcfTel1LL, *pixelcfTel2LL, *pixelcfTel3LL,*pixelcfTel4LL,
  *pixelcfTel5LL, *pixelcfTel6LL);
    if (HasCloudLL == true) {CutOutSites.push_back(i);++gecutteteSites;}
  } else {HasCloudLL = true;}
  if (HasLM == 1 ){
    HasCloudLM = HasToCutOutTheEvent(Pixelmatrix, *activemirrorLM, &HasCloudLM,
  CutOrder, *pixelTel1LM, *pixelTel2LM, *pixelTel3LM, *pixelTel4LM, *pixelTel5LM,
  *pixelTel6LM,
            *pixelcfTel1LM, *pixelcfTel2LM, *pixelcfTel3LM,*pixelcfTel4LM,
  *pixelcfTel5LM, *pixelcfTel6LM);
    if (HasCloudLM == true) {CutOutSites.push_back(i);++gecutteteSites;}
  } else {HasCloudLM = true;}
```



```
  if (HasLA == 1 ){
    HasCloudLA = HasToCutOutTheEvent(Pixelmatrix, *activemirrorLA, &HasCloudLA,
  CutOrder, *pixelTel1LA, *pixelTel2LA, *pixelTel3LA, *pixelTel4LA, *pixelTel5LA,
  *pixelTel6LA,
            *pixelcfTel1LA, *pixelcfTel2LA, *pixelcfTel3LA,*pixelcfTel4LA,
  *pixelcfTel5LA, *pixelcfTel6LA);
    if (HasCloudLA == true) {CutOutSites.push_back(i);++gecutteteSites;}
  } else {HasCloudLA = true;}
  if (HasCO == 1 ){
    HasCloudCO = HasToCutOutTheEvent(Pixelmatrix, *activemirrorCO, &HasCloudCO,
  CutOrder, *pixelTel1CO, *pixelTel2CO, *pixelTel3CO, *pixelTel4CO, *pixelTel5CO,
  *pixelTel6CO,
            *pixelcfTel1CO, *pixelcfTel2CO, *pixelcfTel3CO,*pixelcfTel4CO,
  *pixelcfTel5CO, *pixelcfTel6CO);
    if (HasCloudCO == true) {CutOutSites.push_back(i);++gecutteteSites;}
  } else {HasCloudCO = true;}

   if (HasCloudLL && HasCloudLA && HasCloudLM && HasCloudCO)
  {++gecutteteEvents; continue;}

  // Calculate the XMax and fill them in vectors
  SDiD_vec.push_back(SDiD);
  vector<double> energytemp, energyerrortemp, xmaxtemp, xmaxerrortemp;
  // Monoevents
  if(!HasCloudLL && HasCloudLA && HasCloudLM &&
  HasCloudCO){FillEnergyXmaxVecMono(EHybrid_vec, LogEHybrid_vec,Xmax_vec,
  eventClass_vec, EHybridLL, XmaxLL );}
  else if(HasCloudLL && !HasCloudLA && HasCloudLM &&
  HasCloudCO){FillEnergyXmaxVecMono(EHybrid_vec, LogEHybrid_vec,Xmax_vec,
  eventClass_vec, EHybridLA, XmaxLA );}
  else if(HasCloudLL && HasCloudLA && !HasCloudLM &&
  HasCloudCO){FillEnergyXmaxVecMono(EHybrid_vec, LogEHybrid_vec,Xmax_vec,
  eventClass_vec, EHybridLM, XmaxLM );}
  else if(HasCloudLL && HasCloudLA && HasCloudLM &&
  !HasCloudCO){FillEnergyXmaxVecMono(EHybrid_vec, LogEHybrid_vec,Xmax_vec,
  eventClass_vec, EHybridCO, XmaxCO );}

  // Stereoevents
  else if(!HasCloudLL && !HasCloudLA && HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLL, EHybridLA,
  EHybridLLError, EHybridLAError,  XmaxLL,  XmaxLA, XmaxLLError, XmaxLAError);
    FillEnergyXmaxVec(2,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && HasCloudLA && !HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError);
    FillEnergyXmaxVec(2,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && HasCloudLA && !HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLL, EHybridCO,
  EHybridLLError, EHybridCOError,  XmaxLL,  XmaxCO, XmaxLLError, XmaxCOError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(HasCloudLL && !HasCloudLA && !HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
  EHybridCO, EHybridCOError, XmaxCO, XmaxCOError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && !HasCloudLA && HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridCO, EHybridLL,
  EHybridCOError, EHybridLLError,  XmaxCO,  XmaxLL, XmaxCOError, XmaxLLError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && !HasCloudLA && !HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  // Tripleevents
  else if(HasCloudLL && HasCloudLA && HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridCO,
  EHybridLMError, EHybridCOError,  XmaxLM,  XmaxCO, XmaxLMError, XmaxCOError);
    FillEnergyXmaxVec(2,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && HasCloudLA && HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLL, EHybridCO,
  EHybridLLError, EHybridCOError,  XmaxLL,  XmaxCO, XmaxLLError, XmaxCOError,
  EHybridCO, EHybridCOError, XmaxCO, XmaxCOError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(HasCloudLL && !HasCloudLA && HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  // Quadrupleevents
  else if(!HasCloudLL && !HasCloudLA && !HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
                          EHybridLA, EHybridLAError, XmaxLA, XmaxLAError,
  EHybridCO, EHybridCOError, XmaxCO, XmaxCOError);
    FillEnergyXmaxVec(4,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
}

for (int z=0; z < 18; ++z)
{Xmax_vec_vec.push_back(empty_vec);}

cout << "# SD event id  lgE   Xmax" << endl;
cout << "LogEHybrid_vec.size()  :"  << LogEHybrid_vec.size() << "
  SDID_vec.size()  :"  << SDID_vec.size() << endl;

// Define energy bins for the analysis
vector<double> binBoundaries = CreateBinBoundaryVec();

for(int i=0; i<SDID_vec.size(); i++) {
  LogEHybrid_vec.at(i) = RoundFixedPrecision(LogEHybrid_vec.at(i));
  Xmax_vec.at(i) = RoundFixedPrecision(Xmax_vec.at(i));
  string prefix;
  if(SDID_vec.at(i) < 10000000) prefix = "   ";
  else prefix = "  ";
  cout << prefix << SDID_vec.at(i) << " " << scientific << setprecision(4) <<
  TMath::Log10(EHybrid_vec.at(i)) << " " << Xmax_vec.at(i) << eventClass_vec.at(i)
  << endl;
  cout << prefix << SDID_vec.at(i) << " " << scientific << setprecision(4) <<
  LogEHybrid_vec.at(i) << " " << Xmax_vec.at(i) << endl;
  cout << "LogEHybrid_vec.size " << LogEHybrid_vec.size() <<  " und SDiD.size() "
  << SDID_vec.size() << endl;
```



```
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridCO,
  EHybridLMError, EHybridCOError,  XmaxLM,  XmaxCO, XmaxLMError, XmaxCOError);
    FillEnergyXmaxVec(2,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  // Tripleevents
  else if(HasCloudLL && !HasCloudLA && !HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridCO,
  EHybridLMError, EHybridCOError,  XmaxLM,  XmaxCO, XmaxLMError, XmaxCOError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && HasCloudLA && HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridCO, EHybridLL,
  EHybridCOError, EHybridLLError,  XmaxCO,  XmaxLL, XmaxCOError, XmaxLLError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  else if(!HasCloudLL && !HasCloudLA && !HasCloudLM && HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
  EHybridLA, EHybridLAError, XmaxLA, XmaxLAError);
    FillEnergyXmaxVec(3,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
  // Quadrupleevents
  else if(!HasCloudLL && !HasCloudLA && !HasCloudLM && !HasCloudCO){
    vector<double> RecData = pushbackRecEnergyXmaxData(EHybridLM, EHybridLL,
  EHybridLMError, EHybridLLError,  XmaxLM,  XmaxLL, XmaxLMError, XmaxLLError,
                          EHybridLA, EHybridLAError, XmaxLA, XmaxLAError,
  EHybridCO, EHybridCOError, XmaxCO, XmaxCOError);
    FillEnergyXmaxVec(4,RecData, energytemp, energyerrortemp, xmaxtemp,
  xmaxerrortemp, EHybrid_vec, LogEHybrid_vec, Xmax_vec, eventClass_vec);
  }
}

for (int z=0; z < 18; ++z)
{Xmax_vec_vec.push_back(empty_vec);}

cout << "# SD event id  lgE   Xmax" << endl;
cout << "LogEHybrid_vec.size()  :"  << LogEHybrid_vec.size() << "
  SDID_vec.size()  :"  << SDID_vec.size() << endl;

// Define energy bins for the analysis
vector<double> binBoundaries = CreateBinBoundaryVec();

for(int i=0; i<SDID_vec.size(); i++) {
  LogEHybrid_vec.at(i) = RoundFixedPrecision(LogEHybrid_vec.at(i));
  Xmax_vec.at(i) = RoundFixedPrecision(Xmax_vec.at(i));
  string prefix;
  if(SDID_vec.at(i) < 10000000) prefix = "   ";
  else prefix = "  ";
  cout << prefix << SDID_vec.at(i) << " " << scientific << setprecision(4) <<
  TMath::Log10(EHybrid_vec.at(i)) << " " << Xmax_vec.at(i) << eventClass_vec.at(i)
  << endl;
  cout << prefix << SDID_vec.at(i) << " " << scientific << setprecision(4) <<
  LogEHybrid_vec.at(i) << " " << Xmax_vec.at(i) << endl;
  cout << "LogEHybrid_vec.size " << LogEHybrid_vec.size() <<  " und SDiD.size() "
  << SDID_vec.size() << endl;
```



