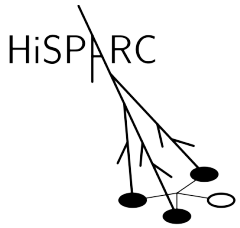
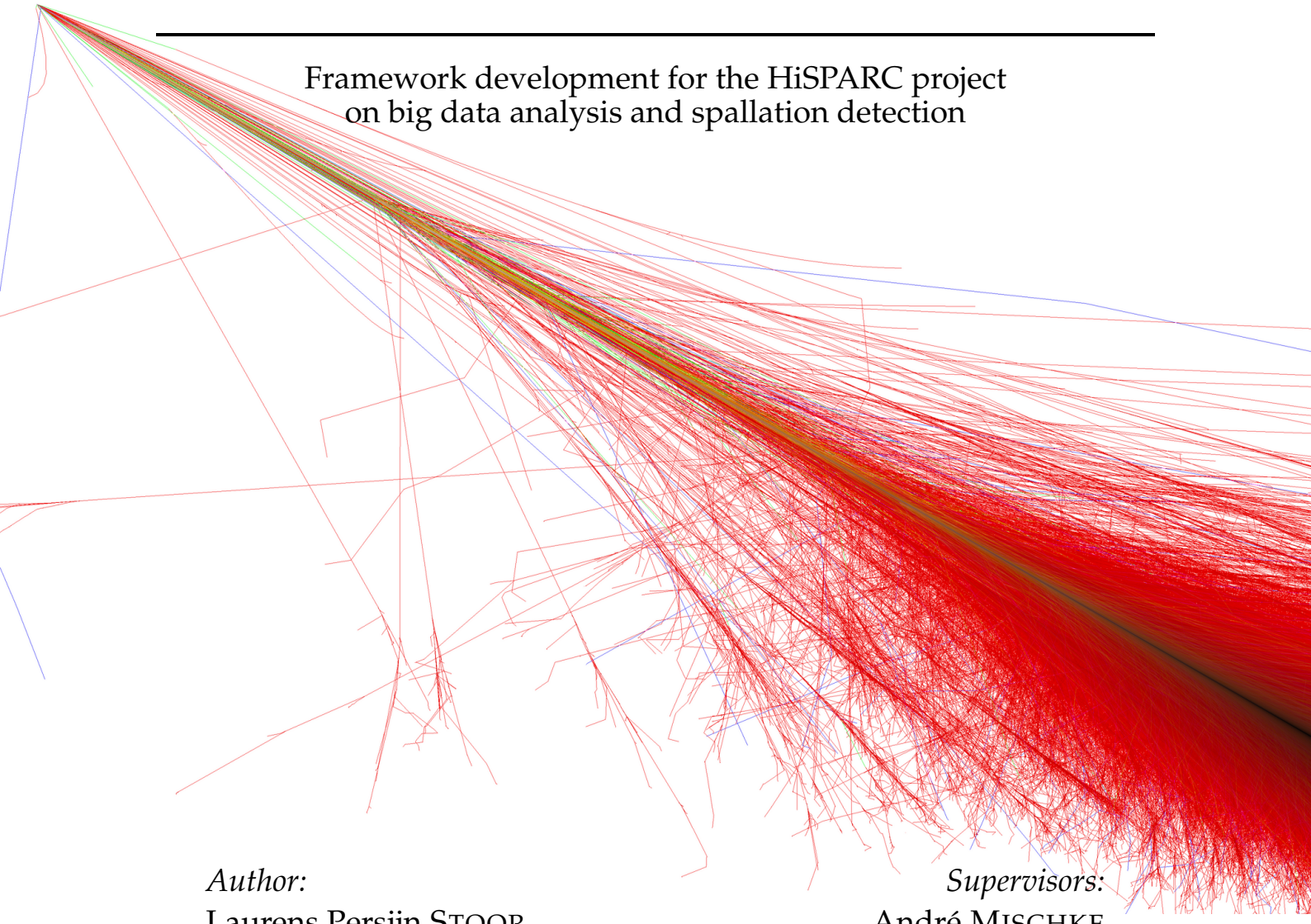


BACHELOR'S THESIS

DEVELOPMENT OF AN EXTENDED HISPARC FRAMEWORK

Framework development for the HiSPARC project
on big data analysis and spallation detection



Author:
Laurens Persijn STOOP
Study: Physics & Astronomy
Student nr.: 3986209

Supervisors:
André MISCHKE
Institute for Subatomic Physics
and
Alessandro GRELLI
Institute for Subatomic Physics

UTRECHT UNIVERSITY

15 June 2016

Cover art created by a COsmic Ray SImulation for KAscade (CORSIKA) software example, see [Kahlsruhe Institue of Technology, 2016]. The image shown is a simulation of the particle tracks for a 10^{15} eV cosmic ray proton at an 45° inclination. The electromagnetic, muonic and hadronic components of an extensive air shower are respectively shown in red, green and blue.

Abstract

The High School Project on Astrophysics Research with Cosmics (HiSPARC) detector array has been used since the change of the century to detect incoming cosmic rays in the central Dutch area. In the last few years the HiSPARC network has expanded throughout the Netherlands and now even includes stations in Great Britain and Denmark. Due to this expansion cosmic ray spallation processes should now be detectable by the array.

A framework extension to the Simulation and Analysis Program Package for HiSPARC Research and Education (SAPPHiRE) framework was developed to allow for the detection of spallation processes. This new framework, Easy gROOT Gate for SAPPHiRE (EGGS), allows for quick statistical analysis of the HiSPARC data with the data analysis framework ROOT. As EGGS is created in the Python coding language, the original HDF5 document structure of HiSPARC can be maintained.

EGGS includes functionality that allows for easy data access, initial data calibration and conversion to physical units. Further development on data calibration will be necessary to make use of the full potential of EGGS.

Summary

In the last four months I have been conducting my bachelor's research within the Subatomic Physics Institute at Utrecht University. Within this institute I worked for the ALICE QGP research group on the High School Project on Astrophysics Research with Cosmics or HiSPARC. The main goal of my bachelor's research was to develop a framework for the detection of spallation processes for the HiSPARC program. Besides this I would guide high school students in their research projects within HiSPARC.

The development of a framework for the detection of spallation processes has taken up most of my time while working on my bachelor's thesis. Even though the initial track of development fully based within the programming language ROOT was a dead end, massive headway has been made with it and the basis of the new framework is finished.

Aside from the development of the new framework, I have spent two full weeks on guiding high school students while they started up their high school research projects. I have also spent some two weeks time on making software required for the new framework to be easily installed and to make it user friendly to work with. Furthermore, I have had to learn and explore the programming language ROOT, which took two and a half weeks.

Additional side projects have been the acquisition of a new station set from Nikhef, the reorganization of the bachelor and master room and I have given an interview on my bachelor's research for the new promotional video of HiSPARC.

During my time working on my bachelor's research I have developed a number of new skills and deepened my understanding of how to teach and guide young students. With this thesis I have tried to give a good representation of the work that I have done.

This thesis starts with a solid background on cosmic rays as I have taught the high school students on these subjects. The second chapter of this thesis is intended to give a good overview of the HiSPARC program. In the third chapter I will discuss the choices that have been made during and the difficulty of the development of a framework for spallation detection. In the fourth and final chapter I will conclude what the current status of the new framework is and I will give an outlook on the next steps within HiSPARC.

Laurens Stoop
Zeist, 2016

Contents

Chapter 1

Cosmic Rays

Continuous showers of cosmic rays bombard the earth at all time. We cannot escape from these cosmic rays, not even within our homes or under the ground. Even though we call them cosmic *rays*, they are in fact highly energetic charged particles from extra-terrestrial origin traveling at nearly the speed of light. The energy range of a cosmic rays spectrum ranges from 10^6 eV up to 3×10^{20} eV [Young and Freedman, 2000, p. 1432]. These cosmic rays can initiate a shower of secondary decay particles when they hit the earth's atmosphere [Fokkema, 2012, p. 4].

Due to their abundance their composition and energy spectrum has been studied with success in a variety of ways[1]. Where most studies on cosmic rays focus on measurements in the outer reaches of the earth's atmosphere and near space, the HiSPARC project focuses on ground based detection.

Cosmic ray *spallation* processes are nuclear processes in which nucleosynthesis takes place due to the very energetic nature of cosmic rays. This process happens when a cosmic ray impacts with a piece of matter and interacts with this matter. The energy of the cosmic ray can then be used within this collision for multiple processes. It can be used *a)* to increases its kinetic energy, *b)* for a fission process itself, to overcome the strong force binding the particles or *c)* to in the creation of particles[2].

The composition and origin of cosmic rays will be discussed in section 1.1, a more detailed analysis of the interaction between cosmic rays and the atmosphere will be discussed in section 1.2 and in section 1.3 the (dis-)advantages of ground based detection of cosmic rays will be discussed.

1.1 Composition and Origin of Cosmic Rays

1.1.1 Relative abundance of primary particle nuclei

The cosmic rays that bombard our atmosphere consist of a wide variety of particles. The major species of cosmic rays consist of protons ($\approx$ 84%) and alpha particles ($\approx$ 12%). Heavier elements[3] with a higher nuclear charge represent $\approx$ 2% and electrons represent $\approx$ 1% of all cosmic rays, [Grupen, 2005, p. 78].

Within the relative abundance of the heavier nuclei lies the first hint of the source of cosmic rays. In figure 1.1 the relative abundance of the elements in cosmic rays is shown and compared to that of our solar system. It can be seen that there are major similarities, but there are also a few differences. Compared to our solar system the

[1] See [Boyle et al., 2008] for measurements in the stratosphere with balloons, see [George et al., 2009] for measurements in space, for ground based detection of cosmic rays see section 1.3.

[2] The creation of particles usually happens in spallation processes so that charge is conserved (i.e. in a process like $p + p \rightarrow np + e^+ + \nu_e$, in which one or both of the original particles is a cosmic ray).

[3] The third component of the cosmic rays are called the *heavier* elements as protons and alpha particles can be considered as Hydrogen and Helium cores.

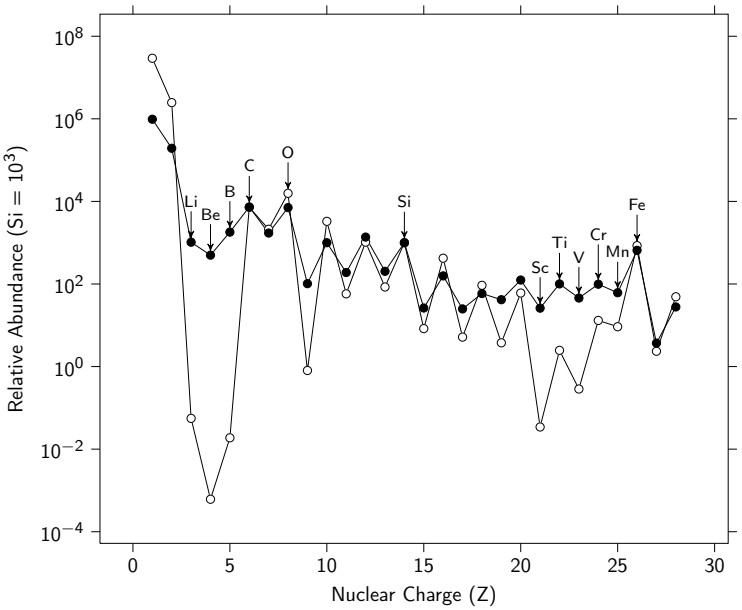


Figure 1.1 – Side by side comparison of the relative abundance of the nuclear component of cosmic rays and of the solar system. The distributions are normalized with respect to the element Silicon at 1000. The distribution of the cosmic ray relative abundance is shown with the closed circles and the solar system abundance is shown with the open circles. Image from [Fokkema, 2012, p. 5].

elements Lithium, Berylium, Boron are about 10^5 more abundant in cosmic rays. The heavier elements Scandium, Titanium, Chromium and Manganese are about $10^2 - 10^3$ more abundant in cosmic rays.

This higher abundance of these elements can be explained by considering the fact that cosmic rays have transversed only a couple g cm^{-2} of matter between their source and the top of our atmosphere. Within such amounts of matter spallation processes can occur and produce the observed distribution. To have transversed a couple g cm^{-2} of matter the cosmic rays need to have wandered for at least $10^6 - 10^7$ light years of space, as the average density of the interstellar medium is of the order 10^{-25} g cm^{-3} [Henley and Garcia, 2007, p. 597-601].

1.1.2 The energy spectrum

The energy spectrum of cosmic rays is shown in figure 1.2. The cut-off at the lower bound of the energy spectrum can be understood by considering the influence of the solar gravitational field. Cosmic rays with a low energy ($< 10^{10}$ eV) can be bent by and trapped within the sun's gravitational field. Measurements at these low energy levels have shown that the flux of these cosmic rays modulates with the solar cycle[4].

The relatively high abundance of several elements within cosmic rays, combined with the fact that the suns magnetic field cannot trap particles with an energy level higher than 10^{10} eV, is a strong indication that cosmic rays come from outside of our solar system[5] [Henley and Garcia, 2007, p. 598].

The knee of the energy spectrum lies at about 5×10^{15} eV and is thought to be caused by different propagation effects and-or new accelerating mechanisms. This change can be explained by taking into account that the gravitational field of our galaxy can no

[4]The solar cycle has a period of 11 years and the low energy cosmic rays can there for be studied easily by repeated measurements of the intensities over multiple periods [Fokkema, 2012, p. 6].

[5]Due to the fact that cosmic rays with a energy $< 10^9$ eV can be generated and contained within our solar system, most textbooks define this as the lower limit for the energy of cosmic rays.

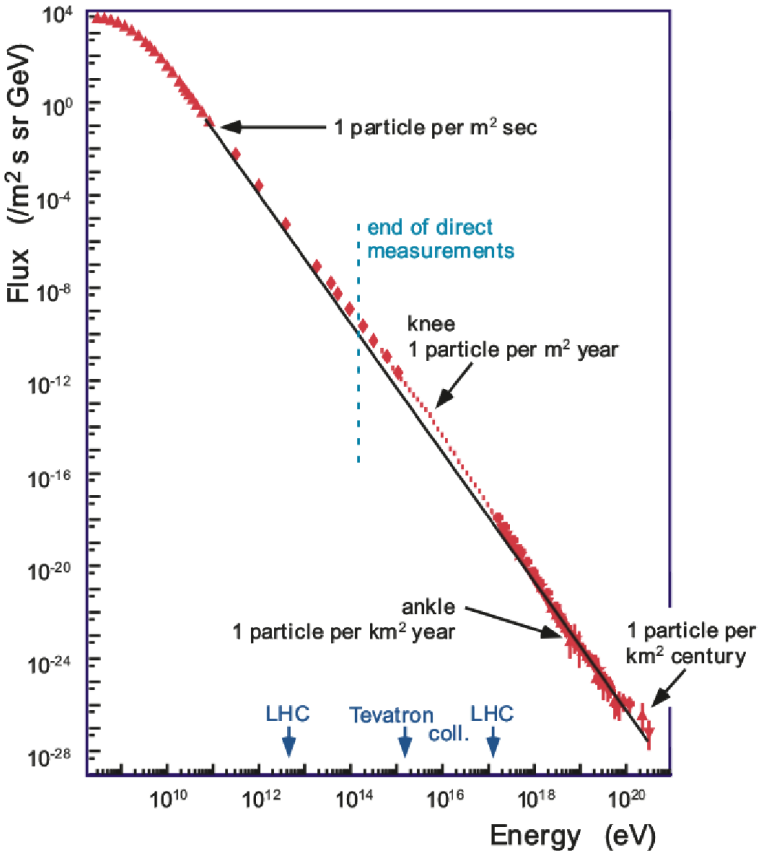


Figure 1.2 – The flux of primary cosmic ray particles as a function of their energy is plotted. In red the data points for different experiments are shown. The top left triangles correspond to data from LEAP, a balloon based experiment. The diamonds are data points from the PROTON satellite experiment. The points correspond to the AGASA experiment. The closed circles correspond to the Yakutsk experiment and the up and down triangles in the bottom right respectively correspond to data points from AUGUR and Fly's Eye. The black line is a fitted power law (i.e. $F(E) \sim E^{-\gamma}$) with a spectral index $\gamma = 2.75$. Recreated in color from [Alania et al., 2009].

longer contain cosmic rays once they rise above the knee energy within its gravitational field and thus signals an overlap region in which the cosmic rays are no longer bound by our galaxy[6].

The ankle of the energy spectrum lies at about 2×10^{18} eV and signals a change in the spectral index of the fitted power law, but this part of the energy spectrum is plagued by very low statistics. At this energy level the gravitational field of our galaxy has little to no effect on the cosmic rays and above this energy level no known galactic accelerator exists. Cosmic rays with an energy level above the ankle are thus considered to be extra-galactic and are called *ultra-high energy cosmic rays* or UHECRs for short.

1.1.3 Acceleration mechanisms

In general, it is believed that cosmic rays with an energy level lower than 10^{18} eV are of galactic origin, though no hard proof for this statement has been found. It is now also good to consider that the most favored hypothesis for the generation and acceleration of cosmic rays, supernova shock wave acceleration, has difficulty in accounting for

[6]The fact that the milky way can no longer contain cosmic rays once they rise above the knee energy within its gravitational field does not imply that they are no longer generated within our galaxy.

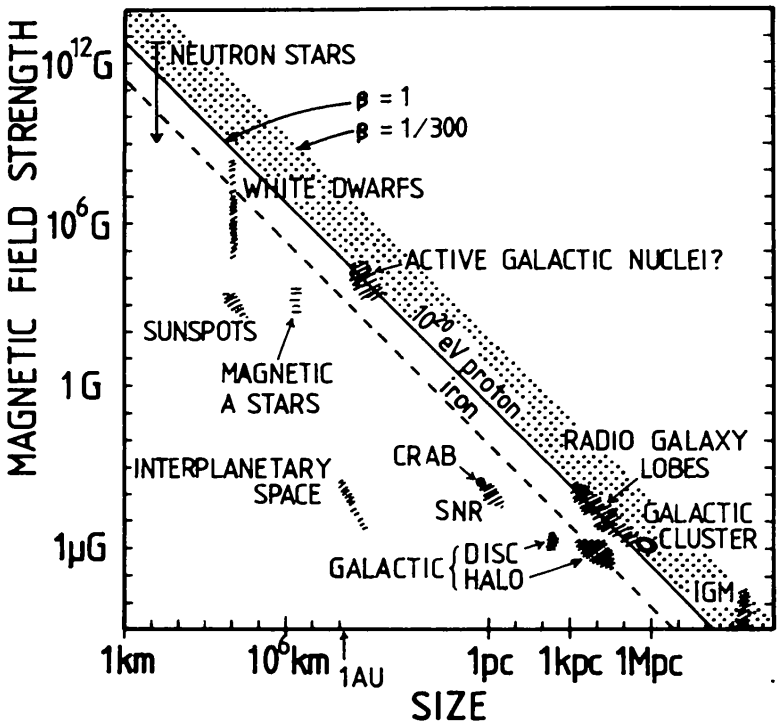


Figure 1.3 – The Hillas plot of the possible sites of particle acceleration. Objects below the diagonal black line cannot accelerate protons to 10^{20} eV. The dashed line shows the same limit of acceleration but then for iron. For more context see [Hillas, 1984, p. 426].

particles with an energy above 10^{15} eV[7].

