GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik

Master's Thesis

Teststrahlanalysen mit Pixelsensoren für den HL-LHC Ausbau

Testbeam measurements with pixel sensors for the ATLAS HL-LHC upgrade

prepared by

Tobias Bisanz

from Linz

at the II. Physikalischen Institut

Thesis number: II.Physik-UniGö-MSc-2014/07

Thesis period: 1st May 2014 until 31st October 2014

First referee: Prof. Dr. Arnulf Quadt

Second referee: Priv.Doz. Dr. Jörn Große-Knetter

Zusammenfassung

Für den Ausbau des LHC werden neue Pixeldetektortechnologien entwickelt, gefertigt und letztendlich erprobt. Für letztere Aufgaben stellen Teststrahlmessungen ein mächtiges Werkzeug dar. Um für zukünftige Detektoren gerüstet zu sein, wurden Veränderung am Rekonstruktionsframework EUTelescope sowie dem Datennahmepaket USBpix vorgenommen. Die Veränderungen betreffen in erster Linie die Sensorgeometrie wie auch die Modullayouts. EUTelescope wurde mit einem Sensor-Geometriemanager versehen, welcher es erlaubt allgemeinere Pixelgeometrien als bisher zu rekonstruieren. USBpix ist nun in der Lage sogenannte Burn-In Adapterkarten mit Multichipmodulen im Teststrahlbetrieb auszulesen.

Stichwörter: Physik, Masterarbeit, Teststrahlrekonstruktion und Analyse, Pixeldetektoren, Pixelgeometrien, ATLAS, HL-LHC, USBpix, EUTelescope

Abstract

For the upcoming LHC upgrades novel pixel detector technologies are being developed, manufactured and ultimately tested and characterised. For detector testing and characterisation, testbeam measurements are a useful tool. To support upcoming sensors, the reconstruction framework EUTelescope, as well as the data acquisition package USBpix, have been modified. Most notably, these changes concern the sensor geometry layouts as well as module layouts. A sensor geometry manager was introduced in EUTelescope, allowing for more general pixel layouts and USBpix now supports the read-out of burn-in adapter cards for multi chip modules in testbeam operation.

Keywords: Physics, Master Thesis, Testbeam Reconstruction and Analysis, Pixel Sensors, Pixel Geometries, ATLAS, HL-LHC, USBpix, EUTelescope

Contents

1. Introduction

1.1. The Standard Model and Physics at the LHC

A brief introduction to the Standard Model of Particle Physics (SM) will be given, as well as a short outlook at the problems which cannot be described without extending it.

1.1.1. Introduction to the Standard Model

The SM is a theory describing interactions between fundamental, subatomic particles, namely electromagnetic, weak and strong interactions between quarks and leptons. These forces are mediated by the so-called gauge bosons (Fig. 1.1).

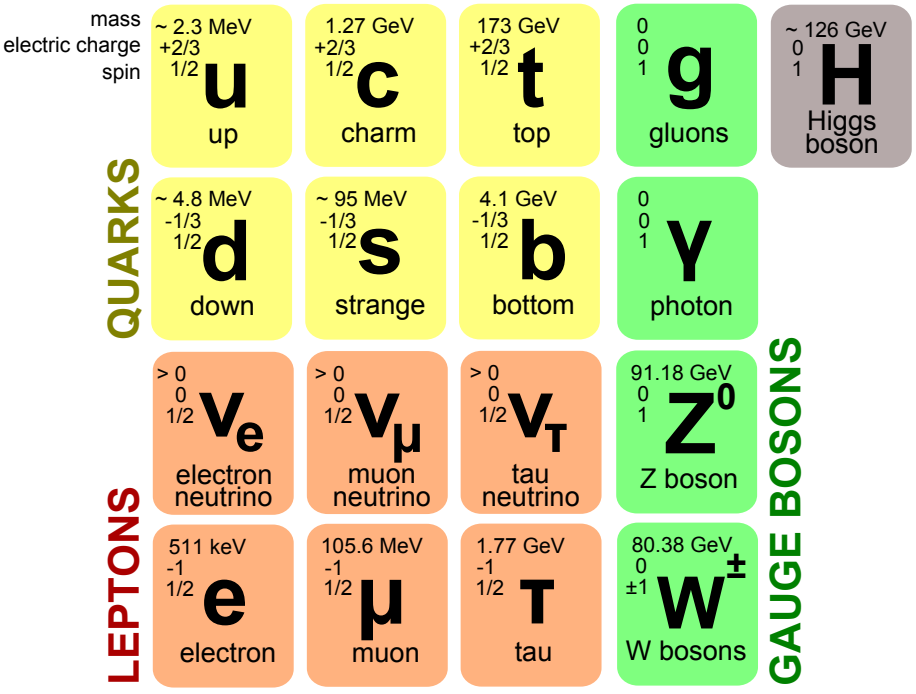

Figure 1.1.: Overview of elementary particles (cf. [1] for detailed values and uncertainties).