```
  // Sort the Eventenergies in the right histograms
  for (int z=0; z < binBoundaries.size(); ++z) {
    if (binBoundaries.at(z) <= LogEHybrid_vec.at(i) && LogEHybrid_vec.at(i) <
  binBoundaries.at(z+1)){
      Xmax_vec_vec.at(z).push_back(Xmax_vec.at(i));
      break;
    }
  }
}

vector<double> meanXmax, sigmaXmax, meanEnergy, meanXmaxError, sigmaXmaxError;
vector<int> nEvents;
cout << endl << "Calculating Xmax moments..." << endl;
GetXmaxMoments(Xmax_vec, LogEHybrid_vec, binBoundaries, meanXmax, sigmaXmax,
  meanEnergy, nEvents);
cout << endl << "Calculating uncertainties on Xmax moments..." << endl;
GetXmaxMomentsUncertainties(Xmax_vec, LogEHybrid_vec, binBoundaries,
  meanXmaxError, sigmaXmaxError);
cout << endl << "Writing out results..." << endl;
cout << "meanEnergy  nEvents  meanXmax  meanXmaxError  sigmaXmax  sigmaXmaxError"
  << endl << fixed;
string filename;
for(int l=0; l<meanEnergy.size(); l++) {
  cout << setprecision(4) << meanEnergy.at(l) << "  " << nEvents.at(l) << "  " <<
  setprecision(3) << meanXmax.at(l) << "  " << setprecision(2) <<
  meanXmaxError.at(l) << "  " << setprecision(4) << sigmaXmax.at(l) << "  " <<
  setprecision(2) << sigmaXmaxError.at(l) << endl;
}
cout << " cut Events "<< gecutteteEvents << "  of  "  << nentries << "  , or " <<
  SDID_vec.size() << " Events remain in the dataset" << endl ;

// Create the outputfiles
ofstream output_file1;
output_file1.open (XmaxResultsName.c_str());
for(int l=0; l<meanEnergy.size(); l++) {
  output_file1 << setprecision(8) << meanEnergy.at(l) << "  " << nEvents.at(l) <<
  "  " << setprecision(8) << meanXmax.at(l) << "  " << setprecision(7) <<
  meanXmaxError.at(l) << "  " << setprecision(8) << sigmaXmax.at(l) << "  " <<
  setprecision(5) << sigmaXmaxError.at(l) << endl;
}
output_file1.close();
ofstream output_file2;
output_file2.open (XmaxVecName.c_str());
for (int i=0;i < 18;++i) {
  for (int k = 0; k < Xmax_vec_vec.at(i).size();++k) {
    output_file2 << Xmax_vec_vec.at(i).at(k) << "  "  << i << endl;
  }
}
output_file2.close();
cout << "Process ended successfully" << endl;
return 0;
}

// ------------------------------------
// ------ Define the functions --------
// ------------------------------------

bool HasToCutOutTheEvent(vector<vector<int> > Pixelmatrix, vector<int> mirror, bool
  HasCloud, int CutOrder,
            vector<int> pT1,  vector<int> pT2,  vector<int> pT3,  vector<int> pT4,
  vector<int> pT5,  vector<int> pT6,
            vector<double> pcfT1, vector<double> pcfT2, vector<double> pcfT3,
  vector<double> pcfT4, vector<double> pcfT5, vector<double> pcfT6)
{
  HasCloud = false;
  for (int k=0; k< mirror.size(); ++k) {
    Pixelmatrix.clear();
    if (k==0) {Pixelmatrix = FindNeighborPixel(pT1); HasCloud =
  CloudCut(Pixelmatrix, pcfT1);   if (HasCloud == true) break;}  //Has Cloud ==
```

```
    true --> Skip Site
     if (k==1) {Pixelmatrix = FindNeighborPixel(pT2); HasCloud =
    CloudCut(Pixelmatrix, pcfT2);    if (HasCloud == true) break;}
     if (k==2) {Pixelmatrix = FindNeighborPixel(pT3); HasCloud =
    CloudCut(Pixelmatrix, pcfT3);    if (HasCloud == true) break;}
     if (k==3) {Pixelmatrix = FindNeighborPixel(pT4); HasCloud =
    CloudCut(Pixelmatrix, pcfT4);    if (HasCloud == true) break;}
     if (k==4) {Pixelmatrix = FindNeighborPixel(pT5); HasCloud =
    CloudCut(Pixelmatrix, pcfT5);    if (HasCloud == true) break;}
     if (k==5) {Pixelmatrix = FindNeighborPixel(pT6); HasCloud =
    CloudCut(Pixelmatrix, pcfT6);    if (HasCloud == true) break;}
  }
  return HasCloud;
}

void FillEnergyXmaxVecMono(vector<double> &EHybrid_vec, vector<double>
    &LogEHybrid_vec, vector<double> &Xmax_vec, vector<string> &eventClass_vec, double
    EHybrid, double Xmax )
{
      EHybrid_vec.push_back(EHybrid);
      LogEHybrid_vec.push_back(TMath::Log10(EHybrid));
      Xmax_vec.push_back(GetCorrectedXmax(Xmax, EHybrid));
      eventClass_vec.push_back("");
}

bool CloudCut(vector<vector<int> > Pixelmatrix, vector<double> cloudfraction, int
    defineCut, double CloudLimit)
{
  // if cloudcut gives an "yes" the event is cutted out
  if (defineCut == -1 || Pixelmatrix.at(0).at(0) == -100)
  {return false;}   // No CloudCut
  double averagecloudfraction = 0;
  double sum = 0;
  double npixel = 0;
  if (defineCut == 3) {
     for (int z=0; z < cloudfraction.size(); ++z) {
      sum += cloudfraction.at(z);
    }
    averagecloudfraction = sum / cloudfraction.size();
    npixel = cloudfraction.size();
  }
  if (defineCut == 0 || defineCut == 1 || defineCut == 2) {
    for (int l = 0; l <= defineCut; ++l) {
      for (int z=0; z < Pixelmatrix.at(l).size(); ++z) {
        sum += cloudfraction.at(Pixelmatrix.at(l).at(z)-1);
      }
      npixel += Pixelmatrix.at(l).size();
    }
    averagecloudfraction = sum / npixel;
  }
  //if (averagecloudfraction == 2) return true;
  //if (averagecloudfraction < 0.95 )   // normally ">"  , but if you only want to
    observe Events with a special cloudfraction then use !=
  if (averagecloudfraction > CloudLimit  )
  {
    return true;
  }
  else {return false;}
}


vector<double> pushbackRecEnergyXmaxData(double EHybrid1, double EHybrid2, double
    EHybrid1err, double EHybrid2err, double Xmax1, double Xmax2, double Xmax1err,
    double Xmax2err,
                 double EHybrid3, double EHybrid3err, double Xmax3, double
    Xmax3err, double EHybrid4,  double EHybrid4err, double Xmax4, double Xmax4err)
{
      vector<double> RecData;
      RecData.push_back(EHybrid1);
```



```
      RecData.push_back(EHybrid2);
      RecData.push_back(EHybrid1err);
      RecData.push_back(EHybrid2err);
      RecData.push_back(Xmax1);
      RecData.push_back(Xmax2);
      RecData.push_back(Xmax1err);
      RecData.push_back(Xmax2err);
      RecData.push_back(EHybrid3);
      RecData.push_back(EHybrid3err);
      RecData.push_back(Xmax3);
      RecData.push_back(Xmax3err);
      RecData.push_back(EHybrid4);
      RecData.push_back(EHybrid4err);
      RecData.push_back(Xmax4);
      RecData.push_back(Xmax4err);
      return RecData;
}

void FillEnergyXmaxVec(int stqevent, vector<double> RecData, vector<double>
    &energytemp, vector<double> &energyerrortemp, vector<double> &xmaxtemp,
    vector<double> &xmaxerrortemp, vector<double> &EHybrid_vec, vector<double>
    &LogEHybrid_vec, vector<double> &Xmax_vec, vector<string> &eventClass_vec)
{
      energytemp.push_back(RecData.at(0));
      energytemp.push_back(RecData.at(1));
      energyerrortemp.push_back(RecData.at(2));
      energyerrortemp.push_back(RecData.at(3));
      xmaxtemp.push_back(RecData.at(4));
      xmaxtemp.push_back(RecData.at(5));
      xmaxerrortemp.push_back(RecData.at(6));
      xmaxerrortemp.push_back(RecData.at(7));
      if (stqevent >= 3) {
    energytemp.push_back(RecData.at(8));
    energyerrortemp.push_back(RecData.at(9));
    xmaxtemp.push_back(RecData.at(10));
    xmaxerrortemp.push_back(RecData.at(11));
      }
      if (stqevent == 4) {
    energytemp.push_back(RecData.at(12));
    energyerrortemp.push_back(RecData.at(13));
    xmaxtemp.push_back(RecData.at(14));
    xmaxerrortemp.push_back(RecData.at(15));
      }
      EHybrid_vec.push_back(GetWeightedAverage(energytemp,energyerrortemp));
      LogEHybrid_vec.push_back( TMath::Log10( GetWeightedAverage(
    energytemp,energyerrortemp)));
      Xmax_vec.push_back(GetCorrectedXmax(
    GetWeightedAverage(xmaxtemp,xmaxerrortemp),
    GetWeightedAverage(energytemp,energyerrortemp)));
      eventClass_vec.push_back(" --- STEREO EVENT");
      energytemp.clear();
      energyerrortemp.clear();
      xmaxtemp.clear();
      xmaxerrortemp.clear();
}

vector<vector<int> > FindNeighborPixel(vector<int> pixel_vec, int order)
{
  vector<vector<int> > Pixelmatrix;
  vector<int> empty_vec ;
  vector<int> addpixel_vec;
  Pixelmatrix.push_back(empty_vec);
  Pixelmatrix.push_back(empty_vec);
  Pixelmatrix.push_back(empty_vec);
  Pixelmatrix.at(0) = pixel_vec;
  int k=0;
  while (k < order) {
    addpixel_vec.clear();
    for(int i=0; i < pixel_vec.size(); ++i) {
```