A candidate accelarator for particles with energies between 10^{15} eV and 10^{18} eV is Fermi acceleration, a theoretical interstellar medium accelerator that accelerates cosmic rays by collision with other particles with co-moving magnetic fields. Other candidates are pulsars or a binary system formed by a combination of a neutron star and a (super-) giant. This candidate comes from the recent detection of cosmic rays originating from the binary system Hercules X-1 and Cygnus X-3. A more in depth description of the galactic origin of cosmic rays can be found in [Henley and Garcia, 2007, p. 599-601].

1.1.4 The Greizen, Zatsepin and Kuzmin limit

In the year 1966 it was theorized by Greizen, Zatsepin and Kuzmin (GZK) that UHECRs have enough energy available to directly interact with a cosmic microwave background (CMB) photon to produce a pion. Above this so-called pion production threshold at 6×10^{19} eV the cosmic rays will lose energy as they produce pions via the delta resonance when they interact with a CMB photon, see Eq. (1.1.1) and (1.1.2).

$$p + \gamma_{CMB} \rightarrow \Delta^{+} \rightarrow p + \pi^{0} \tag{1.1.1}$$

$$p + \gamma_{CMB} \rightarrow \Delta^{+} \rightarrow n + \pi^{+} \tag{1.1.2}$$

The UHECRs will continue to lose energy with every interaction they have with a CMB photon until they are under the pion production threshold for UHERCs. This limit is called the GZK limit and is expected to cause a pile-up of cosmic rays below this limit.

Next to the build-up of cosmic rays just under the GZK limit, this effect also puts a limit on the distance particles can travel if they have a similar or higher energy than the

[7]The full mathematical approach and exactly how the equilibrium between the centrifugal and Lorentz force can account for the galactic containment, can be found in [Grupen, 2005, p.81].

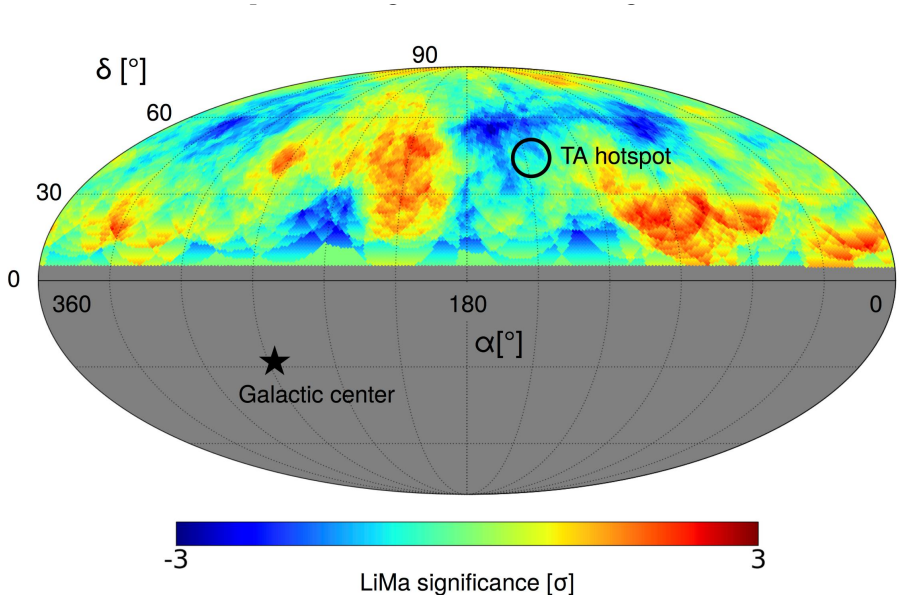


Figure 1.4 – A Mollweide view of the measured directional flux for cosmic rays detected by HiSPARC. A significance of 3σ can be read as a fluctuation of 0.3% of the average flux. The TA hotspot indicates the probable UHECR hotspot as found by the TA collaboration in 2014, see [Abbasi et al., 2014]. Image from [Schultheiss, 2016, p. 9].

GZK limit, due to the interaction length within the interstellar medium. This maximum distance from the source of these UHECRs is calculated to be 50 Mpc[8].

The experimental verification of the GZK limit[9] is still debated as the different measurements of this limit are not unambiguous. Especially, the detection of a non negligible number of events above this limit[10] is difficult to describe by theory.

1.1.5 Acceleration sites of cosmic rays

It was Hillas in 1984 who found that the limit to which a source can be accelerated is depends on the size of the source and the strength of its magnetic field[11],

$$E_{15} \leq 0.5 \beta_s Z B_{\mu G} L_{pc}. \tag{1.1.3}$$

When you plot the magnetic field size versus the size of the source you get what is known as the Hillas plot, the original, as made by Hillas, is shown in figure 1.3. The Hillas plot is generally used to determine whether an acceleration site can produce UHECRs.

Due to the uniform arrival of cosmic rays from all directions, as can be seen in figure 1.4, and our limited understanding of the magnetic field of our galaxy, the acceleration sites for galactic cosmic rays ($E < 10^{18}$ eV) have not yet been determined. In contrast, the sources for UHECRs could be relatively accurately determined if the statistics of UHECRs could be improved, because the galactic and solar gravitational fields have little impact on their trajectories.

[8] A distance of 50 Mpc is relatively close in cosmology, our galaxy is about 35 kpc in diameter and the local supercluster, containing around a 100 galaxies, is 30 Mpc in size. See also [Fokkema, 2012, p. 8].

[9] The first to gather data at this limit were the members of the the AGASA project, followed by the Auger array. In the last few years HiRes, Fly's Eye and the Haverah Park Array have joined the search for experimental justification of this limit.

[10] The most energetic particle measured till this day is the Oh-My-God particle that was detected in October 1991 at 3.2×10^{20} eV by the Fly's Eye cosmic ray detector, as published in [Bird et al., 1993].

[11] In the Hillas equation for the upper limit to which a source can accelerate a particle the energy E_{15} is in units of 10^{15} eV, the magnetic field $B_{\mu G}$ is given in μ G, the velocity of the shock wave β_s in units of c and L_{pc} in units of pc.

1.2 Interaction with the atmosphere

When traveling through the interstellar medium (ISM) cosmic rays only transverse a small column density, a couple of g cm^{-2} at most[12]. This is the reason that galactic cosmic rays usually travel $10^6 - 10^7$ light years before a spallation process occurs and also the reason that UHERCs can travel up to 1.6×10^{11} ly before they have interacted with one or multiple CMB photons and have lost enough energy to fall below the GZK limit[13].

Compared to the ISM the earth's atmosphere has a very high column density of about 1030 g cm^{-2} and cosmic rays are thus very likely to interact with the earth's atmosphere. At between 15 and 20 km from the surface the first interaction happens. As each interaction either has multiple daughters and-or involves the particles in the atmosphere, a cascade of secondary particles is initiated. In the following paragraphs the different interactions within this particle shower are discussed.

In this section general knowledge of the four forces of nature, the elementary particles, their interactions and their decay rules is assumed, this knowledge can be obtained from [Henley and Garcia, 2007].

1.2.1 Extensive Air Shower

An simplified scheme of an interaction between a cosmic ray and a particle or molecule in the atmosphere is shown in figure 1.5. As can be seen from this diagram there are three different components of a cosmic ray shower that reach the ground directly and two that are scattered into the atmosphere, the latter being the Cherenkov and fluorescence radiation. The totality of the cascade is called an *extensive air shower* [Enqvist, 2009].

1.2.2 Hadronic component

The hadronic component of an extensive air shower consists mainly of the debris of the air particles that the cosmic ray and its daughters interacted with. Initially this component consists of all kinds of exotic baryons and possibly even pentaquarks[14], but they decay quickly into the most stable baryons through the strong interaction with interaction times of about 10^{-23} s. These stable baryons, the neutron and proton[16], then transverse the remainder of the atmosphere.

The proton loses energy within the atmosphere due to electromagnetic interactions. The neutron transverses the atmosphere without interacting with the atmosphere as it is charge neutral[17]. The number of ground reaching exotic baryons is very small as they will most likely have decayed. See [Grupen, 2005, p. 144] for a more in depth analysis of the hadronic component.

[12] Some cosmic rays are of extra-galactic origin and will thus have transversed the intergalactic medium or the voids between galaxies, the column density of which is a fraction of the ISM.

[13] For more background see section 1.1.2 and 1.1.4.

[14] The creation of pentaquarks is possible within a cosmic ray due to the (ultra-) high energy of the primary particle with respect to the energy needed for an pentaquark[15].

[15] The energy of ultra high energy cosmic rays can extend well into 10^{19} eV, with the majority of the events happening around 10^{15} eV. The energy of the observed $J/\Psi p$ resonances for pentaquark states in the $\Lambda_b^0 \rightarrow J/\Psi K^- p$ decays lies in the order of 4×10^9 eV [Aaij et al., 2015]. The energy needed for such an event is thus available.

[16] Neutrons and protons are not absolutly stable, their respective half-lifes are approximatly 6.2×10^2 and 3.1×10^{39} s.

[17] Even though the neutron does not interact electromagnetically it can interact with a particle in case of a direct hit, however the chance of this happening is very small.

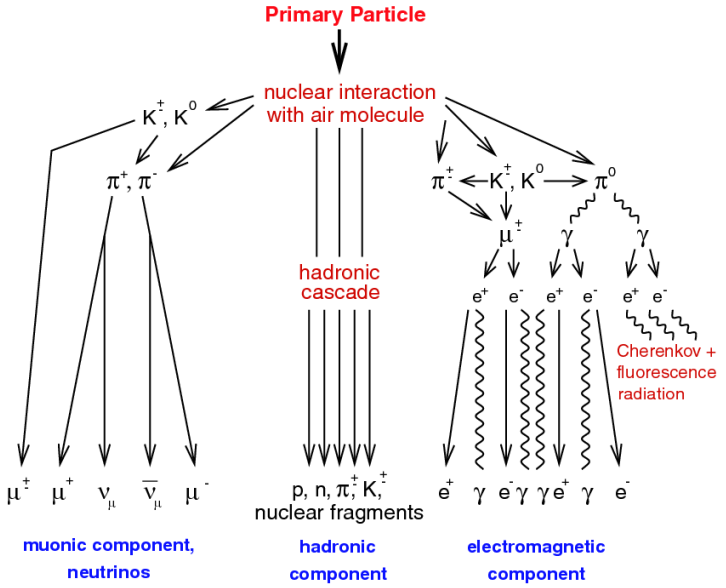


Figure 1.5 – Diagram of an extensive air shower. An incident high-energy proton strikes the top of the atmosphere and produces a cascade shower. The three main components are separated for clarity and example processes are shown. Not all decay laws have been taken into account for clarity. A more realistic cascade can be seen on the cover page. Image from [Enqvist, 2009].

1.2.3 Electromagnetic component

The electromagnetic component of an extensive air shower consists of all photons and electrons[18] formed within the extensive air shower[19]. The origin of these electrons lies within a multitude of decay modes from the unstable mesons. Typical decay times are in the order of $10^{-10} - 10^{-8}$ s. The vast majority of the particles detected at ground level are from the electromagnetic component of the shower as they can be produced at the lowest energy level.

The electromagnetic component loses energy through a couple of processes. Pair production and annihilation are the most dominant at the highest energy levels, due to a quickly increasing number of daughters this effect reduces the energy of the particles quickly. When due to this lower energy the interaction cross section of pair production decreases the Compton effect will become the most dominant process for energy loss[20]. At energy levels below 1 MeV the photoelectric effect dominates. Bremstrahlung and ionization losses also absorb a part of the energy of the electromagnetic component of an extensive air shower. A full description of all the possible decay modes and Feynman diagrams of all processes can be found in [Haverhoek, 2006, p. 7-12].

1.2.4 Muonic component

The muonic part of an extensive air shower is created in a similar way as the electromagnetic component. The majority of the muons are created when the unstable heavy mesons decay, a minority is created via pair production.

[18] When electrons are discussed both electrons and positrons are intended.

[19] Even though a muon is also a lepton and could therefor also be considered as part of the electromagnetic component it is not taken into account. The muon interacts in a different way with the atmosphere compared to the electron.

[20] In fact there is no energy loss, but only energy transfer to atomic electrons. its called energy loss as the energy is lost from the extensive air shower.

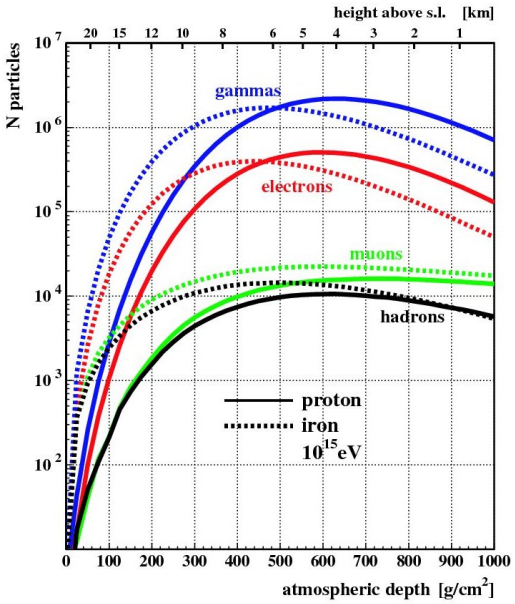


Figure 1.6 – Longitudinal development of a cosmic ray extensive air shower, plotted is the number of particles versus the atmospheric depth for a 10^{15} eV proton and iron core. Image from [Enqvist, 2009, p. 16].

Muons do not have a strong interaction with the particles in the atmosphere and have a relative long life time ($\sim 2.2 \times 10^{-6}$ s). The muon decay modes are given by

$$\mu^- \rightarrow e^- + \bar{\nu}_e + \nu_\mu \tag{1.2.1}$$

$$\mu^+ \rightarrow e^+ + \nu_e + \bar{\nu}_\mu. \tag{1.2.2}$$

At an energy level of $\sim 10^{10}$ eV ($v \approx 0.999c$) the Lorentz factor γ for the muons is of the order ~ 22. The resulting decay length is $s \approx \gamma\tau c = 15$ km [Fokkema, 2012, p. 22]. This Lorentz boost of the decay length will result in the arrival of most muons at sea level, as the muons are created between $10 - 20$ km above the ground.

1.2.5 Particle flux in the atmosphere

To perform measurements of an extensive air shower it is important to know the particle flux in the atmosphere[21]. When the particle flux is too low, at ground level, the properties of the extensive air shower can not be extrapolated. The following paragraphs are not an exhaustive description, a more comprehensive approach can be found in [Grupen, 2005, p. 142-151].

As the general shape of an extensive air shower is radially symmetric, the main characteristics can be obtained by on the one hand considering the development of the particle flux with respect to the atmospheric depth and on the other hand looking at the radial footprint of the extensive air shower.

In figure 1.6 the longitudinal development of an extensive air shower is plotted for a 10^{15} eV proton and an iron core. In this figure a distinction is made between the different components of a cosmic ray. From this figure one can see that the number of photons, electrons and hadrons in a shower is beyond its maximum when they hit the earth's crust. In contrast to this, the number of muons seems to be relatively stable even at larger atmospheric depths.

In figure 1.7 the radial distribution of an extensive air shower is shown for both a proton and an iron core primary particle. The primary particle had an energy of 10^{15}

[21] This reason for this might not be clear at the moment, but it will be explained in section 1.3.1.

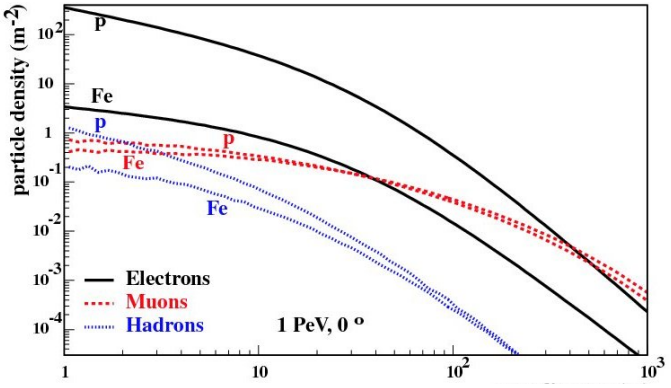


Figure 1.7 – The radial development of the particle density for a primary particle of 10^{15} eV and no inclination, both the distribution for a proton and an iron core are shown. Image from [Enqvist, 2009, p. 10].

eV and no inclination, the different components are shown separately to allow for differentiation. It can be seen that the particle density for the muonic part of an extensive air shower is more stable over longer distances than the electromagnetic and hadron component.

1.3 Ground Based Detection

When a cosmic ray enters the earth's atmosphere and interacts with a particle, it will cause a cascade of secondary particles to shower onto the earth. Within these extensive air showers lies the information about the primary particle, the cosmic ray. The energy of the primary particle can be calculated by comparing the observed shower with simulations and its directional origin can be found by tracking the way the shower hits the earth[22]. In the following sections the way an extensive air shower can be detected and tracked will be discussed.

1.3.1 Information within an air shower

The detection of extensive air showers requires some careful planning and smart use of detection techniques. The reason for this is the mind boggling amount of information stored within an extensive air shower.

A medium energy ($E \sim 10^{15}$ eV) event[23] contains roughly 10^6 particles. As the decay time for some events is in the order of 10^{-8} s[24], the sampling rate would need to be of the order 10^9 s^{-1} for a period of 10^{-5} s, as within that time the shower will have hit the earth. The total number of data points would then be in the order of 10^{10}. If every event can be stored in 12 Byte[25], the total amount of storage for *one* event would be about 120 GB.

It is therefore not reasonable to save all the information of all particles within an extensive air shower. This is also the reason why the detection of cosmic rays on the ground is ussualy done by measuring only a selective number of properties of the extensive air shower that it caused.

[22] For more background see section 1.1 and 1.2.

[23] Such an event would occur every few seconds on every square meter of the earth's surface.

[24] The charged kaon and pion components are of this order, the neutral kaon and pion component have decay times of the order 10^{-16} s.

[25] This would allow you a meager accuracy of a $\sim 10^3$ data points in every spatial dimension within a certain range.

1.3.2 Basic strategy for detection

In general, all strategies for ground based detection of extensive air showers have the same basic strategy behind them. They collect part of the data within the air shower and compare this to some simulations of extensive air showers to estimate the properties of the full shower and its primary particle.

The simulation of extensive air showers can be done in a wide variety of ways, but it is usually based on some form of Monte-Carlo simulation. The basic input that these simulations require is the primary particle, energy and zenith angle. From this the extensive air shower is simulated using High-Energy interaction models, depending on the simulation and the goal of the experiment certain observables, like the number of electrons and the shower maximum, are saved. This simulated shower is then put through a detector response algorithm so that the measured signal of the detector can be compared to the simulations. For more information on extensive air shower simulations and the distribution models they produce see [Enqvist, 2009].

1.3.3 Detection methods

Detection of an extensive air shower through its components can be done in different ways. The major principles will be discussed. It should be taken into account that on many occasions multiple detection principles are combined to reduce any possible bias in the detection technique. For more information on the different detection methods used see [Haverhoek, 2006, p. 4-5] [Enqvist, 2009] and [Fokkema, 2012, p. 23-28].

The first way of measuring an extensive air shower is by directly measuring the particles that make it to the ground, by tracking detectors and-or calorimeters. This will give you information on the number particles and their energy. To get a good image of the particle density a reasonable area should be completely filled with detectors[26]. This method allows you to directly sample the full footprint of an air shower, the drawback is that the amount of data gathered on the background is big.