From a historical and more mathematical point of view, the electromagnetic and weak force can be explained by a $SU(2) \times U(1)$ gauge group, yielding the electroweak unification. Via this approach, the original weak isospin and weak hypercharge gauge bosons form the $W^{\pm}$ and Z^0 bosons via spontaneous symmetry breaking of the electroweak symmetry. Additionally, a complex scalar Higgs doublet is introduced, which provides a mechanism of mass generation by this spontaneously broken symmetry. Not only does the Higgs mechanism give mass to the massive electroweak gauge bosons, but it also provides mass to the fermions via a Yukawa coupling to them.

The discovery of neutral currents in 1973 [2, 3] at CERN brought the evidence to support the model of electroweak unification, proposed by Abdus Salam, Sheldon Glashow and Steven Weinberg, who were awarded the Nobel Prize in 1976. This triggered the search for the predicted weak gauge bosons. They were discovered at the Super Proton Synchrotron (SPS) by the two experiments, UA1 and UA2 [4–7], at CERN in 1983. This discovery lead to the Nobel Prize for Carlo Rubbia and Simon van der Meer, who were deeply involved in the realization of the SPS just one year later in 1984.

It took roughly 30 years for the discovery of another important ingredient of the electroweak theory, the Higgs boson. It was discovered by the two multi-purpose detectors at the LHC, ATLAS and CMS, in 2012 [8, 9]. Subsequently, Peter Higgs and François Englert received the Nobel Prize for their theoretical work on the Higgs mechanism [10, 11] in 2013.

Another ingredient of the SM is the theory describing strong interactions, called quantum chromodynamics (QCD). Like in the previous theory, there is an underlying gauge group, in this case the SU(3) symmetry group. From an experimental point of view, the history of QCD is closely linked to probing the structure of the nucleons via deep inelastic scattering (DIS). DIS was carried out to test the proposed substructure of nucleons, today known as the quark-model, proposed by Murray Gell-Mann and George Zweig in the mid-60s. An important milestone was the discovery of the J/ψ meson in 1974 by two groups, one lead by Samuel Ting at Brookhaven National Laboratory, the other by Burton Richter at the Stanford Linear Accelerator Center (SLAC)[12, 13]. The J/ψ was the first observed particle which contains the at that time proposed charm quark. Ting and Richter were awarded the Nobel Prize in 1976 for this discovery.

The force mediating gauge bosons of QCD, the massless gluons, were discovered at the Positron-Elektron-Tandem-Ring-Anlage (PETRA) at the Deutsches Elektronen Synchrotron (DESY) by first observing three-jet events in 1979 [14].

In 1964, James Cronin and Val Fitch observed indirect CP violation in the Kaon system [15]. Theoretically this was explained by Makoto Kobayashi and Toshihide Maskawa, who

introduced the third generation of quarks to explain CP violation, by expanding the concepts of the Cabibbo matrix which rotates the weak eigenstates into the mass eigenstates by the Cabibbo angle [16], yielding the CKM matrix[1] [17]. Cronin and Fitch received the Nobel Prize in 1980 for their work, whereas it was awarded in 2008 to Kobayashi and Maskawa.
The predicted third generation quarks were ultimately discovered at Fermilab. The bottom quark was discoved in 1977 by Leon Lederman's group [18], the top quark in 1995 by the two experiments DØ and CDF [19, 20].
This gives a rough overview of the historical development of the SM as we know it today. Despite its success, there are several open questions which the SM cannot explain.