```
      if (pixel_vec.at(i) % 22 == 0 ) {  // oberer Rand
        addpixel_vec.push_back(pixel_vec.at(i)+22);
        addpixel_vec.push_back(pixel_vec.at(i)-22);
        addpixel_vec.push_back(pixel_vec.at(i)-1);
        if (pixel_vec.at(i) % 2 == 0 ) {
          addpixel_vec.push_back(pixel_vec.at(i)-23);
        }
        else {
          addpixel_vec.push_back(pixel_vec.at(i)+21);
        }
      }
      else if ((pixel_vec.at(i)-1) % 22 == 0 ) {   // unterer Rand
        addpixel_vec.push_back(pixel_vec.at(i)+22);
        addpixel_vec.push_back(pixel_vec.at(i)-22);
        addpixel_vec.push_back(pixel_vec.at(i)+1);
        if (pixel_vec.at(i) % 2 == 0 ) {
          addpixel_vec.push_back(pixel_vec.at(i)-21);
        }
        else {
          addpixel_vec.push_back(pixel_vec.at(i)+23);
        }
      }

      else if (pixel_vec.at(i) % 2 == 0) {  // if x is even
        addpixel_vec.push_back(pixel_vec.at(i)+1);
        addpixel_vec.push_back(pixel_vec.at(i)-1);
        addpixel_vec.push_back(pixel_vec.at(i)+22);
        addpixel_vec.push_back(pixel_vec.at(i)-22);
        addpixel_vec.push_back(pixel_vec.at(i)-21);
        addpixel_vec.push_back(pixel_vec.at(i)-23);
      }
      else if ((pixel_vec.at(i)+1) % 2 == 0) {  // if x is odd
        addpixel_vec.push_back(pixel_vec.at(i)+1);
        addpixel_vec.push_back(pixel_vec.at(i)-1);
        addpixel_vec.push_back(pixel_vec.at(i)+22);
        addpixel_vec.push_back(pixel_vec.at(i)-22);
        addpixel_vec.push_back(pixel_vec.at(i)+21);
        addpixel_vec.push_back(pixel_vec.at(i)+23);
      }
    }

    sort(addpixel_vec.begin(),addpixel_vec.end());
          // sortiere die Eintraege in dem Vektor
    addpixel_vec.erase( unique(addpixel_vec.begin(), addpixel_vec.end()),
    addpixel_vec.end());      // loesche alle doppelten Eintraege aus dem Vektor
    (vorher sortieren?!)
    int i = 0;
    while ( i < addpixel_vec.size()) {       // loesche alle Eintraege des Vektors
    unterhalb von 1 und oberhalb von 440
      if ((addpixel_vec.at(i) < 1) || (addpixel_vec.at(i) > 440)) {
        addpixel_vec.erase(addpixel_vec.begin() + i);
      }
      else if (find(pixel_vec.begin(), pixel_vec.end(), addpixel_vec.at(i)) !=
    pixel_vec.end() ) { // ueberprueft ob der i-te addpixel_vec Eintrag schon im
    pixel_vec steht
        addpixel_vec.erase(addpixel_vec.begin() + i);
      }
      else {++i;}
    }

    if (k == 0) {            // erstelle die Vektoren, die alle Pixel um die
    "Zeitpixel" in erster/ zweiter usw. Ordnung enthalten
      Pixelmatrix.at(k+1).clear();
      Pixelmatrix.at(k+1) = addpixel_vec;
    }
    else if (k == 1) {
      Pixelmatrix.at(k+1).clear();
      Pixelmatrix.at(k+1) = addpixel_vec;
    }
```



```
    pixel_vec.insert(pixel_vec.end(),addpixel_vec.begin(),addpixel_vec.end());    //
    Fuege beide Vektoren zusammen
    sort(pixel_vec.begin(),pixel_vec.end());                                      //
    sortiere die Eintraege in dem Vektor
    pixel_vec.erase(unique(pixel_vec.begin(),pixel_vec.end()),pixel_vec.end());   //
    loesche alle doppelten Eintraege aus dem Vektor (vorher sortieren?!)
    ++k;
  }
  return Pixelmatrix;
}


/*=======================================================
   Utility function to calculate weighted average of a
   vector of measurements
=======================================================*/

double GetWeightedAverage(vector<double> values, vector<double> errors)
{
  double sum = 0;
  double weightsum = 0;
  for(int i=0; i<values.size(); i++) {
    double weight = 1/(errors.at(i)*errors.at(i));
    sum += weight*values.at(i);
    weightsum += weight;
  }
  return sum/weightsum;
}

/*=======================================================
   Function to correct a given Xmax value for the
   reconstruction bias
=======================================================*/

double GetCorrectedXmax(double xmax, double ecal)
{
  double z18 = TMath::Log10(ecal) - 18.;
  double mu = -3.4 + 0.93 * z18;
  double lwCorrBias = 6.5 / (TMath::Exp((TMath::Log10(ecal)-18.23)/0.41) + 1.);
  return xmax - lwCorrBias - mu;
}

/*=======================================================
   Main function to determine the first and second moments
   of the Xmax distributions in given energy bins
=======================================================*/

void GetXmaxMoments(vector<double> xmaxvalues_input, vector<double>
    logenergyvalues_input, vector<double> binboundaries, vector<double> &mean,
    vector<double> &sigma, vector<double> &meanenergy, vector<int> &nevents)
{
  int nbins = binboundaries.size();
  vector<double> xmaxValues[nbins];
  double nUncorrected[nbins][3] = {0};
  double xmaxSum[nbins][3] = {0};
  double xmaxSum2[nbins][3] = {0};
  double energySum[nbins] = {0};
  for(int i=0; i<xmaxvalues_input.size(); i++) {
    for (int j=0; j<nbins; j++) {
      if (binboundaries.at(j) <= logenergyvalues_input.at(i) &&
    logenergyvalues_input.at(i) < binboundaries.at(j+1)) {
      xmaxValues[j].push_back(xmaxvalues_input.at(i));
      energySum[j] += logenergyvalues_input.at(i);
      const double x1 = CalculateX1(logenergyvalues_input.at(i));
      const double x2 = CalculateX2(logenergyvalues_input.at(i));
      if(xmaxvalues_input.at(i) < x1) {
        nUncorrected[j][0]++;
      }
      else if(x1 <= xmaxvalues_input.at(i) && xmaxvalues_input.at(i) < x2) {
```

```
      nUncorrected[j][1]++;
      xmaxSum[j][1] += xmaxvalues_input.at(i);
      xmaxSum2[j][1] += pow(xmaxvalues_input.at(i), 2);
    }
    else {
      nUncorrected[j][2]++;
    }
      }
    }
  }

  for(int j=0; j<nbins; j++) {
    if (xmaxValues[j].size() > 0) {
      sort(xmaxValues[j].begin(), xmaxValues[j].end());
      double meanLgE;
      if(nUncorrected[j][0]+nUncorrected[j][1]+nUncorrected[j][2] > 0) {
        meanLgE =  ↩
  energySum[j]/(nUncorrected[j][0]+nUncorrected[j][1]+nUncorrected[j][2]);
      }
      else {
        meanLgE = 0;
      }
      meanenergy.push_back(meanLgE);
      int totalEvents;
      totalEvents = nUncorrected[j][0] + nUncorrected[j][1] + nUncorrected[j][2];
      nevents.push_back(totalEvents);
      double x1 = CalculateX1(meanLgE);
      double x2 = CalculateX2(meanLgE);
      double lambda1 = CalculateLambda1(meanLgE);
      double lambda2 = CalculateLambda2(meanLgE);
      double n1 = nUncorrected[j][1];
      double mu1 = n1 ? xmaxSum[j][1] / n1 : 0;
      double vBiased = xmaxSum2[j][1]/n1 - mu1*mu1;
      double v1 = n1 > 1 ? vBiased * n1/(n1 - 1) : 0;
      double etaRisingDepth = CalculateRisingEtaDepth(xmaxValues[j], 0.15);
      double etaFallingDepth = CalculateFallingEtaDepth(xmaxValues[j], 0.2);
      double risingLambda_eta = FitLambdaEta(x1, lambda1, etaRisingDepth, true,  ↩
  xmaxValues[j]);
      double fallingLambda_eta = FitLambdaEta(x2, lambda2, etaFallingDepth, false,  ↩
  xmaxValues[j]);
      double etaRisingIntegral = 0;