By measuring the cosmic ray particles that reach the ground with an scintillator array type of setup less background is measured, as this will allow for an increase in the area measured while not having to deal with massive amounts of data. The big drawback of an array type setup is that not all particles are detected, which will significantly increase the minimum cosmic ray energy that can be detected. Another problem with scintillator detectors is that they are very susceptible to environmental background[27].

Measurements of the fluorescence of atoms or Cherenkov light, as emitted in the atmosphere, with telescopes or wide angle photo-multiplier tubes (PMT) allows for ground based detection of the progress of an extensive air shower through the atmosphere. The way an air shower develops through the atmosphere is very well described by current theory, so these measurements give very precise data. The disadvantage with measurements on Cherenkov light is that the measurements can only be done on clear moonless nights, which limits the detection time considerably.

Measurements on the high energy muons and neutrinos emitted by the decay processes, can be done with Cherenkov detectors. However, these detectors would need to be placed deep underground to reduce the background of other particles.

[26] The rule of thumb for an area completely filled with detectors is that at least half the area is filled, with the size of the two detectors, with intermediate space, smaller than the size of an event.

[27] An in-depth comparison on the difference between various array type detectors recently used in research can be found in [Radu et al., 2008].

Chapter 2

The HiSPARC Experiment

The HiSPARC project was created in 2002 out of the Nijmegen Area High School Array (NAHSA) project to recreate its initial success on a national scale. The HiSPARC project was set up with two main goals: "*to study cosmic rays and to expose high school students to the challenges and rewards of scientific research*", [Fokkema, 2012, p. 26].

By introducing the concepts of astrophysics and by letting high school students construct and maintain their own detector, they are introduced into the world of scientific research. At the same time, the involvement of high schools in the project ensures the creation of a sparse detector array[1]. However, as the majority of the detectors is financed by high schools and maintained by them, the detectors should be cheap, robust and easily maintainable. More background on the design criteria can be found in [Fokkema, 2012, p. 29-34]. In section 2.1 the station setup will be further discussed and the design of the detectors will be discussed in section 2.2. The extent of the HiSPARC network, its general properties and extent are discussed in section 2.3.

As the HiSPARC project is distributed over the whole Dutch area, it requires a well documented and organized software architecture. To achieve this a standardized format for data is used by all stations[2]. As cosmic ray research can only be done with a large number of events, open access to the complete database at a high bandwidth of every station is required[3]. In section 2.4 the software architecture of the HiSPARC project will be discussed.

2.1 The HiSPARC Station Setup

Every HiSPARC station has been set up in a similar manner to allow for cross station data analysis.

The basic hardware make-up of a station consists of two or four scintillator detectors[4], a HiSPARC 'red box', a GPS and a computer. Some stations also include their own weather station to allow for very accurate environmental calibration. In the following paragraphs, the different components and how they are assembled are discussed.

2.1.1 Scintillator detectors

The choice for scintillator detectors in the HiSPARC project was made because scintillator detectors can very accurately measure charged particles, the main component of an

[1] This sparse detector array is achieved as the average distance between Dutch high schools in urban areas is between 500 and 1500 meter. At these distances the average muon density at ground level is of order 1 m^{-2} for a 10^{15} eV event.

[2] A HiSPARC station is the collection of detectors maintained by one school, a station can have either 2 or 4 detectors.

[3] An additional requirement for good scientific research is the integrity of the data, but this is true for any research that wants to be taken seriously.

[4] The four detector setup is the new standard, but used to be too expensive for most high schools.

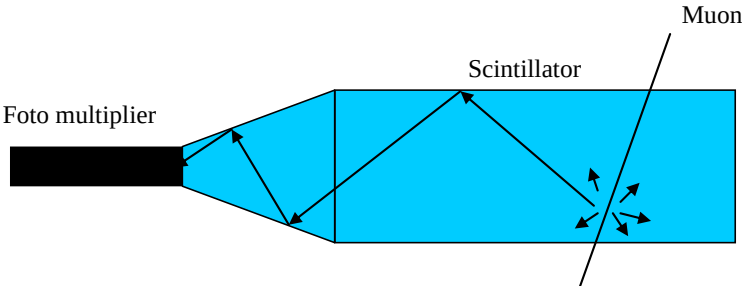


Figure 2.1 – Schematic of an assembled scintillator plate, with fishtail waveguide and photomultiplier tube. The schematic is *not* to scale. The path of a muon is shown with an example path of a photon generated by scintillation. Image from [Otto, 2007].

extensive air shower. The way a scintillator detector works is through a process called scintillation. Scintillation is a process where ionized particles loose energy in a material and cause luminescence. The amount of light generated through luminescence is proportional to the number of particles that hit a scintillator, [Fokkema, 2012, p. 37].

The detectors that HiSPARC uses are cheap and very efficient anthracene scintillator plates with an area of 0.5 m^2 and thickness of 2 cm. The choice for this type of scintillator was made to have a detector that is primarily susceptible to the energy levels of muons. Detailed information of the scintillators used by HiSPARC and their properties can be found in [Fokkema, 2012, p. 34-41]

These scintillator plates are glued to a waveguide on which a photomultiplier tube is connected. The full detector is wrapped in aluminum foil and black plastic, this is done to keep most of the photons generated by scintillation in the plate and to protect the plate from the environment. A schematic of the detector can be found in figure 2.1.

The assembly of a detector and its placement is done by high school students and their teachers under the supervision of a researcher. The assembly and placement procedures can be found in [de Laat and van Eijk, 2016].

2.1.2 Photomultiplier tubes

When a charged particle from an extensive air shower hits a HiSPARC detector, scintillation occurs and light is emitted throughout the scintillator plate. When this light hits a side that is not connected to the waveguide, the light is reflected back into the scintillator plate. When the light hits the side that is connected to the waveguide, it continues and is reflected within the waveguide until it is captured by the photomultiplier tube[5].

Inside the photomultiplier tube the photon signal is changed into an analog electric signal. The analog electric signal is directly proportional to the number of photons that entered the photomultiplier tube, hence the name. The way this conversion of a photon into an analog electronic signal is achieved, is shown schematically in figure 2.2.

The photomultiplier's used by HiSPARC have changed a couple of times since the project started in 2002. The reason for this is that the earlier versions of the photomultiplier's did not produce a proportional signal for more than a couple of photons. Figure 2.3 shows a measurement of the proportionality of a photomultiplier as used at the beginning of the HiSPARC project. The newest generation photomultiplier tubes are made by Nikhef and no longer show large deviations of the proportionality of the photomultiplier's[6].

[5] The light in a scintillator plate normally leaves the plate when it hits the side, but as the HiSPARC scintillators are wrapped in aluminum foil and black plastic, most of the light is reflected back.

[6] A comparison between an old and a new photomultiplier can be found in figure 3.5 on page 32.

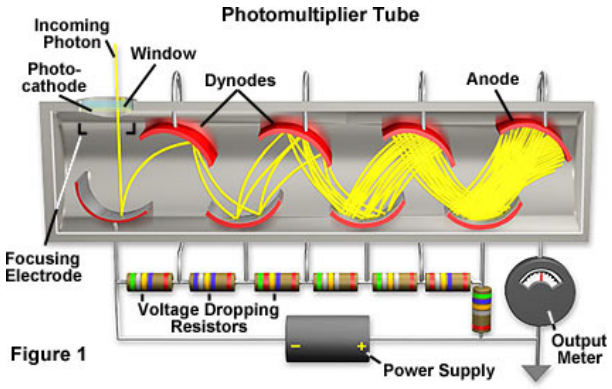


Figure 2.2 – Schematic of a photomultiplier tube and an indication of how it works. The path of an incoming photon is shown. When the incoming photon is absorbed in the photo cathode an electron is expelled from the material. This electron is accelerated to the first dynode, due to a voltage difference. Upon impact in the dynode multiple electrons are expelled from the surface. These newly freed electrons are further accelerated to the next dynode. This process continues until the electrons hit the anode, in which they are absorbed and the electrical signal is measured. The number of generated output electrons is linear to the number of incoming photons. Image from [de Vries, 2012].

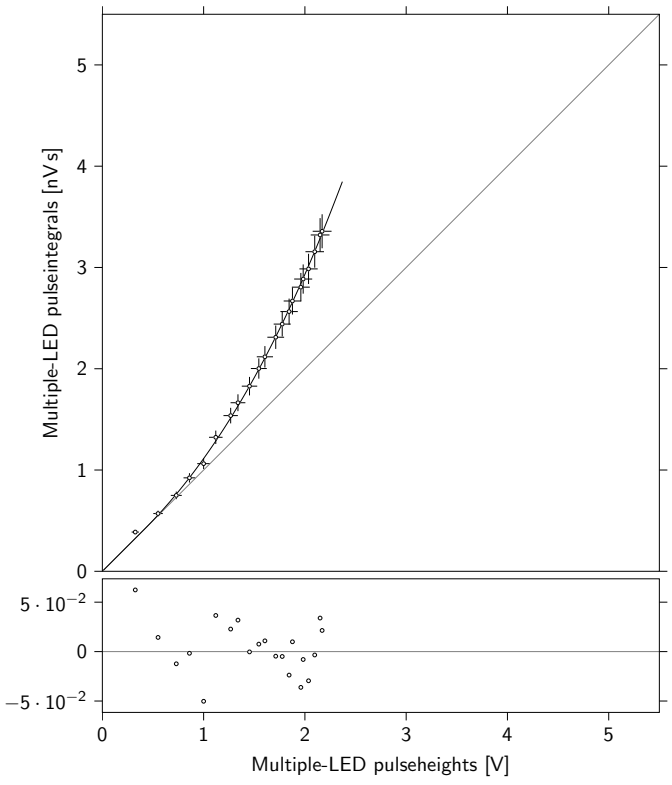
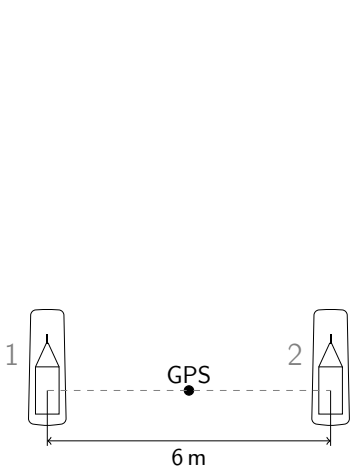
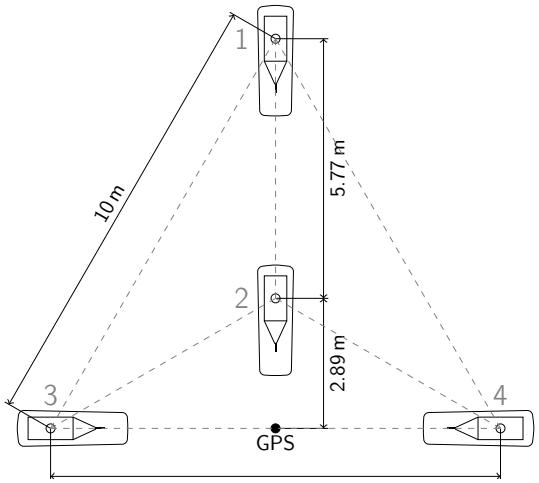


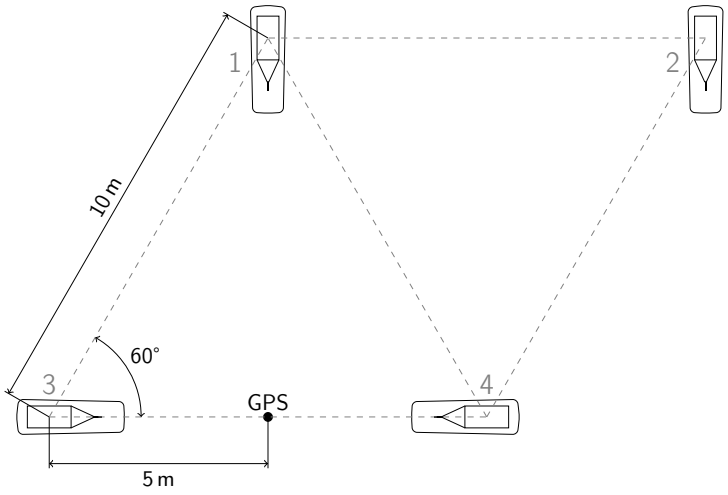
Figure 2.3 – Measurement of the proportionality of a photomultiplier as used before 2014. Theoretically, the pulse height should rise proportionally to the pulse integral for a proportional photon to electron conversion in a photomultiplier tube. The pulse integral is shown versus the pulse height. The measurement was conducted by shining the light of multiple LED's with a well known output into a photomultiplier tube. Measurements have been done by Arne de Laat, the current HiSPARC PhD student. Image from [de Laat, 2016].



(a) Two detector setup



(b) Four detector setup small roof



(c) Four detector setup large roof

Figure 2.4 – The three HiSPARC station layouts. Images 2.4a and 2.4b from [de Laat and van Eijk, 2016], image 2.4c from [Fokkema, 2012, p. 43].

2.1.3 Station layout

The detectors used by the HiSPARC project are placed on the roof of the participating schools to allow for a wide field of view. The setup on the roof depends on the number of detectors and the amount of space available.

For a two detector type station the layout of the detectors is shown in figure 2.4a. For the setup of the detectors of a four detector station there are two possible lay-outs depending on the size of the roof that is available. For a small roof the layout as shown in figure 2.4b is used, for a larger roof the layout as shown in figure 2.4c. If possible the latter layout is used as it allows for more accurate reconstruction of the direction of the extensive air shower. The setups shown are preferred setups as they have the best angular resolution for the least amount of space.

As of February 2016 the exact layout of a station can be registered so that the angular direction of an air shower that hits only one station can be reconstructed.

Figure 2.5 – The HiSPARC 'Red box' in which the trigger and ADC are located. Two detectors are connected. The blue LED, located in the middle, is turned on, indicating that an event is observed. Image from [de Laat and van Eijk, 2016].

2.2 Data acquisition and storage

As an extensive air shower leaves its footprint of particles on a HiSPARC station in less than 1.5 µs and as the difference in arrival times of a particles within an extensive air shower is of the order 10 ns, the electronic control of the photomultiplier's needs to be fast. The nanosecond control of the photomultiplier's is carried out by the HiSPARC 'Red Box', together with the trigger matrix and the fast analog to digital conversion of the signal and GPS data. The processes controlled by the HiSPARC 'Red Box' are further discussed in section 2.2.1.

It is important that the data storage has been structured in such a way that data integrity and open access to all the data is possible, as these are requirements to do extended research within the HiSPARC project. To achieve this the data acquisition of HiSPARC has been ordered into layers called tiers. In section 2.2.2 the data management of HiSPARC is discussed.

2.2.1 The HiSPARC 'Red Box'

The HiSPARC 'Red Box' allows researchers to control how a signal, as generated by scintillation, is captured for two detectors. The 'Red Box' allows control over the voltage at which the photomultiplier's are operated, it allows for limited control over the sampling rate of the analog to digital converter, it allows control over the trigger matrix and it allows full control over the GPS. The way it allows control will be discussed in the following paragraphs. A photograph of a HiSPARC 'Red Box' is shown in figure 2.5. Two boxes can be joined to operate together a total of four detectors.

The HiSPARC 'Red Box' is connected to a detector via two cables, one to operate the high voltage for the photomultiplier and one to read out the signal of the photomultiplier anode. The high voltage needed to drive the photomultiplier can be regulated via a computer that is connected to the 'Red Box' to obtain the best signal for that specific photomultiplier tube and scintillator plate. The procedure to obtain the best signal for that specific photomultiplier tube is usually repeated every month.

The sampling rate of the fast analog to digital converter (ADC) for a HiSPARC 'Red Box' is usually only regulated once. The readout unit contains four ADC's and a 200 MHz crystal for the two detector channels. By driving one ADC per channel on the rising slope of the crystal signal and one on the falling edge the sampling frequency

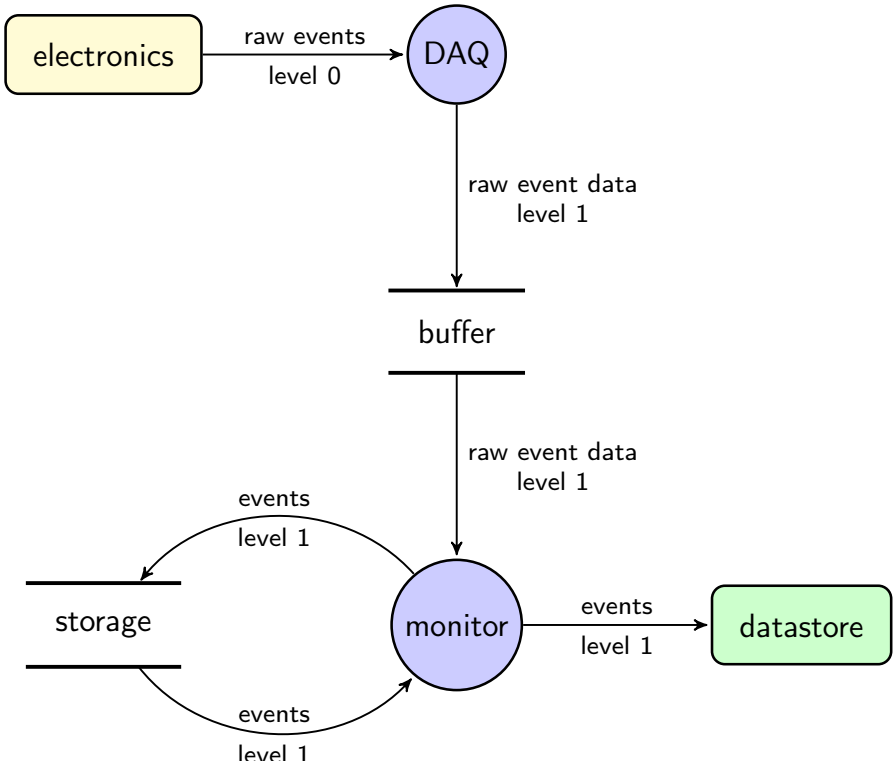


Figure 2.6 – Data flow diagram of a HiSPARC station. The three main units, between which the data flows, are shown. The HiSPARC detector electronics, the 'Red Box' is shown in yellow. The station computer is shown in blue and the HiSPARC data store located at Nikhef is shown in green. Image from [Fokkema, 2012, p. 62].

can be raised to 400 MHz. This allows for measurements at a 2.5 ns interval rate. Realignment of the ADC's is usually only done when a new photomultiplier is installed, [Fokkema, 2012, p. 41-44].

The trigger matrix that is used by the HiSPARC 'Red Box' is rather elaborate in the sense that for each channel two thresholds can be set. The reason for this is that a high signal has a high probability of being generated by a particle in the detector, so the ADC output is stored. There is also a low threshold, if three detectors measure an event then the data is also stored. For a station with only two detectors, there is only one trigger, set at the high threshold.

The HiSPARC 'Red Box' has an inbuilt buffer that allows for 30µs, this allows for five tracks of event data to be stored continuously. Due to this no dead time of the detectors is observed.

The GPS unit that is part of every HiSPARC station is used to exactly determine the location of the station and it is used to define the time stamp of an event. With the use of the HiSPARC 'Red Box' the GPS is configured to continuously switch between *full position mode* and *overdetermined clock mode*. The switching happens 86400 times per 24 hours and results in a very accurate position and GPS time[7].

Details of the structure of the electronics contained within the HiSPARC 'Red Box' can be found in [Otto, 2007].