1.1.2. Physics Outlook

One of the open issues with the SM is that it requires 18 parameters as an input (and even more to account for massive neutrinos), which seems very unnatural to many. Moreover, the SM does not provide any mechanism to incorporate gravity.
Observations of neutrino oscillations, first observed by Super-Kamiokande [21], require neutrinos to be massive, which is currently not explained by the SM.
The hierarchy problem in the SM is related to the Higgs boson's mass. It is very light compared to the Planck scale, and this can only be explained by cancellation of correction terms to the mass which requires a very precise, and somewhat artificial, fine tuning. Supersymmetric extensions to the SM are one approach to solve this issue, leading to an underlying cause for this cancellation. Therefore, they are heavily investigated, also at the LHC.
Despite the assumption that during the Big Bang matter and antimatter were created in equal parts, all the matter surrounding us is ordinary matter. This raises the question where all the antimatter has gone. Violation of CP symmetry is necessary to create an imbalance, but while we have experimental evidence for CP-violating weak processes, these are far too rare to explain this issue.
Additionally, cosmological observations tell us that visible matter is only a small fraction of the matter known to us. Viable candidates for dark matter could originate from supersymmetric extensions to the SM.
Given all those open questions, there is no doubt that the SM in its current form cannot be the final theory. Therefore, searches for physics beyond the SM are of great interest and should yield answers to the mentioned problems.

[1] Cabibbo-Kobayashi-Maskawa matrix

1.2. The LHC and the ATLAS Experiment

1.2.1. The Large Hadron Collider

The Large Hadron Collider (LHC) is currently the world's largest and most powerful particle accelerator in terms of circumference and centre of mass energy.
Geographically, it is located at the main site of the European Organization for Nuclear Research, CERN (Conseil Européen pour la Recherche Nucléaire) near Geneva, Switzerland. The LHC is a circular proton-proton collider (Fig. 1.2), as well as an ion collider, with proton-proton collisions being the mode in which the LHC is mostly operated. It has a circumference of 27 km and a nominal centre of mass energy of 14 TeV in proton-proton mode.
The two opposite travelling beams are brought together for collision at four different interaction points, where the various experiments are located. ATLAS (A Toroidal LHC Apparatus) and CMS (Compact Muon Solenoid) are both general purpose particle detectors, primarily searching for new physics as well as probing the already known. ALICE

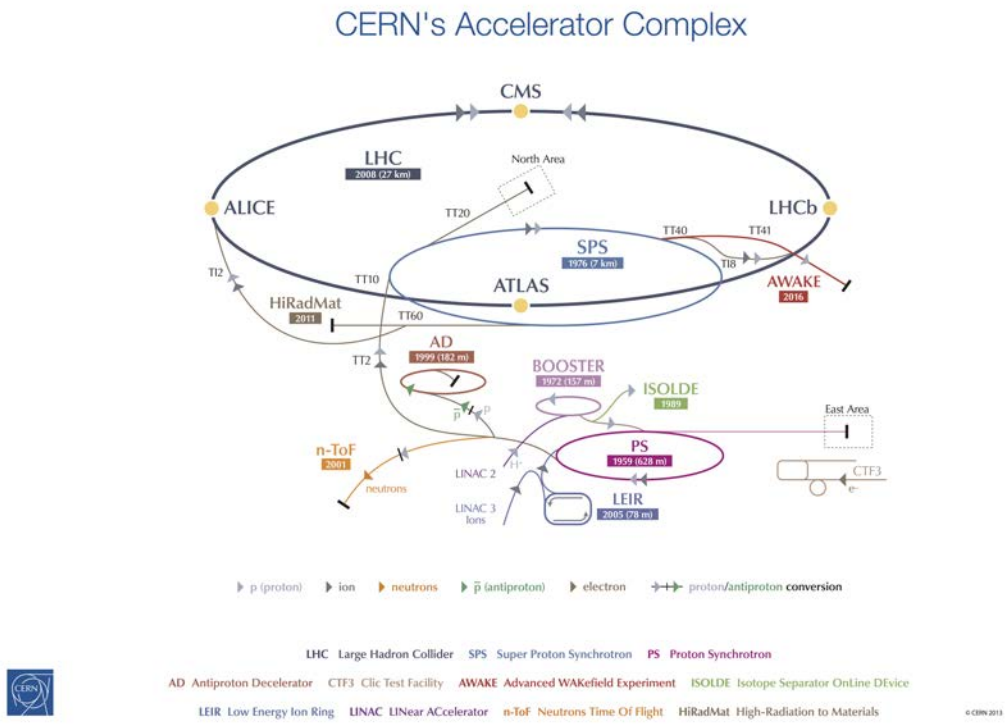

Figure 1.2.: Schematic overview of CERN's accelerator complex. Image courtesy of CERN.