      for (int k=0; k<xmaxValues[j].size(); k++) {
    if (x1 < xmaxValues[j].at(k) && xmaxValues[j].at(k) < etaRisingDepth) {
      etaRisingIntegral++;
    }
      }
      double etaFallingIntegral = 0;
      for (int k=0; k<xmaxValues[j].size(); k++) {
    if (etaFallingDepth < xmaxValues[j].at(k) && xmaxValues[j].at(k) < x2) {
      etaFallingIntegral++;
    }
      }

      double mu0 = x1 - risingLambda_eta;
      double mu2 = x2 + fallingLambda_eta;
      double z0;
      z0 = fabs(x1-etaRisingDepth);
      double risingInt_eta_z0 = -risingLambda_eta*(exp(-z0/risingLambda_eta) - 1);
      double risingNorm = etaRisingIntegral/risingInt_eta_z0;
      double risingInt_z0_inf = risingLambda_eta*exp(-z0/risingLambda_eta);
      // n2
      z0 = fabs(x2 - etaFallingDepth);
      double fallingInt_eta_z0 = -fallingLambda_eta*(exp(-z0/fallingLambda_eta) - 1);
      double fallingNorm = etaFallingIntegral/fallingInt_eta_z0;
      double fallingInt_z0_inf = fallingLambda_eta*exp(-z0/fallingLambda_eta);
      double n0 = nUncorrected[j][0] ? nUncorrected[j][0]/(lambda1/(lambda1 +  ↩
  risingLambda_eta)) : 0;
      double n2 = nUncorrected[j][2] ? nUncorrected[j][2]/(lambda2/(lambda2 +  ↩
```



```
  fallingLambda_eta)) : 0;
      double v0 = pow(risingLambda_eta, 2);
      double v2 = pow(fallingLambda_eta, 2);
      double nTot = n0 + n1 + n2;
      double muTot = nTot ? (mu0*n0 + mu1*n1 + mu2*n2)/nTot : 0;
      double varTot = nTot ? (v0*n0 + v1*n1 + v2*n2) / nTot + (n0*(pow(mu0 - muTot,  ↩
  2)) + n1*(pow(mu1 - muTot, 2)) + n2*(pow(mu2 - muTot, 2)))/nTot : 0;
      double meanXmax = muTot;
      double resCorrection2 = pow(CalculateFDTotalResolutionICRC17(meanLgE),2);
      double sigmaXmax = sqrt(varTot - resCorrection2);
      mean.push_back(meanXmax);
      sigma.push_back(sigmaXmax);
    }
  }
}

/*=========================================================
   Functions to calculate X1, X2, Lambda1 and Lambda2
   which together define the acceptance
=========================================================*/

double CalculateX1(double lgE)
{
  double x1Pars[] = {594.8462226,    50.54349205,  -60.94689503};
  double x = lgE - 18.0;
  double result = x1Pars[0] + x1Pars[1]*x + x1Pars[2]*x*x;
  return result;
}

double CalculateX2(double lgE)
{
  double x2Pars[] = {8.83871678e+02,   1.83673105e+01,  -1.55385654e-01};
  double x = lgE - 18.0;
  double result;
  result = x2Pars[0] + x2Pars[1]*x + x2Pars[2]*x*x;
  return result;
}

double CalculateLambda1(double lgE)
{
  double l1Pars[] = {145.73893611,  245.13007698,  -15.20000551};
  double x = lgE - 18.0;
  double result = l1Pars[0] + l1Pars[1]*x + l1Pars[2]*x*x;
  return result;
}

double CalculateLambda2(double lgE)
{
  double l2Pars[] = {103.97282671,   59.99931423,    0.81381344};
  double x = lgE - 18.0;
  double result = l2Pars[0] + l2Pars[1]*x + l2Pars[2]*x*x;
  return result;
}

/*=========================================================
   Functions for the LambdaEta correction
=========================================================*/

double CalculateK(double Lambda_eta, double z0, double lambda)
{
  return Lambda_eta*(1 + exp(-z0/Lambda_eta)*(lambda/(lambda + Lambda_eta) - 1));
}

double CalculateRisingEtaDepth(vector<double> xmaxValues, double eta)
{
  sort(xmaxValues.begin(), xmaxValues.end());
  int etaValue = (eta)*xmaxValues.size();
  return xmaxValues.at(etaValue);
}
```



```


double CalculateFallingEtaDepth(vector<double> xmaxValues, double eta)
{
  sort(xmaxValues.begin(), xmaxValues.end());
  int etaValue = (1 - eta)*xmaxValues.size();
  return xmaxValues.at(etaValue);
}

// loglikelihood function
double LambdaEtaLLH(double lambda_eta, double x0, double lambda, double etaDepth,  ↩
    bool rising, vector<double> xmaxValues)
{
  double llh = 0;
  int n = 0;
  sort(xmaxValues.begin(), xmaxValues.end());
  for (int iDepth = 0; iDepth < xmaxValues.size(); iDepth++) {
    if (rising) {
      if (xmaxValues.at(iDepth) < etaDepth) {
    llh += (etaDepth - xmaxValues.at(iDepth))/lambda_eta;
    n++;
      }
    }
    else {
      if (xmaxValues.at(iDepth) > etaDepth) {
    llh += (xmaxValues.at(iDepth) - etaDepth)/lambda_eta;
    n++;
      }
    }
  }

  double z0;
  if (rising) z0 = etaDepth - x0;
  else z0 = x0 - etaDepth;
  const double norm = lambda_eta*(1 +  ↩
    exp(-z0/lambda_eta)*(lambda/(lambda+lambda_eta) - 1));
  const double logNorm = log(norm);
  llh += n*logNorm;
  return llh;
}

double FitLambdaEta(double x0, double lambda, double etaDepth, bool rising,  ↩
    vector<double> xmaxValues)
{
  double maxLE = 99;
  double minLE = 1;
  double deltaLE = 0.1;

  double bestLLH = INFINITY;
  double bestLE = INFINITY;

  for (double iLambda_eta = minLE; iLambda_eta < maxLE; iLambda_eta += deltaLE)
  {
    double llh = LambdaEtaLLH(iLambda_eta, x0, lambda, etaDepth, rising, xmaxValues);

    if (llh < bestLLH)
    {
      bestLE = iLambda_eta;
      bestLLH = llh;
    }
  }

  return bestLE;
}

/*=========================================================
   Function to calculate the FD resolution
=========================================================*/

double CalculateFDTotalResolutionICRC17(double lgE)
```



```
{
  double par[3] = {14.78496075,   3.40816591, -19.60465435};
  double resDet = par[0] + par[1] * pow((lgE + par[2]), 2);
  return resDet;
}

/*=========================================================
   Utility function to round doubles to fixed precision
   in scientific notation (4 digits after decimal point)
=========================================================*/

double RoundFixedPrecision(double value)
{
  int exponent = floor(TMath::Log10(value));

  if(exponent == 0) return round(value*10000.)/10000.;
  if(exponent == 1) return round(value*1000.)/1000.;
  if(exponent == 2) return round(value*100.)/100.;
  if(exponent == 3) return round(value*10.)/10.;
  if(exponent == 4) return round(value*1.)/1.;
}

/*=========================================================

   Main function to determine the first and second moments
   of the Imax distributions in given energy bins using
   the bootstrapping method

=========================================================*/

void GetXmaxMomentsUncertainties(vector<double> xmaxvalues_input, vector<double>  ↩
    logenergyvalues_input, vector<double> binboundaries, vector<double> &meanerror,  ↩
    vector<double> &sigmaerror)
{
  int nbins = binboundaries.size();
  vector<double> xmaxValues[nbins];
  vector<double> logenergyValues[nbins];

  for(int i=0; i<xmaxvalues_input.size(); i++)
  {
    for (int j=0; j<nbins; j++)
    {
      if (binboundaries.at(j) <= logenergyvalues_input.at(i) &&  ↩
    logenergyvalues_input.at(i) < binboundaries.at(j+1))
      {
    xmaxValues[j].push_back(xmaxvalues_input.at(i));
    logenergyValues[j].push_back(logenergyvalues_input.at(i));
      }
    }
  }

  vector<double> binboundaries_reduced, meanXmax_mock, sigmaXmax_mock;
  vector<double>xmaxvalues_mock, energyvalues_mock;
  int irandom;
  int nmock = nwuerfe;

  TRandom3 r(100);

  for(int j=0; j<nbins-1; j++)
  {
    binboundaries_reduced.push_back(binboundaries.at(j));
    binboundaries_reduced.push_back(binboundaries.at(j+1));

    cout << "  Processing bin " << binboundaries_reduced.at(0) << " to " <<  ↩
    binboundaries_reduced.at(1) << endl;

    if (xmaxValues[j].size() == 0) continue;

    for(int k=1; k<=nmock; k++)
```

```
    {
      if(k%10 == 0) cout << "    Generating mock dataset " << k << " of " << nmock
    << endl;
      for(int l=0; l<xmaxValues[j].size(); l++)
      {
        irandom = r.Integer(xmaxValues[j].size());
        xmaxvalues_mock.push_back(xmaxValues[j].at(irandom));
        energyvalues_mock.push_back(logenergyValues[j].at(irandom));
      }

      vector<double> meanXmax, sigmaXmax, meanEnergy;
      vector<int> nEvents;