[7]The accuracy of the GPS is very important for the time stamp of the events as it is used to check the 200 MHz internal clock of the HiSPARC 'Red Box'.

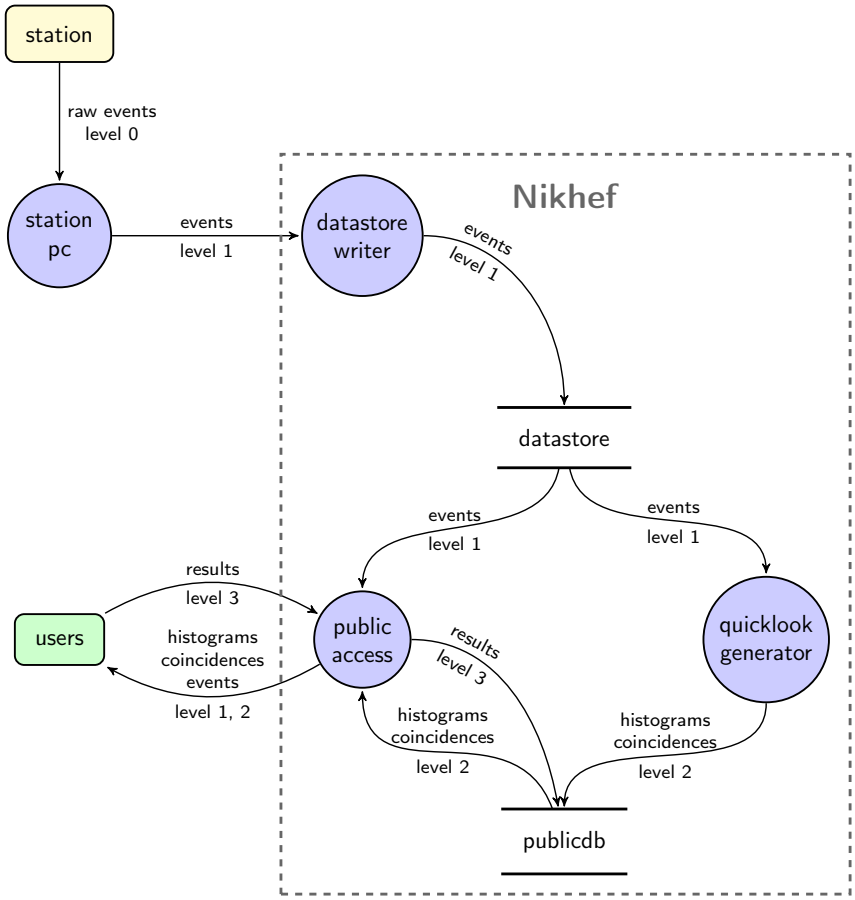


Figure 2.7 – Data flow diagram of the HiSPARC network. The three main units between which the data flows are shown. A HiSPARC station is shown in yellow. The HiSPARC data store is shown in blue and the users are shown in green. Image from [Fokkema, 2012].

2.2.2 Data management

The HiSPARC data management can be considered on two scales, firstly the data flow of the station, see figure 2.6, can be considered, secondly the data flow of the whole system can be looked at, see figure 2.7.

Once an event signal has been converted to a digital signal by the HiSPARC 'Red Box' it is sent to the program `DAQ` on the station computer as tier 0 data. The `DAQ` program checks if the data is ok, does preliminary analysis and sends the data together with the configuration settings to the `buffer` as tier 1 data. The `monitor` retrieves the data from the `buffer` and creates a structured array of the data. This array is then stored into `storage` until enough data is gathered to send a batch to the `data store writer` at Nikhef, [Fokkema, 2012, p. 58-60]. In figure 2.6 a data flow diagram of a HiSPARC station is shown.

Once a batch of tier 1 data has been send from a station computer to the HiSPARC `data store writer` its origin is verified and the data is stored in a buffer. Once the data from one station for a whole day has been received the data is written into one file within the `data store`. The location, structure and properties of the data are then added to that of the HiSPARC HDF5 meta datafile. The tier 1 data is then used to generate histograms, coincidence data and event summary data, see table 2.1 for the structure of this array. This tier 2 data is stored in the `public database`. Both the `public database` and the `data store` data can be called up by users through the

`public access` program, [Fokkema, 2012, p. 60-66].

Field	Type	Description [unit]
`event_id`	unsigned 32-bit integer	unique number *within* dataset []
`ext_time stamp`	32-bit integer	time of event in UNIX time stamp [s]
`nanoseconds`	unsigned 32-bit integer	time of event after the time stamp [ns]
`pulse height (4x)`	signed 16-bit integer	maximum signal pulse height [ADC]
`integral (4x)`	signed 32-bit integer	integral of the signal [ADC.ns]
`n# (1,2,3,4)`	signed 16-bit integer	estimate for the number of particles []
`t# (1,2,3,4)`	signed 16-bit integer	trigger time for detector [ns]
`t_trigger`	signed 16-bit integer	relative time of trigger time stamp [ns]

Table 2.1 – Structure of the event summary data table tier 2, as it is stored in HDF5 by the HiSPARC server. For each field (HDF5 table column) the type and description is give. Updated version of table 3.1 from [Fokkema, 2012, p. 61].

2.3 HiSPARC network

Since the start of the HiSPARC project the number of clusters has steadily been growing over the years. A total of 133 stations have been active in the last year[8]. An overview of the Dutch part of the HiSPARC network is shown in figure 2.8.

The HiSPARC network has been divided into clusters. All the stations in one cluster are around the university or research center that coordinates that cluster. In the recent years some clusters have spawned sub-clusters due to the expansion of the network. These subcluster are created out of interest in HiSPARC by school outside the main clusters. An overview of the number of stations per cluster is shown in table 2.2.

Cluster	2003	2005	2007	2009	2011	2016
Amsterdam	0	16	20	27	29	37
Eindhoven	0	0	0	8	18	18
Leiden	0	3	8	9	15	18
Bristol (UK)	0	0	0	0	3	16
Nijmegen	5	9	11	11	11	14
Utrecht	0	1	1	2	5	12
Enschede	0	0	1	2	7	7
Groningen	0	4	4	4	4	4
Birmingham (UK)	0	0	0	0	0	4
Aarhus (DK)	0	0	3	3	3	3

Table 2.2 – The number of active HiSPARC stations per cluster for the period 2003 – 2016, stations are counted when they've made contact to the HiSPARC server. Table adapted from table 2.1 from [Fokkema, 2012, p. 55], with the number of active stations in 2016 added.

From table 2.2 it can be clearly seen that in recent years HiSPARC has gone beyond the borders and is still expanding. However in some regions the average activity per station per year has been dropping over the years. An example of this is the Groningen cluster where the activity has dropped from 80% in 2005 to 25% in 2016.

[8] A station is said to be active when it has made contact with the HiSPARC server, without producing an error code.

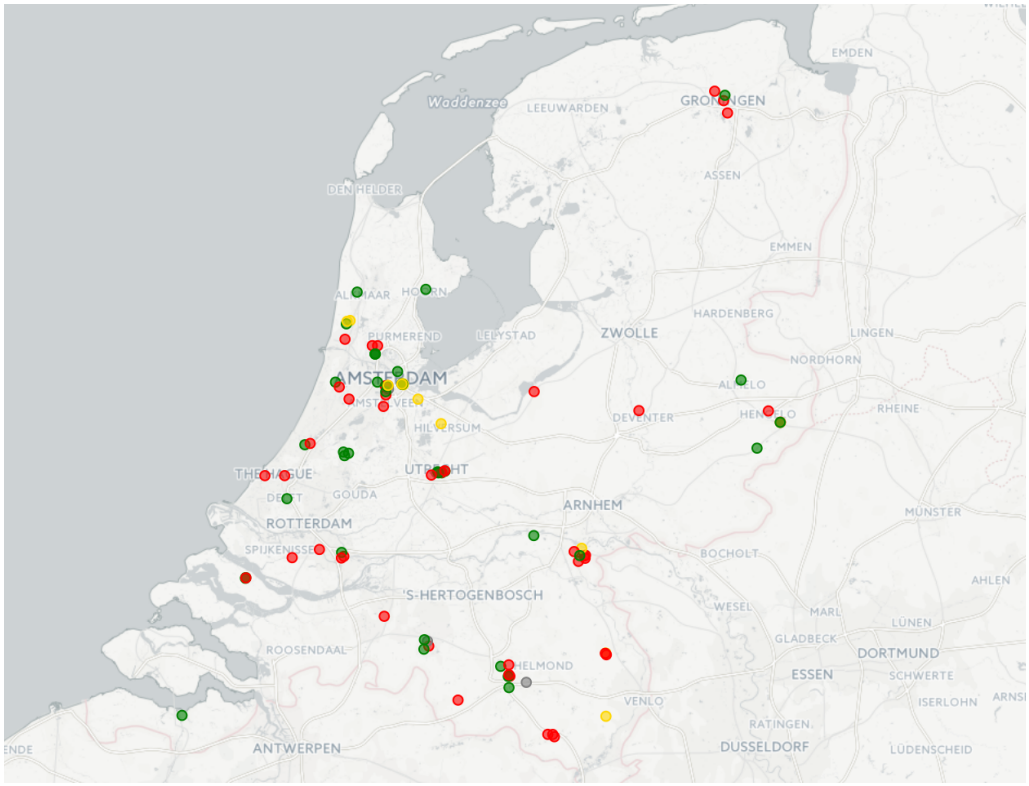


Figure 2.8 – The Dutch part of the HiSPARC network, stations that are active are shown in green (50%), stations that have a problem in sending data are shown in yellow (10%) and stations that have been inactive for more than a week are shown in red (40%). Overview made mid May 2016.

2.4 The SAPPHiRE framework

In the early years of the HiSPARC project a library of functions was collected and shared with the people involved in the analysis of the HiSPARC data. These functions could download data, process events and set up basic simulations in a modular way. Over time the requirements of the HiSPARC library became more complex and required common interfaces for experimental data and simulations. To achieve this SAPPHiRE was developed as a framework for HiSPARC.

In section 2.4.1 the properties of a framework are described and in section 2.4.2 the properties of the HiSPARC framework SAPPHiRE are discussed.

2.4.1 Properties of a framework

A software library is a just a collection of functions, to be used in a specific way and with a specific goal. In general the code and functions of a library is written without the intention of reusability or extensibility, as opposed to a framework. According to [Fokkema, 2012, p. 71], the major differences between a library and a framework are:

1. modularity
2. re-usability
3. extensibility
4. inversion of control

Within a framework the modularity requires that the key property of a class or function is unique and independent of another class or function. This does not mean that

the key properties are unique, but it does mean that every function or class is a unique set of properties.

The requirement of modularity is directly linked to the requirement of re-usability, as different classes can reuse a set of functions, but with a different flow of the data. The re-usability of a framework promotes the reuse of functions in a different way, but also requires the use of a common interface.

The extensibility of a framework requires that a user can redefine part of a function or add some functionality without having to add the code to the framework. However, if the new code as created by the user is an addition to the framework and if it is useful for other users, it could be added to it.

Inversion of control means that the data flow through a program is controlled by the framework and not by a program that uses the framework. 'A framework thus contains a skeleton of complex data flow and allows programs to supply classes which handle certain tasks' [Fokkema, 2012, p. 72].

2.4.2 The HiSPARC framework

In the early years of the HiSPARC project the data analysis and simulation programs where bundled in a library. Some functions where modular and reusable, others where not. When it became clear the HiSPARC project needed to be extended to allow for complex analysis procedures for both experimental data and simulations, common interfaces where developed with a HiSPARC framework in mind. The basic functionality of the HiSPARC library was transcribed into this new framework and a Simulation and Analysis Program Package for HiSPARC Research and Education (SAPPHiRE) was born.

The main goal of the SAPPHiRE framework is to allow researchers, university students and high school pupils to analyze and simulate extensive air showers. To allow access for a wide range of users without losing track of the changes that are done by these users, revision control is enforced through the use of Git. By also making the full code of SAPPHiRE publicly available via GitHub[9], any user can download the code, but has extensively controlled submission rights.

The goal of the SAPPHiRE framework is hindered at this moment as the documentation of its functionality is limited to a reference site[10]. Each class or function does have an entry on the SAPPHiRE reference, but this is usually only the required input and the return values. No in-depth tutorial or example depository exists.

Properties of SAPPHiRE

The HiSPARC framework SAPPHiRE contains the following modules and packages[11]:

`analysis` The analysis package contains modules for the analysis of the HiSPARC data. The majority of the packages includes focus on extracting station calibration data from raw events, finding coincidences between stations and reconstructing the shower position and direction.

`api` The api allows access to the HiSPARC public database API to retrieve data and information from the HiSPARC server.

`clusters` This package allows the definition of virtual and real clusters, these definitions can then be used to see how these stations measure simulated showers.

[9] The SAPPHiRE homepage on github is located at github.com/hisparc/sapphire/.
[10] The SAPPHiRE reference site is located at docs.hisparc.nl/sapphire/index.html.
[11] Only a partial description of a module and its contents is shown here. For all the functions within a module the SAPPHiRE reference site or the SAPPHiRE code should be consulted. A full description of each module would require extensive documentation. This is not available at the moment.

corsika This module can be used to process CORSIKA simulation data, convert the data files into HDF5 and initiate simulations on the Nikhef supercomputer Stoomboot. An overview and the data of the simulations in the HiSPARC CORSIKA library can also be requested from the server. See the next paragraph for more information on CORSIKA.

esd An easy interface for fetching event data summary directly from the HiSPARC server or a local `.tsv` data file. After the file is fetched it is stored to a HDF5 file. Either normal event data or coincidence data is downloaded.

kascade This module is intended for the calibration of the HiSPARC guest setup at KASCADE, see [Fokkema, 2012, Ch. 5]. This module is part of the old HiSPARC library and has not been upgraded to the framework style.

publicdb The public database module allows the collection of raw event data (tier 1) from the server.

qsub An easy interface for submitting jobs to the Nikhef Stoomboot, which can be used only by Nikhef employees.

simulations A sub-framework to work with simulations. Simulations included are detector response algorithms, shower front simulations and more. The base class provides the data flow between the different simulations that are carried out.

storage A library with the HDF5 table descriptions for the different kind of data table structures that are used.

time_util This function converts any time stamp to UTC, GPS or local time[12].

transformations This library contains functions to transform between different coordinate and unit systems.

utils This module contains functions that are commonly used in data analysis. The functions included can show the progress of an iteration, find an index in a list of values, calculate the difference between two angles and more.

The CORSIKA library of HiSPARC

To reconstruct the properties of the cosmic ray particle that initiated the extensive air shower, simulations have to be done on the interactions between cosmic ray particles and the atmosphere.

To simulate extensive air showers the HiSPARC project uses the COsmic Ray SImulations for KAscade (CORSIKA) package. This package was developed for the KArlsruhe Shower Core and Array DEtector (KASCADE) project situated at the Karlsruhe Institute of Technology in Germany. See [Kahlsruhe Institue of Technology, 2016] for more information on this software package and the KASCADE project.

The collection of all the data generated by the simulations carried out by CORSIKA is stored on the HiSPARC server and can be accessed through the `corsika` module of the SAPPHiRE framework. The total number of simulations stored on the HiSPARC server is about 70000.

In figure 2.9 an overview of the number of simulations per energy and zenith angle is given. It can be seen that only a limited amount of simulations have been carried out at non-integer power energies. There are only a few simulations at higher energies of the cosmic ray particle.

In figure 2.10 the shower size is plotted versus the zenith angle, the discrete separation between the 68% confidence bands indicate that a measurement of the zenith angle

[12] This function has very little use, but may prevent outbursts of anger due to incorrect conversions.

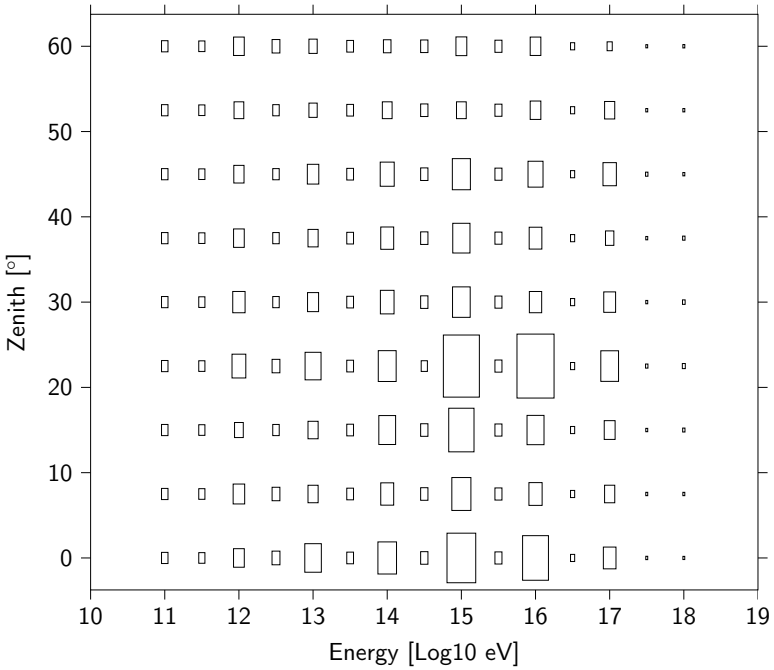


Figure 2.9 – The number of simulations plotted as function of energy and zenith angle. The square of the area is proportional to the number of simulations. Image from [de Laat, 2016].

and shower size gives a good indication for the primary particle energy. More simulations at the lower energy regimes would result in better statistics and thus in better determination of the cosmic ray energy.

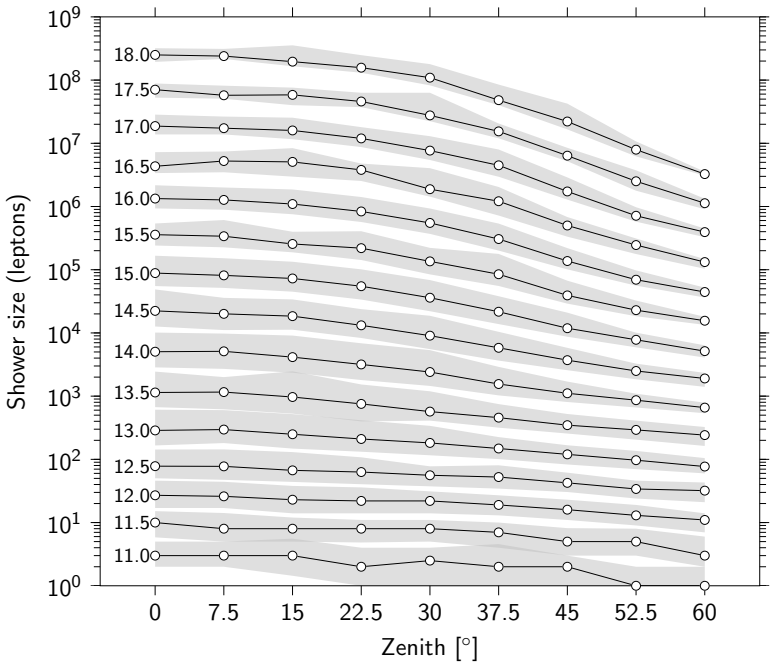


Figure 2.10 – The number of simulation leptons per shower as function of energy and zenith angle. The energy is in units of $\log_{10}$ eV. The dots are the median number of leptons in a shower per energy and angle. Within the gray bands lie 68% of the simulated showers for that energy and angle configuration. Image from [de Laat, 2016].