(A Large Ion Collider Experiment) investigates quark-gluon plasma in ion collisions and LHCb (Large Hadron Collider beauty) is dedicated to charge-parity (CP) violation.

1.2.2. The ATLAS Experiment at the LHC

The ATLAS detector is a typical multi-purpose high energy physics detector. It consists of multiple detector systems layered symmetrically around the interaction point in a cylindrical fashion (Fig. 1.3).

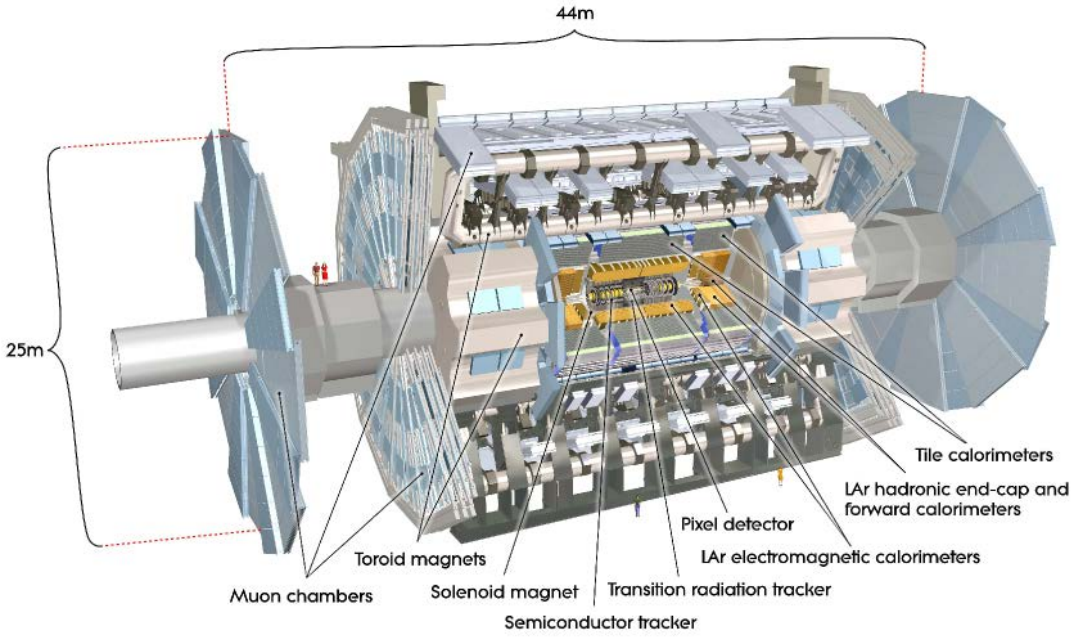

Figure 1.3.: Overview of the ATLAS detector [22].

Additionally, end caps cover the region of high pseudorapidity η[2]. For the purpose of bending particle trajectories for determination of their momentum, two magnet systems are present. The inner detector is immersed in a 2 T magnetic field from a solenoid, whereas three large toroids are arranged around the calorimeters and provide a field between 0.5 and 1 T to bend muon tracks.

Muon System

The outermost layer of the detector is the muon system (Fig. 1.4). It exploits deflection due to the toroidal fields in most of the η-region. In the forwards regions, the additional

[2]Pseudorapidity is defined as $\eta = -\ln(\tan(\theta/2))$, where θ is the angle to the beam axis. It is commonly used in high energy physics. Regions with high $|\eta|$ (i.e. close to the beam pipe) are the so-called forward regions.

magnets in the end caps provide fields for deflection. Due to deflection in this magnetic field, the muon momentum can be measured independently of the Inner Detector.
Muon detection is mostly based on Monitored Drift Tubes (MDTs). Additionally, Cathode Strip Chambers (CSCs) are used in the innermost high η-regions. Their benefit lies in their speed, as they can process the higher track rate present in the inner layers. While MDTs and CSCs are used for precision tracking, Resistive Plate Chambers (RPCs) and Thin Gap Chambers (TGCs) are used by the triggering system and assist in the measurement of the coordinate in the non-bending plane.