      GetXmaxMoments(xmaxvalues_mock, energyvalues_mock, binboundaries_reduced,
    meanXmax, sigmaXmax, meanEnergy, nEvents);

      meanXmax_mock.push_back(meanXmax.at(0));
      sigmaXmax_mock.push_back(sigmaXmax.at(0));

      xmaxvalues_mock.clear();
      energyvalues_mock.clear();
    }

    vector<double> momentsMean = GetMoments(meanXmax_mock);
    meanerror.push_back(momentsMean[1]);
    vector<double> momentsSigma = GetMoments(sigmaXmax_mock);
    sigmaerror.push_back(momentsSigma[1]);

    meanXmax_mock.clear();
    sigmaXmax_mock.clear();
    binboundaries_reduced.clear();
  }
}

/*============================================================
   Utility function to calculate first and second moment
   of a vector
============================================================*/
vector<double> GetMoments(vector<double> v)
{
  double mean;
  double stdev;
  vector<double> Moments;
  double sum = accumulate(v.begin(), v.end(), 0.0);
  mean = sum / v.size();
  vector<double> diff(v.size());
  transform(v.begin(), v.end(), diff.begin(), bind2nd(minus<double>(), mean));
  double sq_sum = inner_product(diff.begin(), diff.end(), diff.begin(), 0.0);
  stdev = sqrt(sq_sum / v.size());
  Moments.push_back(mean);
  Moments.push_back(stdev);
  return Moments;
}

vector<double> CreateBinBoundaryVec()
{
  vector<double> binBoundaries;
  binBoundaries.push_back(17.8);  binBoundaries.push_back(17.9);
  binBoundaries.push_back(18.0);  binBoundaries.push_back(18.1);
  binBoundaries.push_back(18.2);  binBoundaries.push_back(18.3);
  binBoundaries.push_back(18.4);  binBoundaries.push_back(18.5);
  binBoundaries.push_back(18.6);  binBoundaries.push_back(18.7);
  binBoundaries.push_back(18.8);  binBoundaries.push_back(18.9);
  binBoundaries.push_back(19.0);  binBoundaries.push_back(19.1);
  binBoundaries.push_back(19.2);  binBoundaries.push_back(19.3);
  binBoundaries.push_back(19.4);  binBoundaries.push_back(19.5);
  binBoundaries.push_back(20.2);  return binBoundaries;
}
```



Dies ist der Code mit dem verschiedene Datensätze miteinander verglichen werden können. Als Input-Dateien dienen die Output Dateien der $X_{\max}$-Analyse.

```
#include <vector>
#include <string>
#include <iostream>
#include <math.h>
#include <sstream>
#include <fstream>

#include <TFile.h>
#include <TMath.h>
#include <TTree.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TROOT.h>
#include <TLatex.h>
#include <TH1D.h>
#include <TGraphErrors.h>
#include <TLegend.h>

using namespace std;

// Defined functions
vector<vector<double> > Transpose2x2Vec(vector<vector<double> > VecStart);
vector<vector<double> > ReadFileInXmaxVec(string filepathname, bool transpose);
void make18plotsinoneCanvas (vector<vector<double> > Xmax1, string titleXmax1,
    string outputfilefilename);
void compare2XmaxVecsIn18Hists (vector<vector<double> > Xmax1, vector<vector<double>
    > Xmax2, string titleXmax1, string titleXmax2, string outputfilefilename);
void compare4ResVecs(vector<vector<double> > Results1,vector<vector<double> >
    Result2,vector<vector<double> > Result3,vector<vector<double> > Result4,string
    legentry1,string legentry2,string legentry3, string legentry4, string
    outputfilefilename);
void compare2ResVecs(vector<vector<double> > Results1,vector<vector<double> >
    Result2,string legentry1, string legentry2, string outputfilefilename);

TH1D* makeHist(vector<double> vec, bool autorange=1, double xmin=0., double xmax=1.,
    string histname="");
void graphLayout(TGraph* graph,int color,int marker=1,float markersize=1.0,int
    linestyle=0,float linesize=1.0);
vector<double> CreateBinBoundaryVec();
void EmptyResultVec (vector<vector<double> > &Results1, double minnumber );

// Defined variables
vector<vector<double> > Xmax1, Xmax2, Xmax3, Xmax4;

vector<vector<double> > Results1, Results2, Results3, Results4;

// parameters to determine the Files and characterize the datasets
string ReadFile1 = "all";
string cloudlimit1 = "0";
string CutOrder1 = "3";

string ReadFile2 = "all";
string cloudlimit2 = ">50";
string CutOrder2 = "3";

string ReadFile3 = "all";
string cloudlimit3 = "0";
string CutOrder3 = "1";

string ReadFile4 = "all";
string cloudlimit4 = "0";
string CutOrder4 = "0";

//------------------------------------------------------------
// ----------------------MAIN function ---------------------
//------------------------------------------------------------
```



```

int main()
{
  double minnumber = 3;

  string characterize1 = ReadFile1 +  "_CL" + cloudlimit1.c_str() + "_CO" +
    CutOrder1.c_str();
  string filepathname1a = "./Xmax_Vec_Vecs2/XMaxVec_" + characterize1 + ".txt" ;
  string filepathname1b = "./Resultsmatrix2/XMaxRes_" + characterize1 + ".txt" ;
  string characterize2 = ReadFile2 +  "_CL" + cloudlimit2.c_str() + "_CO" +
    CutOrder2.c_str();
  string filepathname2a = "./Xmax_Vec_Vecs2/XMaxVec_" + characterize2 + ".txt" ;
  string filepathname2b = "./Resultsmatrix2/XMaxRes_" + characterize2 + ".txt" ;
  string characterize3 = ReadFile3 +  "_CL" + cloudlimit3.c_str() + "_CO" +
    CutOrder3.c_str();
  string filepathname3a = "./Xmax_Vec_Vecs2/XMaxVec_" + characterize3 + ".txt" ;
  string filepathname3b = "./Resultsmatrix2/XMaxRes_" + characterize3 + ".txt" ;
  string characterize4 = ReadFile4 +  "_CL" + cloudlimit4.c_str() + "_CO" +
    CutOrder4.c_str();
  string filepathname4a = "./Xmax_Vec_Vecs2/XMaxVec_" + characterize4 + ".txt" ;
  string filepathname4b = "./Resultsmatrix2/XMaxRes_" + characterize4 + ".txt" ;

  Xmax1 = ReadFileInXmaxVec(filepathname1a.c_str(),1);
  Xmax2 = ReadFileInXmaxVec(filepathname2a.c_str(),1);
  Xmax3 = ReadFileInXmaxVec(filepathname3a.c_str(),1);
  Xmax4 = ReadFileInXmaxVec(filepathname4a.c_str(),1);

  // graphical presentation of the zmax data for each event in the dataset
  make18plotsinoneCanvas(Xmax1, characterize1.c_str(), characterize1.c_str() );
  compare2XmaxVecsIn18Hists(Xmax1, Xmax2, characterize1.c_str(),
    characterize2.c_str(), "noclvslesscl");

  Results1 = ReadFileInXmaxVec(filepathname1b.c_str(),0);
  Results2 = ReadFileInXmaxVec(filepathname2b.c_str(),0);
  Results3 = ReadFileInXmaxVec(filepathname3b.c_str(),0);
  Results4 = ReadFileInXmaxVec(filepathname4b.c_str(),0);

  // delete those energybins, which has less than minnumber events in it, because
    the statistic is too small
  EmptyResultVec(Results1, minnumber);  EmptyResultVec(Results2, minnumber);
  EmptyResultVec(Results3, minnumber);  EmptyResultVec(Results4, minnumber);

  // Compare two or four results vecs with each other, graphical presentation of the
    results for the zmax moments
  compare2ResVecs(Results1, Results2, characterize1.c_str(), characterize2.c_str(),
    "noclvslesscl");
  compare4ResVecs(Results1, Results2, Results3, Results4, characterize1.c_str(),
    characterize2.c_str(),characterize3.c_str(),characterize4.c_str(),
    "CompareCutOrder");
}

// ---------------------  DEFINITION OF FUNCTIONS ---------------- //

vector<vector<double> > ReadFileInXmaxVec(string filepathname, bool transpose)
{
  vector<vector<double> > ReadInMatrix;
  ifstream file(filepathname.c_str()); // oeffnet diese Datei zum Lesen
  if( !file.is_open() ){
        cerr << "Fehler beim Oeffnen der Datei  ("  << filepathname << " )" << endl;
    }
  vector<vector<double> > Xmaxtemp;
  while (file) {
    string line;
    getline(file, line);
    istringstream iss(line);
    vector<double> values;
    copy(istream_iterator<double>(iss), istream_iterator<double>(),
    back_insert_iterator<vector<double> >(values));
    if (values.size() > 0) Xmaxtemp.push_back(values);
```



```
  }
  file.close();
  if (transpose == true) ReadInMatrix = Transpose2x2Vec(Xmaxtemp);
  else ReadInMatrix = Xmaxtemp;
  return ReadInMatrix;
}

vector<vector<double> > Transpose2x2Vec(vector<vector<double> > VecStart)
{
  vector<double> empty_vec;
  vector<vector<double> > Xmax;
  for (int z=0; z < 18; ++z)
  {Xmax.push_back(empty_vec);}
  for (int k=0;k<VecStart.size();++k){
    for (int l=0; l<18;++l){
      if (VecStart.at(k).at(1) == l) {
    Xmax.at(l).push_back(VecStart.at(k).at(0));
    break;
      }
    }
  }
  return Xmax;
}

void make18plotsinoneCanvas (vector<vector<double> > Xmax1, string titleXmax1,
    string outputfilefilename)
{
  TCanvas *c1 = new TCanvas(outputfilefilename.c_str(),"Daten_Auslese", 200, 100,
    500, 768);
  c1->Divide(3,6);
  for (int i=0;i < 18;++i)
  {
    c1->cd(i+1);
    gPad->SetBottomMargin(0.16);  gPad->SetLeftMargin(0.18);
    gStyle->SetStatH(0.37);  gStyle->SetStatW(0.33);
    gStyle->SetTitleFontSize(0.06);  gStyle->SetTitleFontSize(0.06);
    gStyle->SetTitleSize(0.065,"x");  gStyle->SetTitleSize(0.065,"y");
    gStyle->SetLabelSize(0.065, "X");  gStyle->SetLabelSize(0.065, "Y");
    gStyle->SetOptStat("emr");
    ostringstream oss1;
    oss1 << (17.8 + 0.1*i) << "< lg(E/eV) <" << (17.8 + 0.1*(i+1)) ;