Chapter 3

Development of EGGS

Since the HiSPARC project has been gathering data, over 13 years ago, it has grown a lot in size. From a small array in the Nijmegen area to an international array with a 133 stations spread out over the Netherlands, the United Kingdom and Denmark.

In these years the researchers and educators working on the HiSPARC project have developed a series of lectures and tutorials to teach high school pupils about modern research and modern physics. This educational material has been approved by the Dutch Ministry of Education as part of the exam material for the 'Nieuwe Natuurkunde' (New Physics)[1] that is taught at high schools since 2014. Combined with the fact that most of the detectors built for the HiSPARC project have been made by students it can be said that the HiSPARC project is exposing high school students to the world of scientific research.

At the same time researchers and university students have worked on the analysis of the HiSPARC data that is collected and they have been perfecting the way it is gathered. For instance the measurements of the HiSPARC detectors have been compared with the measurements of KASCADE [Fokkema, 2012, Ch. 5], the dependency of the HiSPARC detectors on the weather and environment has been determined [de Vries, 2012] and the accuracy of the angular reconstruction has been increased [Schultheiss, 2016][2].

The studies done within the HiSPARC project have furthered the understanding of the way the HiSPARC detectors work and the properties of the station layout. They made headway in determining the origin and energy of the cosmic ray initiating the extensive air shower. In general these studies where done by using data from a couple of months and only the data of a select couple of stations. One of the reasons for the 'small' scale of these studies lies in the required computing time for these effects and in the required disk space for the data. The next step forward within HiSPARC would thus be a framework that would allow for big data analysis.

An interest in large scale effects and the big data analysis that this requires has led to the development of a framework that can handle and process large amounts of data in a reasonable time. The extension to the SAPPHiRE framework that has been developed to deal with these tasks is called Easy gROOT Gate for SAPPHiRE (EGGS) as it allows the processing of data within the Data Analysis Framework ROOT while SAPPHiRE handles the data[3]. In section 3.1 the choices for this combination are elaborated upon. In section 3.2 the initial properties of the new framework and the calibration of station data within EGGS is discussed.

[1] The 'Nieuwe Natuurkunde' was intended to be an update of the physics taught at high school and to better adjust the level of high school physics with the university level of physics. Key parts of the new physics are introductions into special relativity and the quantum world.

[2] This is but a selection of the studies conducted within the HiSPARC project. A more complete list of student work can be found at http://www.hisparc.nl/en/docent-student/werk-van-studenten/ and a list of publications of the HiSPARC project can be found at http://www.hisparc.nl/docent-student/publicaties/.

[3] The reason for the use of gROOT instead of ROOT is that this framework allows Python to communicate with the base of ROOT called groot.

3.1 Hatching EGGS

In this section the choices that have been made in the development process of EGGS are discussed. First the reason why EGGS needed to be developed, section 3.1.1, then the choice for the software that makes up the back-end, section 3.1.2, and finally the `bash`-script for easy installation of EGGS will be discussed in section 3.1.3.

3.1.1 The next spark for HiSPARC

Since the beginning of the HiSPARC project the determination of the energy and origin of cosmic ray particles has been the main research goal [Fokkema, 2012, p. 26]. Now that the determination of the angular origin of cosmic ray particles has been completed and automated and the influence of the weather and the environment on the properties of the HiSPARC detectors is known, the determination of the energy of the cosmic ray that initiated the extensive air shower is within reach[4].

New tracks of research

As the current research goal of the HiSPARC project comes thus to a close, it is time to think about the next step of the research conducted within the HiSPARC project.

Due to the extent of the HiSPARC array, distances of 500 km and more between stations, the detection of extra-terrestrial spallation processes[5] is possible as the two extensive air showers that are created are about $400 - 600$ km apart. These events cover a wide range of energies and thus as much data as possible should be gathered and analysed. More background on spallation can be found in section 1.1.1.

Another direction of the research of the HiSPARC project could be the measurement of ultra high energy cosmic rays (UHECR). The footprint of these UHECRs should be detectable within the HiSPARC clusters as it is in the order of a couple of kilometers. However, due to the small number of UHECRs hitting the earth's atmosphere[6] the data of multiple years of measurements should be combined to find a reasonable number of events. More information about UHECRs and the expected corresponding limit of the energy of cosmic rays can be found in section 1.1.4.

A third possible track of research for the HiSPARC program could be the measurement of the angular distribution of the origin of cosmic rays as detected on the earth's crust. Changes into this angular distribution could indicate a change in the earth's magnetic field or the rise or fall of a cosmic ray hotspot. To be able to observe these changes the data of the HiSPARC array would have to be compared over many years as these changes are not expected to happen overnight. More on the influence of gravity on cosmic rays can be found in section 1.1.5.

Big data analysis

When the requirements of these new tracks of research for the HiSPARC project are considered, it is clear that independently of the new track of the HiSPARC research, the next step is going to require big data analysis.

When analysis needs to be done on big data in the field of high energy physics, the data analysis framework ROOT is what comes to mind. As this framework is used for the data analysis at CERN, the European Organization for Nuclear Research, for the analysis of the data generated by experiments with the Large Hadron Collider. The

[4] The research leading to the energy determination is not yet finished, but the technical difficulties of the determination of the energy are expected to be finished within the next year or two [de Laat, 2016].

[5] An extra-terrestrial spallation process is a process in which a heavy nuclei cosmic ray decays into its components due to interactions with a nearby particle.

[6] The number of UHECRs hitting the earth's atmosphere is about one per century per square kilometer. The number of detectable UHECRs is expected to be a lot higher due to the large footprint these cosmic rays generate.

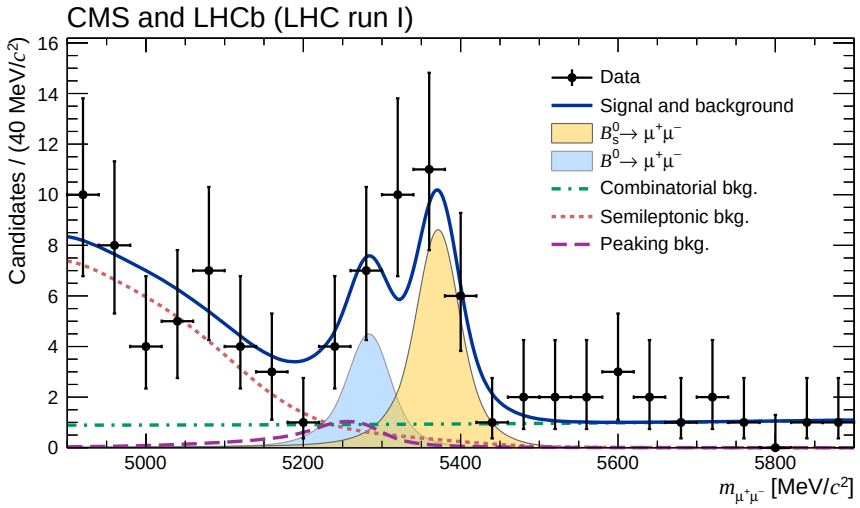


Figure 3.1 – An example of the capability of ROOT, as shown on its homepage root.cern.ch. The physics of the image is explained within [Khachatryan and et al., 2014].

other reason the ROOT framework comes to mind is the large number of users of ROOT within the Subatomic Physics Institute at Utrecht University.

The properties of ROOT

ROOT is a data analysis framework that provides all the functionalities needed to deal with statistical analysis, visualization and storage for big data analysis. The ROOT framework was initially developed for the researchers at CERN but is now widely used within the field of high energy physics due to its specialized fitting algorithms for this field of research [Brun and Rademakers, 1997].

An example of the capabilities of ROOT can be seen in figure 3.1. To create this figure a specialized statistical model was implemented within ROOT to fit the data, while ROOT provided all the blocks for the further statistical analysis of the fit. Note also the description of the quantities plotted, these have been typeset with the internal LaTeX engine of ROOT, this functionality was implemented due to the requirement of clear plots. The last thing to note is the use of elegant styles and colors within the plot to create a good overview of both the data, models for the signal and the background without cluttering the figure.

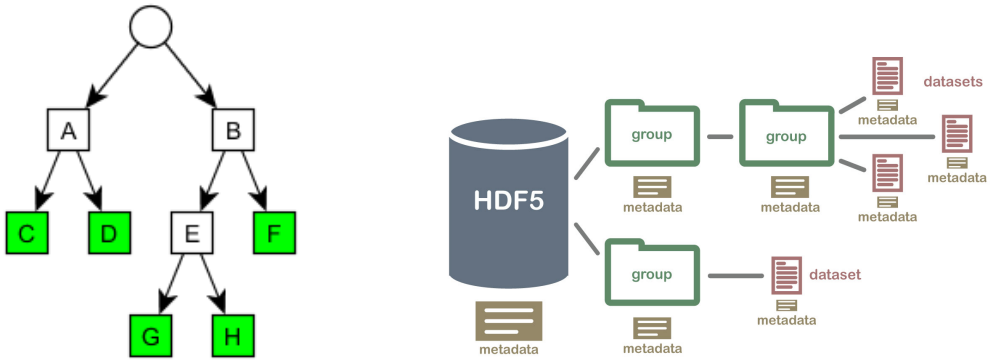
More information on the data analysis framework ROOT and its current capabilities for petabyte data storage, statistical analysis and visualization can be found in [Antcheva et al., 2009].

3.1.2 The building of EGGS

To allow big data analysis within the HiSPARC framework SAPPHiRE, an extension to the framework had to be made. The extension to the framework had to incorporate the data analysis framework ROOT within the SAPPHiRE framework. This incorporation should allow for easy analysis of the high energy physics of cosmic rays.

This extension to the HiSPARC framework SAPPHiRE has been made and is called Easy gROOT Gate for SAPPHiRE (EGGS) as it allows direct access to the main functions and capabilities of the data analysis framework ROOT.

In the following paragraphs the requirements for EGGS, the building blocks of EGGS and how the requirements for EGGS are met.



(a) Root document format. The circle represents the head container, folders within this are called branches and are white coloured squares. The leafs are the data files that contain the information. Within a leaf any type of data can be stored. Image made with Tikz.

(b) The main hdf5 file contains a couple of groups which can be nested. The groups contain the datasets. Meta data is kept at every level of the hierarchy, this allows objects within the file to know properties of the full set. If a copy of a dataset is made with only one column changed, then only the changed data is stored, the rest of the data is linked back to the original file. Image used under Creative Commons.

Figure 3.2 – Visualization of the tree style root document format and the hierarchy based format HDF5. The key properties of the data storage formats are discussed.

Requirements of the extended HiSPARC framework

To allow the extended HiSPARC framework to use the tools that have been developed for the data analysis framework ROOT a couple of requirements where formulated to which the new framework should comply. These requirements of the framework are in line with the requirements of the SAPPHiRE framework and the HiSPARC data store;

1. data integrity and continuity
2. access to the complete HiSPARC data store
3. extension to SAPPHiRE

Data integrity requires that no HiSPARC data should be lost when the data is transferred to ROOT, as any loss of data could influence the results that the data might give. The requirement of continuity was formulated to ensure *continuity* between the style of the data and the associated call functions that are used within EGGS and SAPPHiRE.

As the idea of the development of EGGS is that it should be able to handle the big data analysis of HiSPARC, EGGS should have *access to the complete HiSPARC data store*. This requirement should also allow the EGGS framework to do analysis with the whole HiSPARC network.

The requirement that EGGS should be an *extension to SAPPHiRE* ensures that the EGGS is an addition to the SAPPHiRE framework, uses the functions, modules and classes from SAPPHiRE and follows the requirements of a framework, see section 2.4.1 for more information and background on the framework requirements.

Database format style

The HiSPARC framework SAPPHiRE is a Python based framework that uses a hierarchy based document format called *HDF5* to handle and store its data, see figure 3.2b. In the data analysis framework ROOT the data that is analysed is stored in a tree style document format called *root*, see figure 3.2a. The key difference between theses types of datasets is that a root file is nothing more than a nicely organized data container in

Package	Installed by	Purpose
Anaconda 2 - 4.0	bash file	Package handler for Python 2.7
Pip	conda	Python package dependency installer
Matplotlib	conda	Mathematical plotting package
SciPy	conda	Scientific calculations
PyTables	conda	Data handler for HDF5 as used by SAPPHiRE
SAPPHiRE	pip	The basic HiSPARC framework
ROOT 5.34	apt-get	ROOT with Python bindings enabled
libroot-bindings-python-dev	apt-get	The Python bindings for PyROOT
libroot-bindings-python5.34	apt-get	The ROOT 5.34 bindings for PyROOT
Cython 0.21.0	pip	Python to C translator
PyROOT	echo	Coupling of the ROOT bindings to Python
ROOT numpy	pip	The numerical calculations of ROOT
rootpy	pip	Pythonic interface of ROOT

Table 3.1 – The software combination that is needed to run EGGS and allow communication between ROOT and SAPPHiRE. The packages are grouped in colours per dependency line, the last entry of a colour is the package of which the others are dependencies. The row order as shown here is the order in which these programs and packages need to be installed.

which any compatible type of data can be stored, while a HDF5 file is at its heart a meta data set, with the ability to cross reference between parts of the data.

Due to the different data styles that are used by ROOT and SAPPHiRE, data integrity could not be guaranteed when the data would be converted from a HDF5 to a root file. This is due to the meta data that is stored in the HDF5 file, as it would be lost and would need to be recreated when converting to a root file. This recreation of the meta data that the HDF5 file contains is not a problem in its self, but it is an extra set of calculations that would need to be performed.

An initial attempt was made to allow for conversion of data, as the EGGS framework could then be kept very simple. However, the algorithm used for this hit a fatal memory leak when more then 4000 data points where converted and no way around this was found. The source for this problem probably originates from the way memory is assigned in the C-language. See figure 3.3 for a comparison between this conversion method and the method used within EGGS.

The initial failure of mass data conversion, combined with the possibility of data loss, signaled the end for data conversion as a way to work with ROOT within the HiSPARC framework SAPPHiRE. To not violate the requirement of data integrity EGGS should thus not rely on data conversion when working with ROOT.

The roots of EGGS

The back end of EGGS is a combination of packages that allow the communication of data between the SAPPHiRE framework and ROOT. See table 3.1 for a list of the software and packages used.

The packages from table 3.1 are grouped to the main package that requires its installation. The gray packages are required to have a well functioning environment for Python code with the Spyder graphical user interface enabled. The purple packages are dependencies of the SAPPHiRE framework and allow the handling of calculations on and the plotting of HiSPARC data. The brown package is an installation of ROOT with the Python bindings enabled. The green packages are required to allow interactions between the Python and ROOT bindings. The cyan packages are packages that rebind the ROOT Python bindings in a more Pythonic way.

Data flow within EGGS

The back end of EGGS allows a user to write any analysis program in the Python scripting language while working with both the functionality of ROOT and SAPPHiRE. This can be done due to the way data flows through EGGS.

While working in EGGS, data is stored in the hierarchy document format *HDF5*. For every station of which data is used a group is made with the naming style used by SAPPHiRE. This station group contains the original data as downloaded by the server and a sub-group. In this sub-group any analysed or transformed data is stored. This style is chosen as it allows integrity checks on the handled data with respect to the original data.

While analysing data within EGGS, data is handled by the Python package PyTables and its array functionality. The use of this package, combined with the use of *HDF5*, allows fast data handling due to smart data load and store algorithms. By working with the PyTable array when analysing the data the ROOT library bindings do not require the data to be transformed into a different format for the analysis.

The data flows within EGGS from an *HDF5* file, the stored data, to an array that is managed by the PyTables package. This array is then used as input for the analysis done by ROOT. The results of the analysis are then stored in a subgroup of the *HDF5* file. Data format conversions within EGGS are thus redundant and *data integrity* is ensured.

In figure 3.3 an example of the difference between the initial conversion method and the EGGS method is shown. From this figure it is immediately clear that the conversion method is not able to represent the data accurately and that the EGGS method is capable to show the properties of the data.

The *extended* HiSPARC framework

The back end of EGGS does not only ensure the *data integrity* and *continuity* that is required, it also allows the user *access to the complete HiSPARC data store* and makes EGGS an *extension to SAPPHiRE*. Because EGGS and SAPPHiRE are both Python scripting languages and as EGGS adds the use of ROOT to the vast array of capabilities SAPPHiRE has, it is an *extension to SAPPHiRE*. The *access to the complete HiSPARC data store* can be achieved via the API that is implemented within the SAPPHiRE framework.

3.1.3 The incubator for EGGS

During the process of figuring out what was needed to let ROOT and SAPPHiRE communicate, it became apparent that the vast array of packages needed for EGGS, see 3.1, would make the installation of EGGS difficult and prone to errors. Combined with the fact that the installation of SAPPHiRE is difficult, this resulted in the development of a `bash`-script that installs EGGS and all its dependencies.

This quick installation works and has been tested for Ubuntu 12.04 and up systems and will most likely be expanded to include installation scripts for MAC OS[9].

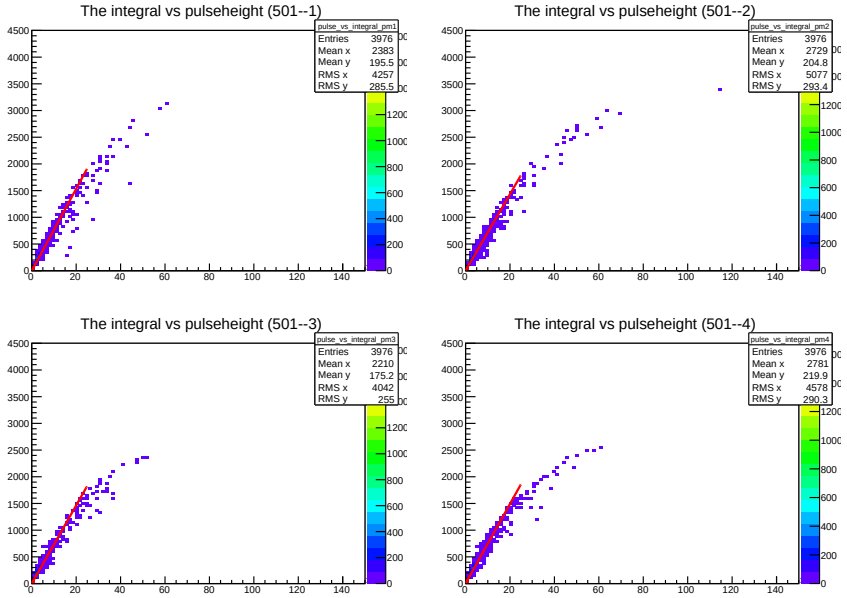
The `bash`-script is initiated by calling `bash eggs_installer.sh` in a terminal opened at the location of the script. During the installation of EGGS the computer needs to restart twice. After these restarts the script has to be manually activated again, a tracker file will keep track of the stage of the installation.

The `bash`-script that installs EGGS can be found in appendix A.1.