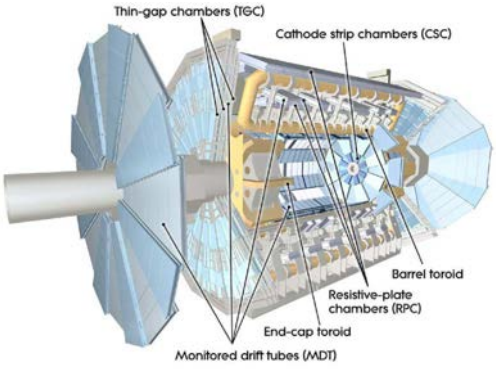

Figure 1.4.: The muon systems present in the ATLAS detector [22].

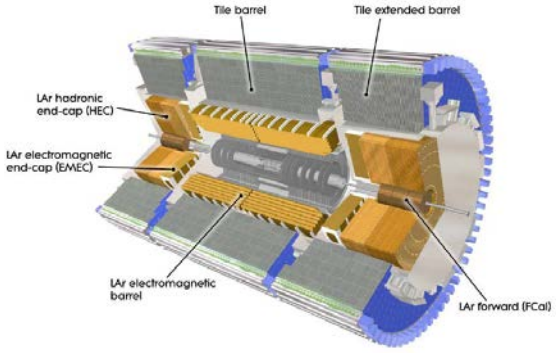

Figure 1.5.: Electromagnetic and hadronic calorimeter [22].

Calorimeters

The calorimeter (Fig. 1.5) is separated into electromagnetic (EM) and hadronic, the EM one being a liquid argon (LAr) calorimeter with lead as an absorber. It consists of a barrel part and two end cap parts at both ends of the detector for high η-coverage. In total at least 22 radiation lengths (X_0) of material are present at any η-value.
While all the parts of the EM calorimeter are in principle the same (lead-LAr sampling calorimeters with slight variations in granularity) this is not the case for the hadronic one. The barrel region is equipped with a sampling calorimeter featuring steel as an absorber and instrumented by active scintillator tiles. This tile calorimeter is then divided into a central barrel region and two extended regions, one on each side. The end caps of the hadronic calorimeter also use LAr as an active material, but copper as an absorber. Given that they also use LAr and are located right behind the EM ones, they share the same cryostat for the liquid argon.
The forward region is also equipped with a LAr calorimeter. It is segmented into three

modules. The first one, primarily for electromagnetic measurements, uses a copper absorber while the other ones, optimised for hadronic measurements, use tungsten.

Inner Detector

The centremost part of the detector is the Inner Detector: amongst other things responsible for track reconstruction, vertex measurements and momentum determination. It can be subdivided into three systems, from the outermost to the inner: the Transition Radiation Tracker (TRT), the Silicon Microstrip Tracker (SCT) and ultimately the Pixel Detector. All of those systems are depicted in Figure 1.6, while Figure 1.7 gives a detailed view of the ATLAS pixel layers.

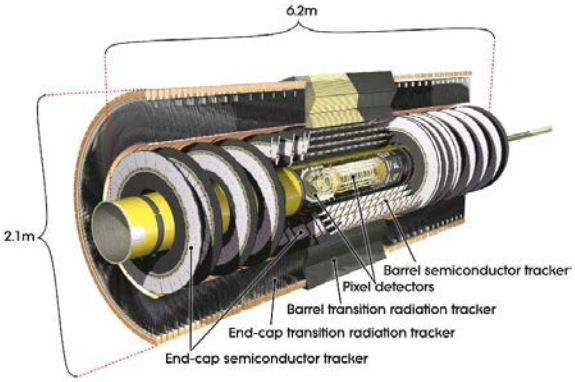

Figure 1.6.: ATLAS Inner Detector [22].

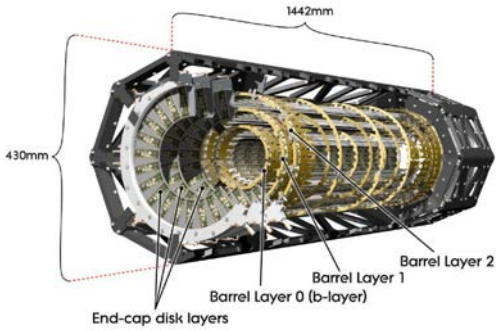

Figure 1.7.: ATLAS Pixel Detector [23].