    TH1D* hist_xMax1 = makeHist(Xmax1.at(i), 0, 400, 1100, "" );
    string histname = oss1.str();
    hist_xMax1->SetTitle(histname.c_str());
    hist_xMax1->GetXaxis()->SetTitle("X_{max} in [g/cm^{2}]");
    hist_xMax1->GetYaxis()->SetTitle("Entries");
    hist_xMax1->SetLineWidth(1.5);
    hist_xMax1->Draw();
  }

  ostringstream oss;
  oss << "File18" << outputfilefilename.c_str() ;
  string filename;
  filename = oss.str() + ".pdf";
  c1->SetLogy(0);
  c1->Print(filename.c_str());
  c1->Clear();
}


void compare2XmaxVecsIn18Hists (vector<vector<double> > Xmax1, vector<vector<double>
    > Xmax2, string titleXmax1, string titleXmax2, string outputfilefilename)
{
  TCanvas *c1 = new TCanvas("c1","Daten_Auslese", 200, 100, 1024, 768);
  TCanvas *c2 = new TCanvas("c2","Daten_Auslese", 200, 100, 1024, 768);
  int gesamtentries1 = 0; int gesamtentries2 = 0;
  int histentries1 = 0; int histentries2 = 0;
```

```
  for (int i=0;i < 18;++i)
  {
    // if (Xmax1.at(i).size() == 0) continue;
    c1->Divide(2,1);    // divide the canvas into two columns
    c1->cd(1)->Divide(1,2);  // divide the right columns into two rows
    c1->cd(1)->cd(1);
    gPad->SetBottomMargin(0.16); gPad->SetLeftMargin(0.18);
    gStyle->SetStatH(0.3); gStyle->SetStatW(0.35);
    gStyle->SetTitleFontSize(0.06);  gStyle->SetTitleFontSize(0.06);
    gStyle->SetTitleSize(0.06,"x");  gStyle->SetTitleSize(0.06,"y");
    gStyle->SetLabelSize(0.055, "X");gStyle->SetLabelSize(0.055, "Y");
    ostringstream oss1;
    oss1 << (17.8 + 0.1*i) << "< lg(E/eV) <" << (17.8 + 0.1*(i+1)) ;

    TH1D* hist_xMax1 = makeHist(Xmax1.at(i), 0, 400, 1100, oss1.str() );
    hist_xMax1->SetTitle(titleXmax1.c_str());
    hist_xMax1->GetXaxis()->SetTitle("X_{max} in [g/cm^{2}]");
    hist_xMax1->GetYaxis()->SetTitle("Entries");
    hist_xMax1->SetLineWidth(1.5);
    hist_xMax1->Draw();

    c1->cd(1)->cd(2);
    gPad->SetBottomMargin(0.16); gPad->SetLeftMargin(0.18);
    TH1D* hist_xMax2 = makeHist(Xmax2.at(i), 0, 400, 1100, oss1.str() );
    hist_xMax2->SetTitle(titleXmax2.c_str());
    hist_xMax2->GetXaxis()->SetTitle("X_{max} in [g/cm^{2}]");
    hist_xMax2->GetYaxis()->SetTitle("Entries");
    hist_xMax2->SetLineWidth(1.5);
    hist_xMax2->Draw();
    hist_xMax1->SetLineColor(4); hist_xMax2->SetLineColor(2);

    c1->cd(2);
    TH1D* hist_xMaxCOMP1 = makeHist(Xmax1.at(i), 0, 400, 1100);
    TH1D* hist_xMaxCOMP2 = makeHist(Xmax2.at(i), 0, 400, 1100);
    gPad->SetLeftMargin(0.18); gPad->SetBottomMargin(0.16);
    hist_xMaxCOMP1->SetStats(0); hist_xMaxCOMP2->SetStats(0);
    hist_xMaxCOMP1->SetLineColor(4); hist_xMaxCOMP2->SetLineColor(2);

    double inthist1 = 1/hist_xMaxCOMP1->Integral();
    double inthist2 = 1/hist_xMaxCOMP2->Integral();
    hist_xMaxCOMP1->Scale(inthist1); hist_xMaxCOMP2->Scale(inthist2);

    hist_xMaxCOMP1->SetTitle("comparison");
    hist_xMaxCOMP1->GetXaxis()->SetTitle("X_{max} in [g/cm^{2}]");
    hist_xMaxCOMP1->GetYaxis()->SetTitle("Entries (normalized)");
    hist_xMaxCOMP1->GetYaxis()->SetTitleOffset(1.5);
    hist_xMaxCOMP1->SetLineWidth(1.5); hist_xMaxCOMP2->SetLineWidth(1.5);
    hist_xMaxCOMP1->Draw();
    hist_xMaxCOMP2->Draw("same");

    histentries1 = hist_xMax1->GetEntries();
    gesamtentries1 = gesamtentries1 + histentries1;
    histentries2 = hist_xMax2->GetEntries();
    gesamtentries2 = gesamtentries2 + histentries2;

    ostringstream oss;
    oss << "Bin" << (17.8 + 0.1*i)*10 << outputfilefilename.c_str() ;
    string filename;
    filename = "XMax" + oss.str()   + ".pdf";
    c1->SetLogy(0);
    c1->Print(filename.c_str());
    c1->Clear();
  }

  cout << "gesamtentries  " << gesamtentries1 << " and " << gesamtentries2 << endl;

}

void compare2ResVecs(vector<vector<double> > Results1,vector<vector<double> >  ↩
```



```
    Result2,string legentry1, string legentry2, string outputfilefilename)
{
  string File1 = "XmaxMoment1" + outputfilefilename   + ".pdf" ;
  string File2 = "XmaxMoment2" + outputfilefilename   + ".pdf";
  string File3 = "XmaxMoment1" + outputfilefilename   + ".root" ;
  string File4 = "XmaxMoment2" + outputfilefilename   + ".root" ;

  TCanvas *c1 = new TCanvas("c1","Ergebnisvergleich", 200, 100, 1024, 768);
  TGraphErrors *g1 = new TGraphErrors(); TGraphErrors *g2 = new TGraphErrors();
  TGraphErrors *g3 = new TGraphErrors(); TGraphErrors *g4 = new TGraphErrors();
  TGraphErrors *g5 = new TGraphErrors(); TGraphErrors *g6 = new TGraphErrors();


  TLegend *leg = new TLegend(0.12,0.95,0.35,0.72, "");   //Position in % des Canvas,  ↩
    x oben links , y- oben links , z- untern rechts, y-unten rechts
  leg->AddEntry(g1,legentry1.c_str(),"p");  leg->AddEntry(g2,legentry2.c_str(),"p");
  TLegend *leg2 = new TLegend(0.12,0.2,0.34,0.45, "");   //Position in % des Canvas,  ↩
    x oben links , y- oben links , z- untern rechts, y-unten rechts
  leg2->AddEntry(g4,legentry1.c_str(),"p");  leg2->AddEntry(g5,legentry2.c_str(),"p");

  c1->Divide(1,2);
  gStyle->SetTitleSize(0.06,"x");  gStyle->SetTitleSize(0.06,"y");
  gStyle->SetLabelSize(0.06, "X"); gStyle->SetLabelSize(0.06, "Y");
  gStyle->SetTitleFontSize(0.06);  gStyle->SetTitleFontSize(0.06);


  for (int i=0; i<Results1.size(); ++i){
    if (Results1.at(i).size() != 0) {
      if (Results1.at(i).at(2) !=-1 || Results1.at(i).at(2) != Results1.at(i).at(2))  ↩
    g1->SetPoint(i,Results1.at(i).at(0),Results1.at(i).at(2));   // 1.Wert teilt dem  ↩
    Datenpaar eine Nummer zu, 2. Wert ist der x-Wert, dritter der y-Wert
      if (Results1.at(i).at(3) !=-1 || Results1.at(i).at(3) != Results1.at(i).at(3))  ↩
    g1->SetPointError (i, 0, Results1.at(i).at(3));
    }
    if (Results2.at(i).size() != 0) {
      if (Results2.at(i).at(2) !=-1 || Results2.at(i).at(2) != Results2.at(i).at(2))  ↩
    g2->SetPoint(i,Results2.at(i).at(0),Results2.at(i).at(2));
      if (Results2.at(i).at(3) !=-1 || Results2.at(i).at(3) != Results2.at(i).at(3))  ↩
    g2->SetPointError (i, 0, Results2.at(i).at(3));
    }
    if (Results1.at(i).size() == 0 || Results2.at(i).size() == 0) continue;
    if (Results1.at(i).at(2) !=-1 && Results2.at(i).at(2) !=-1 ||  ↩
    Results1.at(i).at(2) != Results1.at(i).at(2) || Results2.at(i).at(2) !=  ↩
    Results2.at(i).at(2)) {
      g3->SetPoint(i,(Results1.at(i).at(0) + Results2.at(i).at(0))/2,  ↩
    Results2.at(i).at(2) - Results1.at(i).at(2));
      if (Results1.at(i).at(3) !=-1 && Results2.at(i).at(3) !=-1 ||  ↩
    Results1.at(i).at(3) != Results1.at(i).at(3) || Results2.at(i).at(3) !=  ↩
    Results2.at(i).at(3))
        g3->SetPointError (i, 0, Results2.at(i).at(3) + Results1.at(i).at(3));
    }
  }