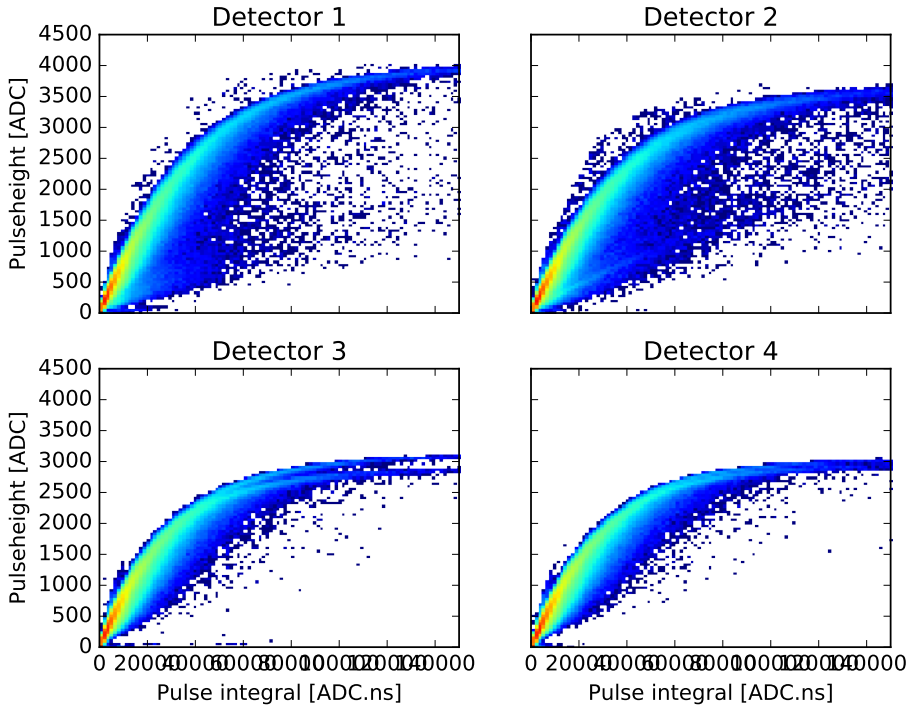
[7] The axes of both the figures do not contain units or readable ticks, the reason for this is that the code has not yet been developed[8]. The only purpose of these plots is to show the difference between the conversion and the EGGS method of handling data.

[8] The code for the conversion method was abandoned as the methodology was flawed. The code for the EGGS method has not yet been developed as this curve will be fitted with ROOT by two lines joined by a circle element and the visualisation will then also be handled by ROOT instead of the Python package MatPlotLib.

[9] Implementation for Windows is at this stage unlikely due to different scripting languages required.



(a) The pulse height is plotted against the pulse integral for data obtained via the conversion method. A line is fitted and the fit results indicate that this is a reasonable fit. In every subplot 4000 data points are shown.



(b) The pulse height is plotted against the pulse integral, graphic made with EGGS. No line is fitted as the code for this has not been implemented yet. The non-linearity of the curve is apparent. In every subplot 1.25×10^11 data points are shown ($\approx$ 1 year of data).

Figure 3.3 – For both the conversion method and the EGGS method the pulse height is plotted against the pulse integral[77] to visualize the photomultiplier tube saturation curve, see section 3.2.1. In the plot with the conversion method, figure 3.3a, the photomuliplier tube seems to have linear behaviour. Due to a massive increase of the number of data points that can be handled with the EGGS method, figure 3.3b, it can be seen that this behaviour is not linear.

3.2 Data analysis with EGGS

In this section the use of the HiSPARC event summary data in the light of big data analysis is discussed. In section 3.2.1 the progress on the initial data calibration functionality of EGGS is elaborated upon.

HiSPARC event summary data

Over the years the HiSPARC data store did not only acquire massive amounts of data, it also gained the functionality to summarize an event that is measured by a station, as explained in section 2.2.2. This functionality was added to reduce the amount of data that needs to be analysed when studying statistical effects of extensive air showers.

The event summary data structure is shown in table 2.1. With the event summary data any event can be reconstructed as the key properties of an event are stored. These properties are obtained by fitting a convolved landau function on raw event data, see [de Vries, 2012, Ch. 4 & 5] for more information on the exact procedure.

3.2.1 Initial data calibration

The extended HiSPARC framework EGGS is in the long run intended to be the big data analysis framework for HiSPARC. To allow for correct data analysis it is of vital importance that the data is calibrated. This calibration is necessary as the HiSPARC detectors are built by high school students and the quality of the build can vary even within a station, among others due to the learning curve of the students during the build of a detector.

The calibration of the data that is used within an analysis should not only be good, it should also be fast. For this reason functionality of EGGS should include these calibrations in an optimized way. The calibration should also be done in such a way that the calibration needs to be done only at the first start up of the analysis program.

In the following sections the initial data calibration will be discussed. First the conversion of the data to physical quantities, then the pulse height characteristic calibration and finally the photomultiplier tube saturation curve calibration will be discussed.

The properties of EGGS as discussed in the following paragraphs can be compared to the EGGS framework code as given in appendix B. The code is the most recent bug free version of EGGS and includes functions for conversion to physical units, a fitting procedure to obtain pulse height characteristics and an initial plotting function for the photomultiplier tube saturation curve.

Pulse height characteristics

The pulse height spectrum or histogram is initially used to check how well the behaviour of the measured data is. From this spectrum the type and number of particles within a shower can then later be determined and therefore it is crucial that the theoretical and expected behaviour are alike.

The pulse height spectrum consists of two major components, the electron/photon component and the muon component. The electron and photon component is the initial peak within the spectrum and is easily affected by effects other than extensive air showers. The interesting part of the spectrum lies thus with the muon component. The spectrum of the photon component can be described by an exponential decay function and the muonic component is described by a landau distribution. More background on the use of the pulse height spectrum in data analysis can be found in [Fokkema, 2012, p. 49-51].

The extended HiSPARC framework EGGS contains a fitting procedure that uses rough estimates of the distribution to make an initial fit of components to the spectrum.

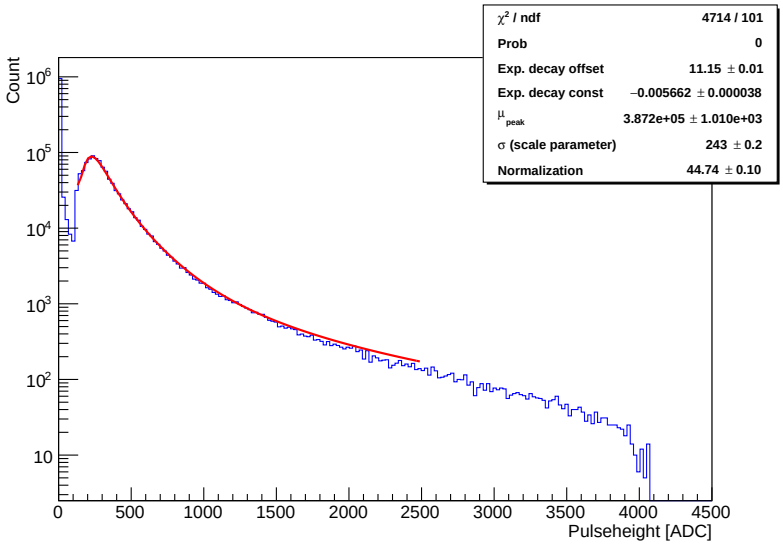


Figure 3.4 – Pulse height histogram that is fitted with an exponential and a landau function. The jump in the histogram around 120 ADC is due to the trigger matrix used by the HiSPARC 'Red Box'. The data shown is from detector 1 of station 501 (Nikhef) for the whole of 2015.

These initial fits are then used to fit the combined spectrum. An example of a combined fit is shown in figure 3.4 for station 501.

The function that is fitted to the pulse height histogram can later be used in determining cut values for the pulse height spectrum and in determining the number of particles within the detector.

Photomultiplier tube saturation

The photomultipliers used by the HiSPARC project have been improved over the years, see section 2.1.2. The early photomultipliers did not have linear behaviour when a large number of particles hit the scintillator plate at the same time.

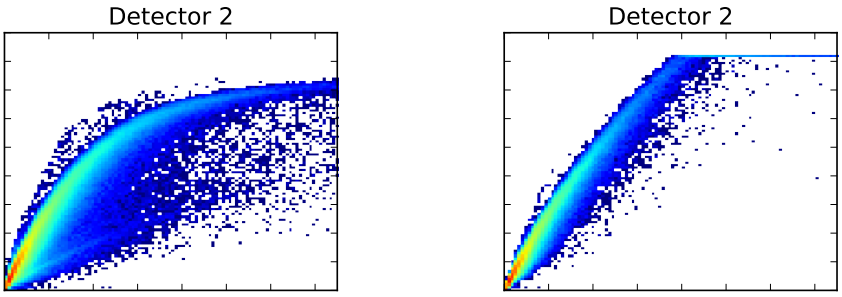
This non-linear behaviour of the photomultiplier tube can be seen in the pulse height histogram, see 3.4. The behaviour of the histogram is closely matched by the fitted curve around the top. However, at a higher ADC count the curve does not fit the spectrum very well.

A different way of looking at the linearity of the photomulitplier tube is by looking at the so called photomultiplier saturation curve. In a photomultiplier saturation curve the pulse height is plotted against the pulse integral for a large number of data points. If the behaviour of the photomultiplier tube is linear then linear behaviour is expected to be visible. See section 2.1.2 for more information on photomultiplier tubes.

In figure 3.5 the photomultiplier saturation curve is shown for two stations. The difference between the linear behaviour, figure 3.5b, and the non-linear behaviour, figure 3.5a, can be seen clearly.

The non-linear behaviour of a photomultiplier tube can be compensated by fitting the data set with two lines joined by a circle element. This fitted function can then be used to compensate the data for the non-linear behaviour of the photomultiplier tube.

The code for the fitting procedure of the photomultiplier saturation curve has not been implemented yet. This curve will be fitted with ROOT and the visualisation will then also be handled by ROOT instead of the Python package MatPlotLib.



(a) Photomultiplier saturation curve for detector 2 of station 501 (Nikhef) is shown with data from 2015. The maximum voltage of the photomultiplier tube is never reached. A wide spread in the data is observed.

(b) Photomultiplier saturation curve for detector 2 of station 14001 (Birmingham University (UK)) is shown with data from 2015. The maximum voltage of the photomultiplier tube is reached and the signal is saturated.

Figure 3.5 – Photomultiplier saturation curve comparison between two stations. No quantities on the axis are shown, but the same range is used in both plots.

Conversion to physical quantities

The data that is stored within the HiSPARC project is never converted to physical quantities as in this way data integrity is maintained. When doing data analysis it is, however, best to work with physical quantities. Working with physical quantities does not only ensure that the physical concepts of the measurement are not forgotten, but it is also a safeguard against interpretation errors[10].

The way the data is stored on the HiSPARC server depends on the observable that is measured. The unit of time on the server is in nanoseconds as determined by the 'Red Box'. The unit for the voltage that the photomultiplier measures is Analog to Digital Converter (ADC) counts. The ADC count is not a physical quantity, but it can be converted into one by taking the calibration of the analog to digital converter into account.

The conversion formula for x ADC counts to units of mV is given by

$$V(x) = -0.57x + 113.$$

This conversion formula has been implemented into a conversion function within the EGGS code[11], see lines 143 to 164 in appendix B.

[10] Due to a couple of different interpretations of the ADC counts this calibration was built in. The other calibration functions should be adapted to match this behaviour.

[11] The calibration of the analog to digital converter can change from station to station and depends on a number of factors. The conversion to physical units that is given is the average conversion formula. The EGGS code could be improved by implementing a station dependent conversion formula.

Chapter 4

Conclusion

To aid the processing of data an extension of the HiSPARC data analysis framework SAPPHiRE has been developed. This extension is called Easy gROOT Gate for SAPPHiRE or EGGS for short. This new framework allows the easy use of ROOT's statistical tools on the HiSPARC data and therefore adds the possibility of big data analysis to the HiSPARC project[1]. The back-end of EGGS is finished, but it will need more functionality to reach its full potential, see Chapter 3.

As EGGS requires a combination of ROOT 5.34, Python 2.7 and other additional programs and packages to work together in a specific way, an installation script has been made to allow for easy installation, see section 3.1.3. This bash script installs all the necessary components and packages for EGGS in the correct order to function properly. This bash script has been tested and works for any computer with an Ubuntu 12.04 or higher operating system, but requires a manual reactivation after the required restarts of the computer. The bash script has not been tested on Windows or Mac OS, further development is therefore necessary.

Due to EGGS the HiSPARC project is now one step closer to the detection of spallation processes and the cross station calibration that is needed to achieve this.

4.1 Outlook

When considering the next steps that can be taken within HiSPARC it is important to consider on the one hand the research conducted with HiSPARC, but it is also equally important to look at the outreach and educational side of HiSPARC. Therefore both these aspects of the HiSPARC project are considered in the following paragraphs combined with an outlook on the further development of the initial data calibration functionality.

4.1.1 On the research of HiSPARC

With the new big data framework that is now under development, a whole new range of research can be done with the HiSPARC array aside from further research into spallation. The majority of these new fields of research are now possible due to the increase in the statistics of measurements on cosmic rays and their extensive air showers.

A very interesting new direction for the HiSPARC array could be the *increased statistical analysis of Ultra-High Energy Cosmic Rays*. This would also directly allow further research into the GZK limit and its still disputed existence. However, HiSPARC's library of simulated cosmic rays would have to be extended in order not to be a limiting factor when determining the energy of the cosmic rays.

[1] The limiting factor to the calculations on a laptop would be storage space, not speed.

Further research into long term fluctuations of cosmic rays would now also be possible as years of data can easily be handled and analysed through EGGS. One particularly interesting field of research would be into the *long term fluctuations of the earth's magnetic field*, as all but the Ultra-High Energy Cosmic Rays are affected by it.

Research into the status of a HiSPARC station should now also be possible as EGGS would allow for analysis on the decline of a station over time. This research could be used to *predict when a station needs an upgrade or service* so that the HiSPARC network doesn't influence its own data due to bad or broken instrumentation.

To allow these new possible research fields to flourish, *the baseline functionality of EGGS should be extended*. Not only should EGGS have functions for the basic photon and electron cut of the pulse height histogram, it should also have functions for analyzing the PMT-saturation curves and allow for compensation of these spectra due to the saturation of the PMT. Other extensions of EGGS functionality could be the analysis and compensation for the weather and seasonal flux in the detection of extensive air showers.

Further programming can also be done by extending the functionality of the EGGS installation script to also include the installation of EGGS on Mac OS and possibly even Windows. Another possible extension of this script would be to allow for more customization. The user friendliness could be increased by including automatic continuation after the required restart of the computer.

4.1.2 On education within HiSPARC

In recent years the HiSPARC project has grown and expanded beyond the borders of the Netherlands, see section 2.3. The recent growth of the United Kingdom part of the network is clear evidence of this. However, the recent expansion of the HiSPARC network abroad has not yet been followed up with a similar follow up of the translation of the HiSPARC educational material. To secure the future of the HiSPARC network abroad more headway should be made with this.

While the HiSPARC network is expanding, the uptime of some clusters of the network has gone down dramatically, an example of this is the Groningen cluster. To increase the uptime of these clusters renewed interest at the high schools and universities in the clusters should be sparked. This could be done by visiting the high schools and giving lectures on the advancements on the HiSPARC project.

The documentation off the SAPPHiRE framework is limited, to increase the learning curve of new students and researchers joining the HiSPARC project tutorials on SAPPHiRE could be made.

4.1.3 On the calibration functionality of EGGS

The extent of the initial data calibration functionality of EGGS is rather limited. So further data calibration should be implemented within the program in the future.

The two main fields of calibration will be discussed in the following paragraphs. First the environmental calibration will be discussed, followed by the calibration on time effects.

Environmental calibration

The HiSPARC stations are subject to the environment they stand in, as this environment is not a laboratory but the roof of a high school or university building there is no control over this environment. The detectors are shielded from the full force of nature by the skibox they lie in, but there are still some effects of the weather. This is no problem when only one detector is considered, but when the weather is radically different between two stations, the measurements between the stations can not be compared.

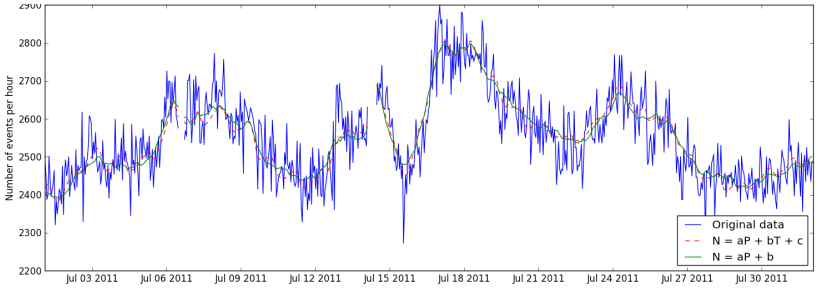


Figure 4.1 – The dependancy of the number of events measured agains the weather. The blue line shows the event rate of station 501 for July 2011. A linear model that takes the atmospheric pressure into account is fitted and shown in red. A model that also includes the temperature is shown in green. Image from [de Vries, 2012, p. 54].

The effect of the weather on the data gathered by a HiSPARC station has been studied in detail in [de Vries, 2012, Ch. 6 & 7]. Theoretical models were made for the dependancy of the number of events of weather effects. The models that were developed within that thesis where all based on linear behaviour between the measurements of an event and the weather. An example of the accuracy of such a model is shown in figure 4.1 and is a clear example that this dependancy can be predicted and thus compensated for accurately.

The implementation of a correction on the data due to the weather should allow the comparison between two stations even though the weather is different. The effects that should be taken into account when developing a model for the number of events will be discussed shortly, see [de Vries, 2012, Ch. 6 & 7] for a more in depth study on these effects.

In the first place the model should include the atmospheric pressure as an increase in pressure will result in an increase of matter through which secundary particles have to travel. For with increased pressure the expected number of events per hour goes down.

The number of events is also expected to be affected due to a change in temperature, the efficiency of the photomultiplier tube and the scintillator plate are temperature dependent. The effect of this dependancy is however expected to be small.

According to the study done by [de Vries, 2012] the relative humidity and the solar radiation should not have an effect on the number of measured events. For this analysis data of a month was used.

Calibration on the time dependancy

While the earth rotates around its axis and the sun the view of the sky by a HiSPARC changes as well throughout the year. As the view of a detector changes over time so do the measurements of that detector change over time. Depending on the analysis done with the HiSPARC data, these effects of time might have an influence on the outcome of the analysis. It is thus important to know how the measurements of the HiSPARC data depends on the time.

The first effect caused by the passing of time is the day and night cycle of the earth. During the day the detector might have detected more events due to low energy cosmic rays that originate from the sun. The calibration of this effect should be done after the environmental calibration as the measurement of the number of events per time is dependent on the temperature of the setup.

As the earth rotates around the sun, the distance between the earth and the sun changes. When the earth is further away from the sun the number of solar cosmic rays is expected to be less due to the smaller angular size of the earth, this might cause fewer events to be measured. Another effect of the greater distance between the earth and the sun is that the local effect of the sun's gravitational field is different. This disturbance in

the local gravitational field can cause a change in the number of events measured. The calibration of this effect will first require further research into the mechanisms behind it.

A third time dependancy of the measured number of events is due to the decay of a HiSPARC detector. The continuous exposure to the elements of the experimental setup of a HiSPARC detector can change the measurements of a detector over time. This effect has not yet been studied and calibration might thus be necessary.

4.1.4 A personal note on further research

Within my bachelor research project I have worked on the development of a big data framework. This development has seen its fair share of ups and downs and has made me realize something about the most expensive part of research: *you only have a limited amount of time*.

My advise to future bachelor students: *do not work on the development of a framework* as this might not give you the satisfaction of reaching your goals. If you want to do your bachelor research within the HiSPARC project, I recommend you work on the extension of the simulation library or the analysis of a single phenomenon. However, if you like a challenge, then working on the development of the HiSPARC framework may well suit you fine.