Straws with a diameter of 4 mm make up the TRT. They provide only spatial information along the R-ϕ coordinate with an intrinsic resolution of 130 µm. The SCT uses double sided strip modules with a strip pitch of 80 µm in the barrel region and end caps. Compared to the roughly 350 000 read-out channels of the TRT, the 63 m^2 area of the SCT features about 6.3 million read-out channels.
Despite being the smallest detector subsystem in the Inner Detector, the Pixel Detector has over 80 million channels and features the highest resolution. Having a nominal pixel size of $50 \times 400\,\mu m^2$, this yields an intrinsic resolution of about 10 µm in R-ϕ and 100 µm in z-direction for the worst case of one hit clusters. In total, three layers of detectors are used in the barrel region and in the end cap region.

Insertable B-Layer (IBL)

The phase-0 upgrade of the ATLAS detector includes the insertion of a fourth pixel layer into the ATLAS pixel detector. This is achieved by removing the old beam pipe and

replacing it by a new one featuring a smaller diameter, thus giving space for an additional layer of pixels. This layer is the so-called Insertable B-Layer [23] and will provide even better vertexing and b-tagging capability.
The schematic layout of IBL with three staves around the new beam pipe is given in Figure 1.8, a render of the inserted IBL sensors is shown in Figure 1.9.

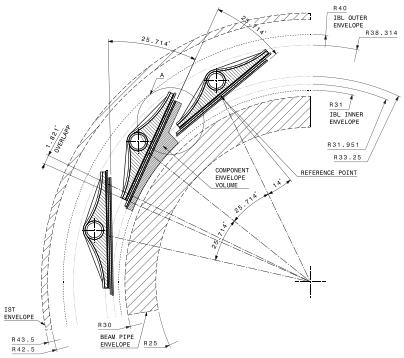

Figure 1.8.: Transverse schematic of the IBL [23].

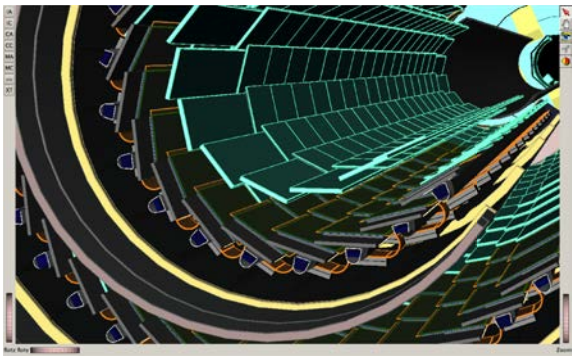

Figure 1.9.: Render of innermost parts of the pixel detector with inserted 4th layer [23].

1.3. Semiconductor Pixel Sensors

It is of interest to reconstruct the point of interaction, the so-called primary vertex. Furthermore, there are some particles created which do not decay immediately, but do not live long enough to pass through the entire detector either. The most prominent example are hadrons containing a b-quark. As their life-time is dominated by the decay of the heavy quark (i.e. the b-quark) which is mediated by the weak interaction, the fact that the corresponding CKM matrix elements are small leads to a long mean life-time (as dictated by the exponential decay law) of those hadrons. The long mean life-time together with a Lorentz boost enables those particles to traverse several hundreds of micrometers before decaying. This subsequent decay vertex is called the secondary vertex. Precise track reconstruction gives rise to b-tagging, a technique to identify hadrons containing a b-quark. To be able to perform b-tagging, it is necessary to resolve the primary vertex as well as the tracks pointing towards it, hence a tracking detector with sufficient resolution is desired.

1.3.1. Energy Loss in Matter

Charged particles passing through matter will deposit energy via ionisation and atomic as well as lattice excitation. This can be exploited for the purpose of particle detection, especially if prepared in a way to provide spatial resolution.
The mean energy loss per distance travelled, i.e. the stopping power, is given by the Bethe-Bloch formula:

$$\left\langle -\frac{dE}{dx} \right\rangle = Kz^2 \frac{Z}{A} \frac{1}{\beta^2} \left[\frac{1}{2} \ln \left(\frac{2m_e c^2 \beta^2 \gamma^2 W_{\max}}{I^2} \right) - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \tag{1.1}$$

With:

$$K = 4\pi N_A r_e^2 m_e c^2 \tag{1.2}$$

$$W_{\max} = \frac{2m_e c^2 \beta^2 \gamma^2}{1 + 2\gamma m_e/M + (m_e/M)^2} \tag{1.3}$$

The various variables are summarised in Table 1.1. A plot of the stopping power of anti-muons in copper is given in Figure 1.10, the section where it is appropriately modelled by the Bethe-Bloch equation is indicated.