  graphLayout(g1,4,2,2);
  g1->SetTitle("Energie/X_{max}- Analyse");
  g1->GetXaxis()->SetTitle("lg(E/eV)");
  g1->GetYaxis()->SetTitle("<Xmax> in [g/cm^{2}]");
  g1->GetXaxis()->CenterTitle();  g1->GetYaxis()->CenterTitle();
  g1->GetYaxis()->SetTitleOffset(0.7); g1->GetYaxis()->SetTickSize(0.05);
  graphLayout(g2,2,3,2);
  g2->SetTitle("Energie/X_{max}- Analyse");
  g2->GetXaxis()->SetTitle("lg(E/eV)");
  g2->GetYaxis()->SetTitle("<X_{max}> in [g/cm^{2}]");
  g2->GetXaxis()->CenterTitle();  g2->GetYaxis()->CenterTitle();
  g2->GetYaxis()->SetTitleOffset(0.7);
  graphLayout(g3,1,2,2);
  g3->SetTitle("Differenz");
  g3->GetXaxis()->SetTitle("lg(E/eV)");
  g3->GetYaxis()->SetTitle("Differenz in [g/cm^{2}]");
```



```
  g3->GetXaxis()->CenterTitle(); g3->GetYaxis()->CenterTitle();
  g3->GetYaxis()->SetTitleOffset(0.7);

  // Draw all things on the first pad
  c1->cd(1);  gPad->SetBottomMargin(0.17);
  g2->Draw("AP");  g2->Draw("AP");  g1->Draw("SAMEP");
  g1->GetYaxis()->SetTickSize(0.1);
  g1->Draw("SAMEP");
  leg->Draw();
  c1->cd(1)->SetGrid(1, 1);
  c1->cd(1)->Update();

  // Draw all things on the second pad
  c1->cd(2);  gPad->SetBottomMargin(0.15);
  g3->Draw("AP");  g3->Draw("AP");
  c1->cd(2)->SetGrid(1, 1);
  c1->cd(2)->Update();

  c1->Print(File1.c_str(),"pdf");
  c1->SaveAs(File3.c_str());
  c1->Clear();
  c1->Divide(1,2);
  // Do the same process for the second zmax moment
  for (int i=0; i<Results2.size()-7; ++i){
    if (Results1.at(i).size() != 0) {
      if (Results1.at(i).at(4) !=-1 )  ↩
    g4->SetPoint(i,Results1.at(i).at(0),Results1.at(i).at(4));    // 1.Wert teilt dem  ↩
    Datenpaar eine Nummer zu, 2. Wert ist der x-Wert, dritter der y-Wert
      if (Results1.at(i).at(5) !=-1 ) g4->SetPointError (i, 0, Results1.at(i).at(5));
    }
    if (Results2.at(i).size() != 0) {
      if (Results2.at(i).at(4) !=-1 )  ↩
    g5->SetPoint(i,Results2.at(i).at(0),Results2.at(i).at(4));
      if (Results2.at(i).at(5) !=-1 ) g5->SetPointError (i, 0, Results2.at(i).at(5));
    }
    if (Results1.at(i).size() == 0 || Results2.at(i).size() == 0) continue;
    if (Results1.at(i).at(4) !=-1 ) {
      ↩
    g6->SetPoint(i,(Results1.at(i).at(0)+Results2.at(i).at(0))/2,Results2.at(i).at(4)  ↩
    - Results1.at(i).at(4));
      if (Results1.at(i).at(5) !=-1 && Results2.at(i).at(5) !=-1)
      g6->SetPointError (i, 0, Results2.at(i).at(5) + Results1.at(i).at(5));
    }
  }

  graphLayout(g4,4,2,2);
  g4->SetTitle("Energie/X_{max}- Analyse");
  g4->GetXaxis()->SetTitle("lg(E/eV)");
  g4->GetYaxis()->SetTitle("sigma(X_{max}) in [g/cm^{2}]");
  g4->GetXaxis()->CenterTitle(); g4->GetYaxis()->CenterTitle();
  g4->GetYaxis()->SetTitleOffset(0.7);
  graphLayout(g5,2,3,2);
  g5->SetTitle("Energie/XMax- Analyse");
  g5->GetXaxis()->SetTitle("lg(E/eV)");
  g5->GetXaxis()->SetTitle("sigma(X_{max}) in [g/cm^{2}]");
  g5->GetXaxis()->CenterTitle(); g5->GetYaxis()->CenterTitle();
  g5->GetYaxis()->SetTitleOffset(0.7);
  graphLayout(g6,1,2,2);
  g6->SetTitle("Differenz");
  g6->GetXaxis()->SetTitle("lg(E/eV)");
  g6->GetXaxis()->SetTitle("Differenz in [g/cm^{2}]");
  g6->GetXaxis()->CenterTitle(); g6->GetYaxis()->CenterTitle();
  g6->GetYaxis()->SetTitleOffset(0.7);

  // Draw all things on the first pad
  c1->cd(1); gPad->SetBottomMargin(0.15);
  g5->Draw("AP"); g5->Draw("AP"); g4->Draw("SAMEP");
  leg2->Draw();
  c1->cd(1)->SetGrid(1, 1);
```



```
  c1->cd(1)->Update();
  // Draw all things on the second pad
  c1->cd(2); gPad->SetBottomMargin(0.15);
  g6->Draw("AP"); g6->Draw("AP");
  c1->cd(2)->SetGrid(1, 1);
  c1->cd(2)->Update();
  c1->Print(File2.c_str(),"pdf");
  c1->SaveAs(File4.c_str());
  c1->Clear();
}

void compare4ResVecs(vector<vector<double> > Results1,vector<vector<double> >  ↩
    Result2,vector<vector<double> > Result3,vector<vector<double> > Result4,string  ↩
    legentry1,string legentry2,string legentry3, string legentry4, string  ↩
    outputfilefilename)
{
  string File1 = "XmaxMoment1" + outputfilefilename   + ".pdf" ;
  string File2 = "XmaxMoment2" + outputfilefilename   + ".pdf";
  string File3 = "XmaxMoment1" + outputfilefilename   + ".root" ;
  string File4 = "XmaxMoment2" + outputfilefilename   + ".root" ;
  TCanvas *c1 = new TCanvas("c1","Ergebnisvergleich", 200, 100, 1024, 768);
  TGraphErrors *g1 = new TGraphErrors();  TGraphErrors *g2 = new TGraphErrors();
  TGraphErrors *g3 = new TGraphErrors();  TGraphErrors *g4 = new TGraphErrors();
  TGraphErrors *g5 = new TGraphErrors();  TGraphErrors *g6 = new TGraphErrors();
  TGraphErrors *g7 = new TGraphErrors();

  TLegend *leg = new TLegend(0.12,0.95,0.35,0.72, "");   //Position in % des Canvas,  ↩
    x oben links , y- oben links , z- untern rechts, y-unten rechts
  leg->AddEntry(g1,legentry1.c_str(),"p"); leg->AddEntry(g2,legentry2.c_str(),"p");
  leg->AddEntry(g3,legentry3.c_str(),"p"); leg->AddEntry(g4,legentry4.c_str(),"p");

  TLegend *leg2 = new TLegend(0.12,0.2,0.34,0.45, "");   //Position in % des Canvas,  ↩
    x oben links , y- oben links , z- untern rechts, y-unten rechts
  leg2->AddEntry(g1,legentry1.c_str(),"p"); leg2->AddEntry(g2,legentry2.c_str(),"p");
  leg2->AddEntry(g3,legentry3.c_str(),"p"); leg2->AddEntry(g4,legentry4.c_str(),"p");

  c1->Divide(1,2);
  gStyle->SetTitleSize(0.06,"x");  gStyle->SetTitleSize(0.06,"y");
  gStyle->SetLabelSize(0.06, "X");  gStyle->SetLabelSize(0.06, "Y");
  gStyle->SetTitleFontSize(0.06);  gStyle->SetTitleFontSize(0.06);