Appendix A

Bash installation script

```
#! /bin/sh

################################################################################
#
#~~~~~~~Created by Laurens Stoop
#   This software is made under the GNU General Public License, version 3 (GPL-3.0)
#
#########################################################################################################
#
#~~~~~~~Execute this script as
#     $ bash eggs_installer.sh
#
#   NB: do NOT execute as sudo!
#
################################################################################




################################################################################
#
####### FUNCTION DEFINITIONS
#
################################################################################



# Start up function
function start_up ()
{
  # Start with a nice clear screen
  clear

  # Enable firewall cuz..
  ufw enable

  # Tell what we do
  echo "THIS IS THE INSTALLATION OF EGGS"

}


# Reboot the system, some errors can occur if this is not done
function reboot ()
{
  echo "~~~~~~~ REBOOT THE SYSTEM"
  while true; do
    echo "~~~~~~~ Do you wish to reboot now? [y|n]"
    read -sp "" answer_reboot
    case $answer_reboot in
      [Yy]* )   # If the user want to reboot we complete stage
        echo "####### REBOOT NOW";
        sudo reboot -h now;
        break;;
      [Nn]* )   # If the user doesn't want to reboot we push him to do so
        echo "~~~~~~~ Restart before you continue, please!";
        exit;;
      * )      # If the user is crap we call him a badger...
        echo "~~~~~~~ PLease anwer yes or no! Badger!";;
    esac
  done
}



# This function checks for updates and cleans up the system
function preperation ()
{
  echo "####### STARTING STAGE 0: PREPERATION"

  # Get some updates en clear old update files away
  echo "~~~~~~~ UPDATE IN PROGRESS"
  read -sp "~~~~~~~ Press enter to start the update" placeholder
    sudo apt-get -qqy update

  # Clean up shite
  echo "~~~~~~~ REMOVE OLD UPDATE FILES AND CLEAN UP SHIT"
```

```
  while true; do
    echo "~~~~~~~ Do you wish to clean up? (this is higly recommended!) [y|n]"
    read -sp "" answer_preperation
    case $answer_preperation in
      [Yy]* )   # If the user wants to clean up we do so
        sudo apt-get -qqy autoremove;
        sudo apt-get -qqy autoclean;
        # Let the checkfile be increased
        echo "Python" >> eggs_installation_tracker
        break;;
      [Nn]* )   # If the user wants to be filty we let it be...
        echo "~~~~~~~ Ok, but we can't garantee that shite will work";
        # Let the checkfile be increased
        echo "Python" >> eggs_installation_tracker
        break;;
      * )      # If the user is incompetent we call him a squid...
        echo "~~~~~~~ PLease anwer yes or no! Squid!";;
    esac
  done
}


# This functions reads the checklist file and returns the status under the variable stage
function read_file ()
{
  # Read the tracker file and check the stage
  filename="eggs_installation_tracker"
  while read -r LINE
    do
      # We keep overwriting the variable stage so that we have the last input
      stage="$LINE"
  done < "$filename"
}


# This function checks if an arrays contains stuff you want
function array_contains ()
{
  local seeking=$1; shift
  local in=1
  for element; do
    if [[ $element == $seeking ]]; then
      in=0
      break
    fi
  done
  return $in
} # use: array_contains ..stuff_you_seek.. ..array..


# Redundant function to check if the system is rebooted, and promts the user to reboot.
function reboot_check ()
{
  # Let them know what they have done
  echo "-------  SYSTEM_UPDATE_COMPLETED"
  echo "~~~~~~~ Have you rebooted your computer? [y|n]"
  read -sp "" answer_reboot_check
  case $answer_reboot_check in
    [Yy]* )
      check="good";;
    [Nn]* )   # If the user doesn't want to reboot we push him to do so
      echo "~~~~~~~ We asked nicely you duck! ";
      reboot;
      exit;;
    * )      # If the user is crap we call him a Panda...
      echo "~~~~~~~ You do not answer correctly... Panda!";
      reboot;;
  esac

}


# The installation function of Python 2.7
function python_inst ()
{

# Let them know what they have done
  echo "-------  SYSTEM_UPDATE_COMPLETED"
  echo "#######  STARTING STAGE 1: PYTHON"

  echo "~~~~~~~ DOWNLOADING ANACONDA"
  # Ask to continue and install Anaconda
  read -sp "~~~~~~~ Press enter to start the download" placeholder

  # Anaconda installation for 64 bit systems
  if [ `getconf LONG_BIT` = "64" ]; then
    # Download the bash file for Anaconda (takes approx 10 min)
    wget -c https://3230d63b5fc54e62148e-c95ac804525aac4b6dba79b00b39d1d3.ssl.cf1.rackcdn.com/Anaconda2-4.0.0-Linux-x86_64.sh --
       progress=bar:force 2>&1 | tail -f -n +6

    # Ask to continue and install Anaconda
    echo "~~~~~~~ INSTALLATION OF ANACONDA"
    echo "~~~~~~~ Install Anaconda in a folder in your HOME directory, this can be any name"
    echo "~~~~~~~ Say YES (non default) when prompted to create a path in your .bashrc"
    read -sp "~~~~~~~ Press enter to start the installation" placeholder
    bash Anaconda2-4.0.0-Linux-x86_64.sh

  # Anaconda installation for 32 bit systems
  elif [ `getconf LONG_BIT` = "32" ]; then
    # Download the bash file for Anaconda (takes approx 10 min)
    wget https://3230d63b5fc54e62148e-c95ac804525aac4b6dba79b00b39d1d3.ssl.cf1.rackcdn.com/Anaconda2-4.0.0-Linux-x86.sh --
       progress=bar:force 2>&1 | tail -f -n +6
    bash Anaconda2-4.0.0-Linux-x86.sh
```

```
    # Ask to continue and install Anaconda
    echo "~~~~~~~ INSTALLATION OF ANACONDA"
    echo "~~~~~~~ Install Anaconda in a folder in your HOME directory, this can be any name"
    echo "~~~~~~~ Say YES (non default) when prompted to create a path in your .bashrc"
    read -sp "~~~~~~~ Press enter to start the installation" placeholder
    bash Anaconda2-4.0.0-Linux-x86.sh

  # Anaconda installation for other .. bit systems
  else
    echo "You old.. You might want to consider doing this on a different system... "
  fi

  # Reload the .bashrc to activate the path to python
  source ~/.bashrc

  # Now Anaconda will be updated and the correct packages installed
  echo "~~~~~~~ REFRESHING ANACONDA"
  conda update conda
  echo "~~~~~~~ Now some extra packages are installed"
  conda install matplotlib
  conda install scipy
  conda install pytables
  echo "~~~~~~~ All packages will be updated "
  conda update --all
  conda clean --all


  # Let the checkfile be increased
  echo "SAPPHiRE" >> eggs_installation_tracker

}



# The installation function for SAPPHiRE
function sapphire_inst ()
{

  # Let them know what they have done
  echo "~~~~~~~ ANACONDA INSTALLATION COMPLETED"
  echo "####### STARTING STAGE 2: SAPPHIRE"

  # Install SAPPHiRE
  read -sp "~~~~~~~ Press enter to start to install SAPPHiRE" placeholder
  pip install hisparc-sapphire

  # Let the checkfile be increased
  echo "ROOT" >> eggs_installation_tracker

}



# The installation function for ROOT 5.34 with PyROOT enabled
function root_inst ()
{

  # Let them know what they have done
  echo "~~~~~~~ SAPPHIRE INSTALLATION COMPLETED"
  echo "####### STARTING STAGE 3: ROOT"

  minimal_ubuntu_version="1204"
  current_ubuntu_version=$( lsb_release -r | awk '{ print $2 }' | sed 's/[.]//' )

  # ROOT installation for Ubuntu 12.04 and up
  if [ "$current_ubuntu_version" -ge "$minimal_ubuntu_version"  ] ; then

    # Start the installation of ROOT through apt-get
    echo "~~~~~~~ Press enter (default installation) when kereboros asks something"
    read -sp "~~~~~~~ Press enter to start to install ROOT" placeholder
    sudo apt-get install root-system

    # Let the checkfile be increased
    echo "PyROOT" >> eggs_installation_tracker

  # If the OS veriosn < 12.04 then a manual installation is required
  else
    echo "~~~~~~~ You're version of Ubuntu is to old to do this easily."
    echo "~~~~~~~ You could manually install ROOT (usually goes wrong)"
    echo "~~~~~~~ but it is quicker to update your os to any verion of UBUNTU 12.04 and up"
  fi

}



# This function installs PyROOT if ROOT is correctly installed
function pyroot_inst ()
{

  # Let them know what they have done
  echo "~~~~~~~ STAGE 3 COMPLETED"
  echo "####### STARTING STAGE 4: PYROOT"

  echo "~~~~~~~ Checking on ROOT configuration"
  echo "~~~~~~~ Installing libroot bindings "
  read -sp "~~~~~~~ Press enter to go on" placeholder
  sudo apt-get install libroot-bindings-python-dev
  sudo apt-get install libroot-bindings-python5.34
  pip install -u Cython==0.21.1


  # Here we get the array of root-configuration
  var3=$( root-config --features )

  # Do not ask questions about this line
  arr=$var3
```

```
  # Check if the root-config array contains python
  var4=$(array_contains python ${arr[@]} && echo yes || echo no)
  if [ "$var4" = yes ]; then

    echo "____ROOT_is_installed_with_PyROOT_enabled"

    # PyROOT path installation for 64 bit systems
    if [ `getconf LONG_BIT` = "64" ]; then

        echo "~~~~~~~_INSTALLATION_OF_PYROOT_PATH_64BIT_SYSTEM"
        echo "~~~~~~~_Put_the_correct_path's_in_the_.bashrc"
        read -sp "~~~~~~~_Press_enter_to_put_the_paths_nicely" placeholder
        echo "export_PYTHONPATH=\$PYTHONPATH:/usr/lib/x86_64-linux-gnu/root5.34" >> ~/.bashrc
        echo "export_LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/usr/lib/x86_64-linux-gnu/root5.34" >> ~/.bashrc
        echo "export_PYTHONPATH=\$ROOTSYS/lib:\$PYTHONPATH" >> ~/.bashrc
        source ~/.bashrc

    # PyROOT path installation for 32 bit systems
    elif [ `getconf LONG_BIT` = "32" ]; then

        echo "~~~~~~~_INSTALLATION_OF_PYROOT_PATH_32BIT_SYSTEM"
        echo "~~~~~~~_Put_the_correct_path's_in_the_.bashrc"
        read -sp "~~~~~~~_Press_enter_to_put_the_paths_nicely" placeholder
        echo 'export PYTHONPATH=$PYTHONPATH:/usr/lib/i386-linux-gnu/root5.34' >> ~/.bashrc
        echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/i386-linux-gnu/root5.34' >> ~/.bashrc
        echo "export_PYTHONPATH=\$ROOTSYS/lib:\$PYTHONPATH" >> ~/.bashrc
        source ~/.bashrc

    # PyROOT path installation for other .. bit systems
    else
      echo "You_old.._You_might_want_to_consider_doing_this_on_a_different_system..._Go_Tiger!"
    fi

  elif [ "$var4" = no ]; then
    echo "____ROOT_is_not_correctly_installed,_install_it_manually_with_PyROOT_enabled"
  else
    echo "____Something_went_horribly_wrong..._Good_luck!"
  fi

  # Let the checkfile be increased
  echo "rootpy" >> eggs_installation_tracker

}



# This gets rootpy and puts it nicely into play
function rootpy_inst ()
{

  # Let them know what they have done
  echo "-------_STAGE_4_COMPLETED"
  echo "#######_STARTING_STAGE_5:_PYROOT"

  echo "~~~~~~~_INSTALLATION_OF_ROOTPY"
  echo "~~~~~~~_Installs_root_numpy_and_rootpy"
  read -sp "~~~~~~~_Press_enter_to_put_the_install_rootpy" placeholder

  # This installs root_numpy (the numerical side of rootpy)
  pip install root_numpy

  # This installs rootpy
  pip install rootpy

  # Let the checkfile be increased
  echo "done" >> eggs_installation_tracker


}



# This is the roundup function
function good_bye ()
{

  # Let them know what they have done
  echo "-------_STAGE_5_COMPLETED"
  echo "#######_STARTING_STAGE_6:_ROUNDUP"

  # Short cut to the Spyder program with multithread enabled
  echo "alias_eggs='spyder_--multithread_&'">> ~/.bashrc
  echo "alias_groot='root_-l'">> ~/.bashrc
  echo "EGGS_and_GROOT_enabled,_all_systems_are_a_go!_">> ~/.bashrc
  source ~/.bashrc

  echo "~~~~~~~_CLEANING_UP_THE_INSTALLATION_FILES"
  echo "~~~~~~~_Want_to_remove_the_tracker_file?_[y|n]"
  read -sp "_" answer_good_bye
  case $answer_good_bye in
    [Yy]* )
      rm -rf eggs_installation_tracker;
      echo "~~~~~~~_Oke,_it_is_gone";;
    [Nn]* )   # If the user doesn't want to reboot we push him to do so
      echo "~~~~~~~_Oke,_but_it_is_not_anything_usefull";
      exit;;
    * )     # If the user is crap we call him a Panda...
      echo "~~~~~~~_You_do_not_answer_correctly..._Fish!";;
  esac

  reboot;

}



################################################################################
```

```
#
####### MAIN CODE
#
################################################################################
# We start up stuff
start_up;

# If the tracker file doesn't exit, we do Stage I
# (as this is the first initiation of the program)
if [ ! -f eggs_installation_tracker ] ; then

  # Now we do them preperation
  preperation;

  # Now we will reboot the system
  reboot;


# The tracker file exist, we check at what stage it is
else

  # Read the tracker file
  read_file;

  # If we have completed stage 0 and none above, then
  if [ "$stage" = "Python" -o "$stage" = "SAPPHiRE" -o "$check" = "good" ] ; then

    python_inst;

    sapphire_inst;

    reboot;

  # If we have completed stage 2 and none above, then
  elif [ "$stage" = "ROOT" ] ; then

    root_inst;

    pyroot_inst;

    rootpy_inst;

    good_bye;

  # If we have completed stage 3 and none above, but for some reason did not continue
  elif [ "$stage" = "PyROOT" ] ; then

    pyroot_inst;

    rootpy_inst;

    good_bye;

  # If we have completed stage 4 and none above, then
  elif [ "$stage" = "rootpy" ] ; then

    rootpy_inst;

    good_bye;

  # If we have completed stage 5 and none above, then
  elif [ "$stage" = "done" ] ; then

    good_bye;

  else

    echo "-------␣Stage␣is␣unknown..."

  fi

fi
```