Variable	Explanation
N_A	Avogadro's number
r_e	classical electron radius
z	charge number of incident particle
Z	atomic number of absorber
A	atomic mass of absorber
β	$v = \beta c$
m_e	electron mass
c	speed of light in vacuum
γ	Lorentz factor
$W_{\max}$	max. energy transfer in single collision
I	mean excitation energy
$\delta(\beta\gamma)$	density effect correction to ionization energy loss
M	incident particle mass

Table 1.1.: Variables appearing in the Bethe-Bloch equation.

The Bethe-Bloch equation describes the mean energy loss per unit distance well in the range $0.1 \lesssim \beta\gamma \lesssim 1000$. At higher energies, radiative processes need to be considered,

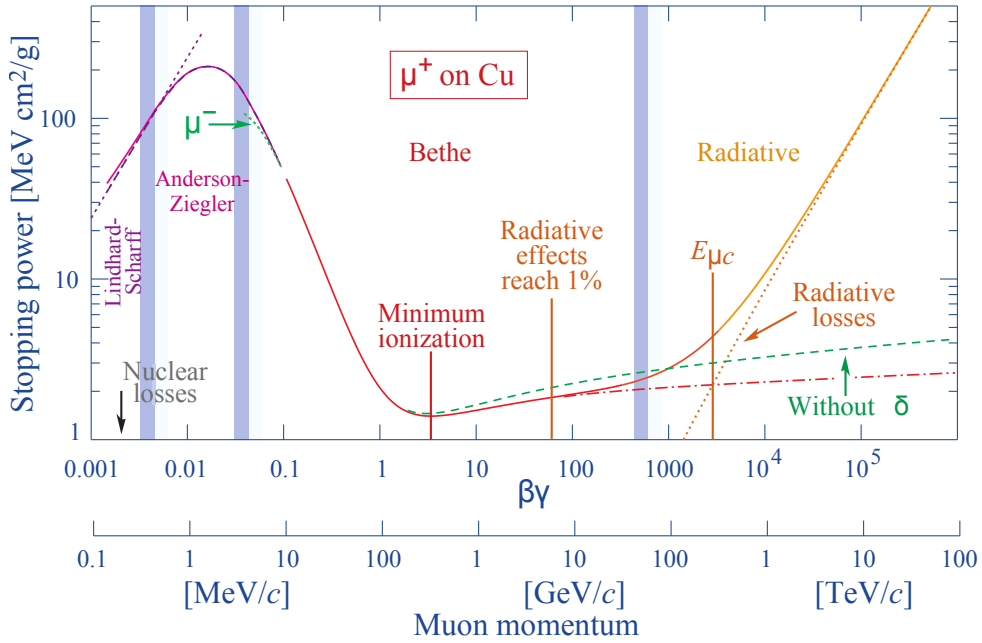

Figure 1.10.: Stopping power (normalised to the density of the material) $\langle -dE/dx \rangle$ for anti-muons in copper [1].

lower energies require shell corrections.

Despite that, the regime where the Bethe-Bloch equation holds is usually the one of interest in high energy physics. In this region the stopping power for a given material mainly depends on β (with just slight M dependence in $W_{\max}$). Another interesting feature about the Bethe-Bloch equation is that it features a broad minimum over a large range of $\beta\gamma$. A particle in this region is called minimal ionising particle (MIP), which again holds true for many particles in high energy physics. As a rule of thumb, they experience a mean stopping power of roughly $1.5\,\mathrm{MeV\ cm^2\ g^{-1}}$.

A very important limitation to the Bethe-Bloch equation is that it does not describe the energy loss of electrons or positrons very well. This is due to two effects, firstly due to the low mass of them and secondly in the case of electrons, scattering on the same particle. For electrons, Møller scattering also has to be taken into account, as is the case with positrons and Bhabha scattering.