  // produces the graphs for the first zmax moment
  for (int i=0; i < Results1.size(); ++i) {
      g1->SetPoint(i,Results1.at(i).at(0),Results1.at(i).at(2));
      g1->SetPointError (i, 0, Results1.at(i).at(3));
      g2->SetPoint(i,Results2.at(i).at(0),Results2.at(i).at(2));
      g2->SetPointError (i, 0, Results2.at(i).at(3));
      g3->SetPoint(i,Results3.at(i).at(0),Results3.at(i).at(2));
      g3->SetPointError (i, 0, Results3.at(i).at(3));
      g4->SetPoint(i,Results4.at(i).at(0),Results4.at(i).at(2));
      g4->SetPointError (i, 0, Results4.at(i).at(3));
      g5->SetPoint(i,(Results1.at(i).at(0) + Results2.at(i).at(0))/2,  ↩
    Results2.at(i).at(2) - Results1.at(i).at(2));
      g5->SetPointError (i, 0, Results1.at(i).at(3) + Results2.at(i).at(3));
      g6->SetPoint(i,(Results1.at(i).at(0) + Results3.at(i).at(0))/2,  ↩
    Results3.at(i).at(2) - Results1.at(i).at(2));
      g6->SetPointError (i, 0, Results1.at(i).at(3) + Results3.at(i).at(3));
      g7->SetPoint(i,(Results1.at(i).at(0) + Results4.at(i).at(0))/2,  ↩
    Results4.at(i).at(2) - Results1.at(i).at(2));
      g7->SetPointError (i, 0, Results1.at(i).at(3) + Results4.at(i).at(3));
  }

  graphLayout(g1,1,3,2);   // Results1
  graphLayout(g2,2,2,1.5);   // Results2
  graphLayout(g3,8,5,1.5);   // Results3
  graphLayout(g4,4,26,1.5);   // Results4
  g1->SetTitle("Energie/X_{max} - Analyse");
  g1->GetXaxis()->SetTitle("lg(E/eV)");
  g1->GetYaxis()->SetTitle("<X_{max}> in [g/cm^{2}]");
```

```
  g1->GetXaxis()->CenterTitle();
  g1->GetYaxis()->CenterTitle();
  g1->GetYaxis()->SetTitleOffset(0.7);
  g1->GetYaxis()->SetTickSize(0.05);

  graphLayout(g5,2,2,1.5);   // Results2 - Results1
  graphLayout(g6,8,5,1.5);   // Results3 - Results1
  graphLayout(g7,4,26,1.5);   // Results4 - Results1
  string title = "Differenz der Datens#ddot{a}tze zu dem all_CL0_CO3 Datensatz" ;
  g5->SetTitle( title.c_str() );
  g5->GetXaxis()->SetTitle("lg(E/eV)");
  g5->GetYaxis()->SetTitle("Differenz in [g/cm^{2}]");
  g5->GetXaxis()->CenterTitle(); g5->GetYaxis()->CenterTitle();
  g5->GetYaxis()->SetTitleOffset(0.7);

  // Draw all things on the first pad
  c1->cd(1); gPad->SetBottomMargin(0.17);
  g1->Draw("AP"); g2->Draw("SAMEP");
  g3->Draw("SAMEP"); g4->Draw("SAMEP");
  leg->Draw(); c1->cd(1)->SetGrid(1, 1);
  c1->cd(1)->Update();

  // Draw all things on the second pad
  c1->cd(2); gPad->SetBottomMargin(0.15);
  g5->Draw("AP"); g6->Draw("SAMEP"); g7->Draw("SAMEP");
  c1->cd(2)->SetGrid(1, 1);
  c1->cd(2)->Update();
  c1->Print(File1.c_str(),"pdf");
  c1->SaveAs(File3.c_str());
  c1->Clear();
   //  //  // STARTS the same process again for the 2.Moment of Xmax  /// //// ///
  c1->Divide(1,2);
  for (int i=0; i< Results1.size()-8; ++i){
      g1->SetPoint(i,Results1.at(i).at(0),Results1.at(i).at(4));    // 1.Wert teilt
    dem Datenpaar eine Nummer zu, 2. Wert ist der x-Wert, dritter der y-Wert
      g1->SetPointError (i, 0, Results1.at(i).at(5));
      g2->SetPoint(i,Results2.at(i).at(0),Results2.at(i).at(4));
      g2->SetPointError (i, 0, Results2.at(i).at(5));
      g3->SetPoint(i,Results3.at(i).at(0),Results3.at(i).at(4));
      g3->SetPointError (i, 0, Results3.at(i).at(5));
      g4->SetPoint(i,Results4.at(i).at(0),Results4.at(i).at(4));
      g4->SetPointError (i, 0, Results4.at(i).at(5));
    g5->SetPoint(i,(Results1.at(i).at(0)+Results2.at(i).at(0))/2,Results2.at(i).at(4)
    - Results1.at(i).at(4));
      g5->SetPointError (i, 0, Results2.at(i).at(5) + Results1.at(i).at(5));
    g6->SetPoint(i,(Results1.at(i).at(0)+Results3.at(i).at(0))/2,Results3.at(i).at(4)
    - Results1.at(i).at(4));
      g6->SetPointError (i, 0, Results3.at(i).at(5) + Results1.at(i).at(5));
    g7->SetPoint(i,(Results1.at(i).at(0)+Results4.at(i).at(0))/2,Results4.at(i).at(4)
    - Results1.at(i).at(4));
      g7->SetPointError (i, 0, Results4.at(i).at(5) + Results1.at(i).at(5));
  }

  graphLayout(g1,1,3,2);   // Results1
  graphLayout(g2,2,2,1.5);   // Results2
  graphLayout(g3,8,5,1.5);   // Results3
  graphLayout(g4,4,26,1.5);   // Results4
  g1->SetTitle("Energie/X_{max} - Analyse");
  g1->GetXaxis()->SetTitle("lg(E/eV)");
  g1->GetYaxis()->SetTitle("sigma(X_{max}) in [g/cm^{2}]");
  g1->GetXaxis()->CenterTitle();  g1->GetYaxis()->CenterTitle();
  g1->GetYaxis()->SetTitleOffset(0.7); g1->GetYaxis()->SetTickSize(0.05);

  graphLayout(g5,2,2,1.5);   // Results2 - Results1
  graphLayout(g6,8,5,1.5);   // Results3 - Results1
  graphLayout(g7,4,26,1.5);   // Results4 - Results1
```



```
  g5->SetTitle( title.c_str() );
  g5->GetXaxis()->SetTitle("lg(E/eV)");
  g5->GetYaxis()->SetTitle("Differenz in [g/cm^{2}]");
  g5->GetXaxis()->CenterTitle();
  g5->GetYaxis()->CenterTitle();
  g5->GetYaxis()->SetTitleOffset(0.7);

  // Draw all things on the first pad
  c1->cd(1); gPad->SetBottomMargin(0.17);
  g1->Draw("AP"); g2->Draw("SAMEP");
  g3->Draw("SAMEP"); g4->Draw("SAMEP");
  leg2->Draw(); c1->cd(1)->SetGrid(1, 1);
  c1->cd(1)->Update();
  // Draw all things on the second pad
  c1->cd(2);
  gPad->SetBottomMargin(0.15);
  g5->Draw("AP"); g6->Draw("SAMEP");
  g7->Draw("SAMEP");
  c1->cd(2)->SetGrid(1, 1);
  c1->cd(2)->Update();
  c1->Print(File2.c_str(),"pdf");
  c1->SaveAs(File4.c_str());
  c1->Clear();
}

vector<double> CreateBinBoundaryVec()
{
  vector<double> binBoundaries;
  binBoundaries.push_back(17.8);  binBoundaries.push_back(17.9);
  binBoundaries.push_back(18.0);  binBoundaries.push_back(18.1);
  binBoundaries.push_back(18.2);  binBoundaries.push_back(18.3);
  binBoundaries.push_back(18.4);  binBoundaries.push_back(18.5);
  binBoundaries.push_back(18.6);  binBoundaries.push_back(18.7);
  binBoundaries.push_back(18.8);  binBoundaries.push_back(18.9);
  binBoundaries.push_back(19.0);  binBoundaries.push_back(19.1);
  binBoundaries.push_back(19.2);  binBoundaries.push_back(19.3);
  binBoundaries.push_back(19.4);  binBoundaries.push_back(19.5);
  binBoundaries.push_back(20.2);
  return binBoundaries;
}

/*=======================================================
   Utility functions for making histograms
=======================================================*/

TH1D* makeHist(vector<double> vec, bool autorange, double xmin, double xmax, string
    histname)
{
  double max, min;
  if(vec.size()>0){
    max = vec.at(0);
    min = vec.at(0);
    vector<double>::iterator it;
    for(it=vec.begin() ; it < vec.end(); it++) {
      if(*it > max) max = *it;
      if(*it < min) min = *it;
    }
  }

  TH1D* hist;
  if(autorange==1){
    hist = new TH1D(histname.c_str(), "", 35, 1/8.*(9.*min-max), 1/8.*(9.*max-min));
  }
  else if(autorange==0){
    hist = new TH1D(histname.c_str(), "", 35, xmin, xmax);
  }
  if(vec.size()>0){
    vector<double>::iterator it;
    for(it=vec.begin() ; it < vec.end(); it++){
```



```
      hist->Fill(*it);
    }
  }
  return hist;
}



void EmptyResultVec (vector<vector<double> > &Results1, double minnumber)
{
  int i = 0;
  while (i < Results1.size()){
    if (Results1.at(i).at(1) < minnumber){
      Results1.at(i).clear();
      i++;
    }  else {i++;}
  }
}

void graphLayout(TGraph* graph,int color,int marker,float markersize,int
    linestyle,float linesize)
{
      graph->SetMarkerColor(color);
      graph->SetMarkerSize(markersize);
      graph->SetLineColor(color);
      graph->SetLineWidth(linesize);
      graph->SetLineStyle(linestyle);
      graph->SetMarkerStyle(marker);
}
```

==========

ERKLÄRUNG

==========

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate und Ergebnisse Anderer kenntlich gemacht habe.

_____ _____

(Datum) (Ort) (Unterschrift)