Listing A.1 – The Installation Bash file for EGGS

Appendix B

The EGGS framework

```
# -*- coding: utf-8 -*-
#
#############################################################################
#
# Program for analysing HiSPARC data
#
# This software is made under the GNU General Public License, version 3 (GPL-3.0)
#
#############################################################################
"""
==================================
Created on Thu Mar 24 13:17:57 2016

@author: Laurens Stoop
==================================
"""
################################## HEADER ###################################

"""
Import of Packages

"""

print "POKING eggs, NOW IMPORTING PYTHON FEED (~ 18s)"

import sapphire                   # The HiSparc Python Framework
import tables                     # A HDF5 python module that allows to store data
import datetime                   # A package to decode the timeformat of HiSparc data
import matplotlib.pyplot as plt   # Plotting functionality of MatPlotLib
import numpy as np                # This is NumPy
import rootpy                     # Get the pythonesc version of ROOT
import os.path                    # To check if files exist (so you don't do stuff again)
import rootpy.interactive         # Get some option like a wait()
from rootpy.plotting import root2matplotlib
from matplotlib.colors import LogNorm
import ROOT
import array


"""
Getting the data file and setting the variables

"""


# Time between which the data is downloaded (jjjj,mm,dd,[hh])
START = datetime.datetime(2012,01,01)
END = datetime.datetime(2016,01,01)

# Give the list of stations
STATIONS = [501]#,503,1006,1101,3001,13002,14001,20003]

# Do not show the figures
plt.ioff()




################################## STYLE ####################################

def style(name='hisparc', shape='rect', orientation='landscape'):

    # The style we make
    STYLE = rootpy.plotting.Style(name, 'HiSPARC')


    # We do not want borders
    STYLE.SetCanvasBorderMode(0)
    STYLE.SetFrameBorderMode(0)

    # The style of the line
    STYLE.SetHistLineColor(1)
    STYLE.SetHistLineStyle(0)
    STYLE.SetHistLineWidth(2)

    #For the fit/function information
    STYLE.SetOptFit(1111)
    STYLE.SetFitFormat("5.4g")
```

```
    STYLE.SetFuncColor(2)
    STYLE.SetFuncStyle(1)
    STYLE.SetFuncWidth(2)
    STYLE.SetOptStat(0)

    # Change for log plots:
    STYLE.SetOptLogx(0)
    STYLE.SetOptLogy(1)
    STYLE.SetOptLogz(0)

    # Whit background
    STYLE.SetFrameFillColor(0)
    STYLE.SetCanvasColor(0)
    STYLE.SetPadColor(0)
    STYLE.SetStatColor(0)


    return STYLE


################################## FUNC ##################################

print "GETTING_THE_FUNC_OUT"

# This function gets the data into a file with obvious naming structure
def eggs_data_download( data_file_name, station_number=501, begin=datetime.datetime(2016,01,01), final = datetime.datetime
       (2016,01,02) ):

    # Data is downloaded
    if '/s%d' %station_number not in data_file_name:

        # Let them know what we do
        print "\nGetting_event_data_from_station_%d_" % station

        # Now retrieve the event data
        sapphire.esd.download_data(
            data_file_name,      # File (as opened above)
            '/s%d' %station_number,   # Group name (/s..station..)
            station_number,        # Station number
            begin,          # Start data date
            final,            # End data date
            'events',       # Download events (or 'weather')
            True)           # Show progress

        # Let them know what we do
        print "\nGetting_wheater_data_from_station_%d_" % station

        # Now retrieve the wheater data
        sapphire.esd.download_data(
            data_file_name,      # File (as opened above)
            '/s%d' %station_number,   # Group name (/s..station..)
            station_number,        # Station number
            begin,          # Start data date
            final,            # End data date
            'weather',      # Download wheater
            True)           # Show progress

    # If the datafile has the group we do not download them data
    else:
        print "All_data_present_for_station_%d" % station


# Do the dataconversion of the ADC
def eggs_adc_conversion( data_file_name , station_number = 501 ):

    # Get the unconverted data base
    data_set = data_file_name.get_node(
            '/s%d' %station_number,       # From the group (/s..station..)
            'events')           # Get the node with events

    # Get the column we want to convert
    data_converted_ph = 0.57*data_set.col('pulseheights')- 113

    # If no new group present then create one
    if ('/s%d/handled/converted' %station_number) not in data_file_name:

        # Create a group for the converted data
        handled = data_file_name.create_group('/s%d/' % station_number, 'handled', 'Handled_data')

        # Write the new data set to the File
        new_data_set = data_set.copy(handled,'converted') ######################################

        # Modify the pulseheight data column
        new_data_set.modify_column(column=data_converted_ph, colname='pulseheights')




# Load a histogram
def eggs_load_pulseheight( data_file_name, station_number=501, detector=0, number_bins=200, range_start=0., range_end=4500):

    # Get event data
    event_data = data_file_name.get_node(
            '/s%d' %station_number,       # From the group (/s..station..)
            'events')           # Get the node with events

    # Get the pulseheight from all events
    data_ph = event_data.col('pulseheights') # col takes all data from events

    # Create a histogram
    ph_histo = rootpy.plotting.Hist(number_bins, range_start, range_end, drawstyle='hist')

    # Fill it with data
    ph_histo.fill_array(data_ph[:,detector])
```

```
    return ph_histo
190
195
# This function plots the pulse_histograms depending on number of bins and range
def eggs_plot_pulseheight( data_file_name, station_number=501, detector=0, number_bins=200, range_start=0., range_end=4500 ):

    # If the plot exist we skip the plotting
    if os.path.isfile('./img/pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector)):
        # Say if the plot is present
        print "Plot already present for station %d" % station_number
200
    # If there is no plot we make it
    else:

        # Now transform the ROOT histogram to a python figure
205     rootpy.plotting.root2matplotlib.hist(ph_histo)

        # Setting the limits on the axis
        plt.ylim((pow(10,-1),pow(10,7)))
        plt.xlim((range_start, range_end))
210     plt.yscale('log')

        # Setting the plot labels and title
        plt.xlabel("Pulseheight [ADC]")
        plt.ylabel("Counts")
215     plt.title("Pulseheight histogram (log scale) for station (%d)" %station_number)

        # Saving them Pica
        plt.savefig(
            './img/pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector) ,    # Name of the file
220         bbox_inches='tight')                        # Use less whitespace


def eggs_fitplot_pulseheight( ph_histo, station_number, detector=0):
225
        # If the plot exist we skip the plotting
    if os.path.isfile('./img/fitted_pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector)):
        # Say if the plot is present
        print "Fitted plot already present for station %d, detector %d" % (station_number, detector)
230
    # If there is no plot we make it
    else:
        # Set the plot style
        rootpy.plotting.set_style(style('hisparc'))
235
        # We work on a invisible canvas and fit the histogram and plot it
        with rootpy.context.invisible_canvas() as canv:

            # Properties of the canvas
240         canv.axes(xlimits=[100,4500],ylimits=[pow(10,-1),pow(10,7)], xbins=200)

            # The fitfunction definitions
            ph_fitf_signal = ROOT.TF1( 'ph_fitf_signal', 'landau',120,2500)
            ph_fitf_bkg = ROOT.TF1( 'ph_fitf_bkg', 'expo',0,200)
245         ph_fitf_total = ROOT.TF1('ph_fitf_total','expo(0)+landau(2)',120,2500)

            # THe fitting of the pre-functions
            ph_histo.Fit(ph_fitf_signal,'MQR')
            ph_histo.Fit(ph_fitf_bkg,'MQR+')
250
            # Retrieving the fitparameters for final fit
            ph_par_signal= ph_fitf_signal.GetParameters()
            ph_par_bkg = ph_fitf_bkg.GetParameters()

255         # Making an empty array in which the fitparameters can be loaded
            ph_par_total = array.array( 'd', 5*[0.] )

            # Loading of the fitparameters
            ph_par_total[0], ph_par_total[1] = ph_par_bkg[0], ph_par_bkg[1]
260         ph_par_total[2], ph_par_total[3], ph_par_total[4] = ph_par_signal[0], ph_par_signal[1], ph_par_signal[2]

            # Set the fitparameters and their names for final fit
            ph_fitf_total.SetParameters( ph_par_total )
            ph_fitf_total.SetParName(0,'Exp. decay offset')
265         ph_fitf_total.SetParName(1,'Exp. decay const')
            ph_fitf_total.SetParName(2,'#mu_{peak}')
            ph_fitf_total.SetParName(3,'#sigma (scale parameter)')
            ph_fitf_total.SetParName(4,'Normalization')

270         # Fit the full function
            ph_histo.Fit(ph_fitf_total,'MR')

            # Get the parameters for later use
            ph_par = ph_fitf_total.GetParameters()
275
            # set visual attributes
            ph_histo.linecolor = 'blue'
            ph_histo.markercolor = 'blue'
            ph_histo.xaxis.SetTitle("Pulseheight [ADC]")
280         ph_histo.yaxis.SetTitle("Count")

            # Draw the histo gram and the fitfunction
            ph_histo.Draw()
            ph_fitf_total.Draw('same hisparc')
285
            # Save the image for thesis
            canv.SaveAs('./img/fitted_pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector))

            # Return the found fitparameters for the total fit function
290         return [ph_par]



295
```

```
300
305
# The event discription for the correct table column names
class event_description(tables.IsDescription):

    event_id = tables.UInt32Col(pos=0)
310 timestamp = tables.Time32Col(pos=1)
    nanoseconds = tables.UInt32Col(pos=2)
    ext_timestamp = tables.UInt64Col(pos=3)
    pulseheights = tables.Int16Col(pos=4, shape=4)
    integrals = tables.Int32Col(pos=5, shape=4)
315 n1 = tables.Float32Col(pos=6)
    n2 = tables.Float32Col(pos=7)
    n3 = tables.Float32Col(pos=8)
    n4 = tables.Float32Col(pos=9)
    t1 = tables.Float32Col(pos=10)
320 t2 = tables.Float32Col(pos=11)
    t3 = tables.Float32Col(pos=12)
    t4 = tables.Float32Col(pos=13)
    t_trigger = tables.Float32Col(pos=14)

325


def eggs_cut_pulseheight( data_file_name, station_number):

330 # If no pulseheight cut present then create one
    if ('/s%d/handled/pulseheight_cut' %station_number) not in data_file_name:

        # Make a empty table with the correct properties
        data_cut_ph = data_file_name.create_table(
335         '/s%d/handled' % station_number,        # Destination group
            'pulseheight_cut' ,                     # Destination filename
            event_description,      # Format of the table
            "Converted data")                       # Title of the table
    else:
340     data_cut_ph = data_file_name.get_node(
            '/s%d/handled' %station_number,       # From the group (/s..station..)
            'pulseheight_cut')

    # Get the uncut converted data base
345 data_set = data_file_name.get_node(
            '/s%d/handled' %station_number,       # From the group (/s..station..)
            'converted')                            # Get the node with events

    # Check that the destination file is not in read-only mode.
350 data_cut_ph._v_file._check_writable()

    # Shortcut to the column name
    colNames = [colName for colName in data_set.colpathnames]

355 # The destination row is in the new file
    dstRow = data_cut_ph.row

    # The number of rows we have transfered (mostly check up)
    nRows = 0
360
    # For all the rows in the data set we only take the wanted pulseheights.
    for srcRow in data_set:

        # The selection criteria for the transfer of the row
365     if srcRow['pulseheights'][0] >= 0:

            # Link the column names if the row matches the selection criterion
            for colName in colNames:
                dstRow[colName] = srcRow[colName]
370
            # The new filled row is appended to the table
            dstRow.append()

            # We count the number of allowed rows
375         nRows += 1

    # After the selection of the allowed rows we clean the memory
    data_cut_ph.flush()

380 # The function has as output the number of converted rows
    return nRows


385


390


395
def eggs_plot_pmt(data_file_name, station_number):

    # If the plot exist we skip the plotting
    if os.path.isfile('./img/pmt_saturation_s%d.pdf' %station_number):
400     # Say if the plot is present
        print "PMT saturation histogram already present for station %d" % station_number

    # If there is no plot we make it
```

```
    else:
        # Get event data
        event_data = data_file_name.get_node(
            '/s%d' %station,       # From the group (/s..station..)
            'events')           # Get the node with events

        # Get the pulseheight from all events
        data_phs = event_data.col('pulseheights')   # col takes all data from events (this improves the speed)

        # Get the integral from all events
        data_ints = event_data.col('integrals') # col takes all data from events

        # Make a figure so it can be closed
        figure_combo, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex = 'col', sharey = 'row')

        # Setting the plot titles
        ax1.set_title('Detector 1')
        ax2.set_title('Detector 2')
        ax3.set_title('Detector 3')
        ax4.set_title('Detector 4')

        # Setting the plot labels
        ax1.set_ylabel('Pulseheight [ADC]')
        ax3.set_ylabel('Pulseheight [ADC]')
        ax3.set_xlabel('Pulse integral [ADC.ns]')
        ax4.set_xlabel('Pulse integral [ADC.ns]')

        # Now we plot the data of every detector
        for detector in range(0,4):

            # Select the detector data
            data_ph_detector = data_phs[:,detector]
            data_int_detector = data_ints[:,detector]

            # Combine the detector data
            data_combo = np.stack(
                (data_int_detector,      # The pulse integral on y axis
                 data_ph_detector),    # The pulseheight on x axis
                 axis=-1)               # To get the direction correct

            # Initiate a 2D histogram (ROOT style)
            histo_combo_detector = rootpy.plotting.Hist2D(100, 0, 150000, 100, 0, 4500)

            # Fill the Histogram
            histo_combo_detector.fill_array(data_combo)

            # Plot the histogram with logarithmic colors in correct place
            if detector == 0:
                root2matplotlib.hist2d(histo_combo_detector, norm=LogNorm(), axes=ax1)
            elif detector == 1:
                root2matplotlib.hist2d(histo_combo_detector, norm=LogNorm(), axes=ax2)
            elif detector == 2:
                root2matplotlib.hist2d(histo_combo_detector, norm=LogNorm(), axes=ax3)
            elif detector == 3:
                root2matplotlib.hist2d(histo_combo_detector, norm=LogNorm(), axes=ax4)

                # Save the file
                figure_combo.savefig(
                    './img/pmt_saturation_s%d.pdf' %station_number)    # Name of the file

                # Close the figure
                plt.close(figure_combo)

            # Now we go to the next detector
            detector +1









def eggs_clean(data_file_name, station_number):

    # Cleans the pica's
    if os.path.isfile('./img/pmt_saturation_s%d.pdf' %station_number):
        os.remove('./img/pmt_saturation_s%d.pdf' %station_number)

    detector=0
    if os.path.isfile('./img/fitted_pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector)):
        os.remove('./img/fitted_pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector))

    if os.path.isfile('./img/pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector)):
        os.remove('./img/pulseheigt_histogram_%d_detector_%d.pdf' % (station_number, detector))




################################## BODY ###################################

"""
Data acquisition

"""

print "SPARCS ENABLED, TREE ROOTS ACTIVATED, TIMING ADJUSTED"


# Open a data file (automatic close)
with tables.open_file('egg_eggs.h5','a') as data_file:

    # Retrieve for every station the data and plot a pulsehisto
    for station in STATIONS:
```

```
        eggs_clean(data_file, station)

515

        # Getting the data
        eggs_data_download( data_file, station, START, END)

520     # Set the conversion for adc
        eggs_adc_conversion( data_file, station)

        # Create the histogram
        ph_histo = eggs_load_pulseheight( data_file, station, 0, 200, 0., 4500)
525
        # Fit the functions
        eggs_fitplot_pulseheight(ph_histo,station,0)

        # Plot some data
530     eggs_plot_pulseheight( data_file, station, 0, 200, 0., 4500)

        # Plot the PMT curves
        eggs_plot_pmt( data_file, station )

535

        aantal = eggs_cut_pulseheight( data_file, station)

        print('Values that pass the cuts:', aantal)
540



545 print "####### I'm Done! #######"


    ################################## FOOTER ##################################

550 """
    Clean up shit

    """
```

Listing B.1 – Easy gROOT Gate for SAPPHiREcode

Bibliography

[Aaij et al., 2015] Aaij, R. et al. (2015). Observation of $J/\psi p$ Resonances Consistent with Pentaquark States in $\Lambda_b^0 \to J/\psi K^- p$ Decays. *Phys. Rev. Lett.*, 115:072001. arxiv.org/abs/1507.03414.

[Abbasi et al., 2014] Abbasi, R. U. et al. (2014). Indications of Intermediate-Scale Anisotropy of Cosmic Rays with Energy Greater Than 57 EeV in the Northern Sky Measured with the Surface Detector of the Telescope Array Experiment. *Astrophys. J.*, 790:L21. arxiv.org/pdf/1404.5890.pdf.

[Alania et al., 2009] Alania, M., Araya, I. J., Gómez, A. V. C., Huerta, H. M., Flores, A. P., and Knapp, J. (2009). Air Shower Simulations. In Solano Salinas, C. J., Bellido, J., Wahl, D., and Saavedra, O., editors, *American Institute of Physics Conference Series*, volume 1123 of *American Institute of Physics Conference Series*, pages 150–165. adsabs.harvard.edu/abs/2009AIPC.1123..150A.

[Antcheva et al., 2009] Antcheva, I., Ballintijn, M., Bellenot, B., Biskup, M., Brun, R., Buncic, N., Canal, P., Casadei, D., Couet, O., Fine, V., Franco, L., Ganis, G., Gheata, A., Maline, D. G., Goto, M., Iwaszkiewicz, J., Kreshuk, A., Segura, D. M., Maunder, R., Moneta, L., Naumann, A., Offermann, E., Onuchin, V., Panacek, S., Rademakers, F., Russo, P., and Tadel, M. (2009). {ROOT} — a c++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 180(12):2499 – 2512. 40 {YEARS} {OF} CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures.

[Bird et al., 1993] Bird, D. J., Corbató, S. C., Dai, H. Y., Dawson, B. R., Elbert, J. W., Gaisser, T. K., Green, K. D., Huang, M. A., Kieda, D. B., Ko, S., Larsen, C. G., Loh, E. C., Luo, M., Salamon, M. H., Smith, D., Sokolsky, P., Sommers, P., Stanev, T., Tang, J. K. K., Thomas, S. B., and Tilav, S. (1993). Evidence for correlated changes in the spectrum and composition of cosmic rays at extremely high energies. *Phys. Rev. Lett.*, 71:3401–3404. link.aps.org/doi/10.1103/PhysRevLett.71.3401.

[Boyle et al., 2008] Boyle, P. J., Gahbauer, F., Höppner, C., Hörandel, J., Ichimura, M., Müller, D., Wolf, A. R., and TRACER project (2008). Cosmic ray composition at high energies: The TRACER project. *Advances in Space Research*, 42:409–416. adsabs.harvard.edu/abs/2008AdSpR..42..409B.

[Brun and Rademakers, 1997] Brun, R. and Rademakers, F. (1997). {ROOT} — an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1–2):81 – 86. New Computing Techniques in Physics Research V.

[de Laat, 2016] de Laat, A. (2016). Private communication.

[de Laat and van Eijk, 2016] de Laat, A. and van Eijk, B. (2016). Informatie pakket. HiSPARCinformation packet. hisparc.nl/docent-student/lesmateriaal/informatie-pakket/.

[de Vries, 2012] de Vries, L. (2012). Search for correlation between HiSPARCcosmic-ray data and weather measurements. Master's thesis, University of Twente. hisparc.nl/fileadmin/HiSPARC/werk_van_studenten/Loran2012_.pdf.

[Enqvist, 2009] Enqvist, T. (2009). Astroparticle physics lecture notes. University of Oulu. cupp.oulu.fi/timo/opetus/lecture-app2009-1.pdf.

[Fokkema, 2012] Fokkema, D. B. R. A. (2012). *The HiSPARC experiment*. PhD thesis, Universiteit Twente, The Netherlands. doc.utwente.nl/81982/1/thesis_D_Fokkema.pdf.

[George et al., 2009] George, J. S., Lave, K. A., Wiedenbeck, M. E., Binns, W. R., Cummings, A. C., Davis, A. J., de Nolfo, G. A., Hink, P. L., Israel, M. H., Leske, R. A., Mewaldt, R. A., Scott, L. M., Stone, E. C., von Rosenvinge, T. T., and Yanasak, N. E. (2009). Elemental Composition and Energy Spectra of Galactic Cosmic Rays During Solar Cycle 23. *The Astrophysical Journal*, 698:1666–1681. adsabs.harvard.edu/abs/2009ApJ...698.1666G.

[Grupen, 2005] Grupen, C. (2005). *Astroparticle Physics*. adsabs.harvard.edu/abs/2005asph.book.....G.

[Haverhoek, 2006] Haverhoek, J. D. (2006). Ultra-high-energy cosmic-ray extensive air shower simulations using corsika. Master's thesis, Universiteit Leiden. hisparc.nl/fileadmin/HiSPARC/werk_van_studenten/UHECRs-thesis-JDHaverhoek.pdf.

[Henley and Garcia, 2007] Henley, E. M. and Garcia, A. (2007). *Subatomic Physics: Third Edition*. World Scientific Publishing Co, 3 edition. adsabs.harvard.edu/abs/2007suph.book.....H.

[Hillas, 1984] Hillas, A. M. (1984). The Origin of Ultra-High-Energy Cosmic Rays. *Annual review of astronomy and astrophysics*, 22:425–444. adsabs.harvard.edu/abs/1984ARA%26A..22..425H.

[Kahlsruhe Institue of Technology, 2016] Kahlsruhe Institue of Technology (2016). Cosmic ray simulation for kascade. ikp.kit.edu/corsika/.

[Khachatryan and et al., 2014] Khachatryan, V. and et al. (2014). Observation of the rare $B_s^0 \to \mu^+\mu^-$ decay from the combined analysis of CMS and LHCb data. *Nature*, 522(arXiv:1411.4413. CERN-PH-EP-2014-220. CMS-BPH-13-007. LHCB-PAPER-2014-049):68–72. 46 p. http://cds.cern.ch/record/1970675.

[Otto, 2007] Otto, J. (2007). Hisparc pulsgenerator. Master's thesis, Hogeschool van Amsterdam. hisparc.nl/fileadmin/HiSPARC/werk_van_studenten/Afstudeer_verslag_HiSPARC_pulsgenerator_J_Otto_v1.2.pdf.

[Radu et al., 2008] Radu, A., Brancus, I., Broscaru, O., Felea, D., and Mitrica, B. (2008). Mini-arrays of detectors for the study of cosmic-ray air showers. *Romanian Reports in Physics*, 60(1):45–55. rrp.infim.ro/2008_60_1/04-045-055.pdf.

[Schultheiss, 2016] Schultheiss, N. G. (2016). The acceptance of the HiSPARC experiment. arxiv.org/pdf/1601.00849v1.pdf.

[Young and Freedman, 2000] Young, H. D. and Freedman, R. A. (2000). *University physics with modern physics*. Addison-Wesley, 10 edition.

Word of Thanks

During the time I have been working on my bachelor's thesis I have had help from quite a few people and I would like to thank them all for their continuous patience and support, even while I worked on some side-projects for HiSPARC. Some people I would like to thank especially.

First of all I would like to thank Alessandro Grelli for always urging me on and being open to questions. Thanks to you I have always had a clear goal and you kept me motivated even when the first track of development failed.

André Mischke, I would like to thank you for your initial advice when I was choosing a bachelor research project and for providing me with valuable feedback during the course of the project.

I would also like to thank Arne de Laat for his patience and quick replies to emails while I tried to learn how to work with the SAPPHiRE framework. Thanks to your thorough answers this thesis could be just that bit better. I hope your PhD defence will go well.

For their feedback and support during the development of EGGS I would like to thank Niels Schaminée and Monique Dorresteijn. Thanks to your timely help I went on a break when I was stuck.

For their presence I would like to thank all that followed the LaTeX thesis that I set up, your requests and feedback helped me write the basic frame and style of this thesis. I hope you have all been able to finish your thesis in time with your new LaTeX skill.

Last but definitely not least, I would like to thank my fiancée Shawn for being by my side these last seven years, for not minding that I did little cooking or cleaning these past few weeks and for cheering me up when I was frustrated.