Equation 1.1 only yields the average stopping power, but does not describe the distribution for individual particles. In rare cases, secondary electrons can be released by the primary particle passing through the detector leading to larger ionisations, the so-called delta electrons. Such high energy collisions are the cause of the distribution not being a symmetric Gaussian. Instead, this leads to a skewed distribution, featuring a long tail at high deposited energies, which can be described by a Landau distribution. The skewness

depends on the detector thickness. For thicker detectors the distribution approaches a Gaussian. For thin absorbers, including thin Si sensors, the Landau distribution fails in its description as shown by H. Bichsel [24]. This deviation has also been experimentally observed by S. Meroli et al. [25]. They fitted the measured energy loss distributions for thin silicon layers via a convolution of a Landau and normal distribution. Figure 1.11

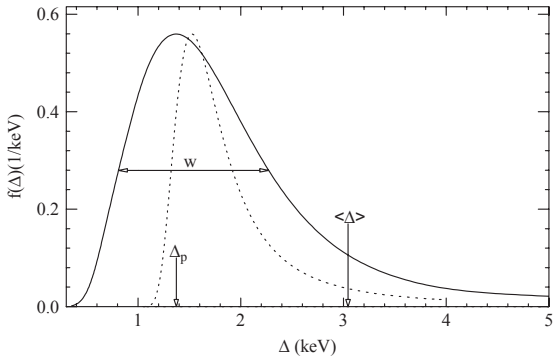

Figure 1.11.: Probability density function for energy losses in 1.2 cm Ar gas according to Bichsel (solid line) and the Landau description (dotted line) [24].

shows Bichsel's straggling functions $f(\Delta)$ compared to the Landau description. It can be seen that the mean deposited energy $\langle\Delta\rangle$ is a bad measure when working on a single particle level (e.g. for particle identification). Rather, the most probable value Δ_p should be used.

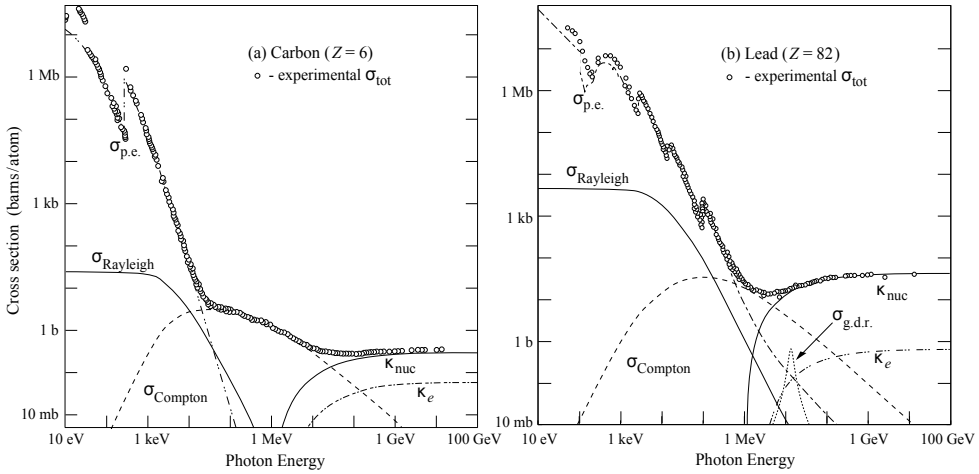

Figure 1.12.: Photon total cross section [1].

Unlike charged particles, photons cannot directly ionise matter, instead there are three effects which dominate the photon-matter-interaction in different energy regimes. At low energies up to $\mathcal{O}(10)$ keV, the photoelectric effect dominates. The incident photon is

fully absorbed by an electron, which is emitted in this process. At higher energies, the Compton effect takes over, during which again an electron is released, but the photon is not fully absorbed, but re-emitted with a decreased wavelength. At energies sufficient to produce e^+e^--pairs, i.e. slightly above 1 MeV, pair production is possible and will become the dominant effect of photon-matter-interaction at even higher energies.
The cross section of these effects for carbon and lead are shown in Figure 1.12, the Feynman diagrams are given in Figure 1.13. $\sigma_{\mathrm{p.e}}$ is the cross section for the photoelectric effect, $\sigma_{\mathrm{Compton}}$ for the Compton effect and κ_{nuc} as well as κ_e for pair production (in nuclear and electron field). Additionally, the coherent scattering cross section $\sigma_{\mathrm{Rayleigh}}$ and nuclear reaction interactions ($\sigma_{\mathrm{g.d.r.}}$, Giant Dipole Resonance) are given.

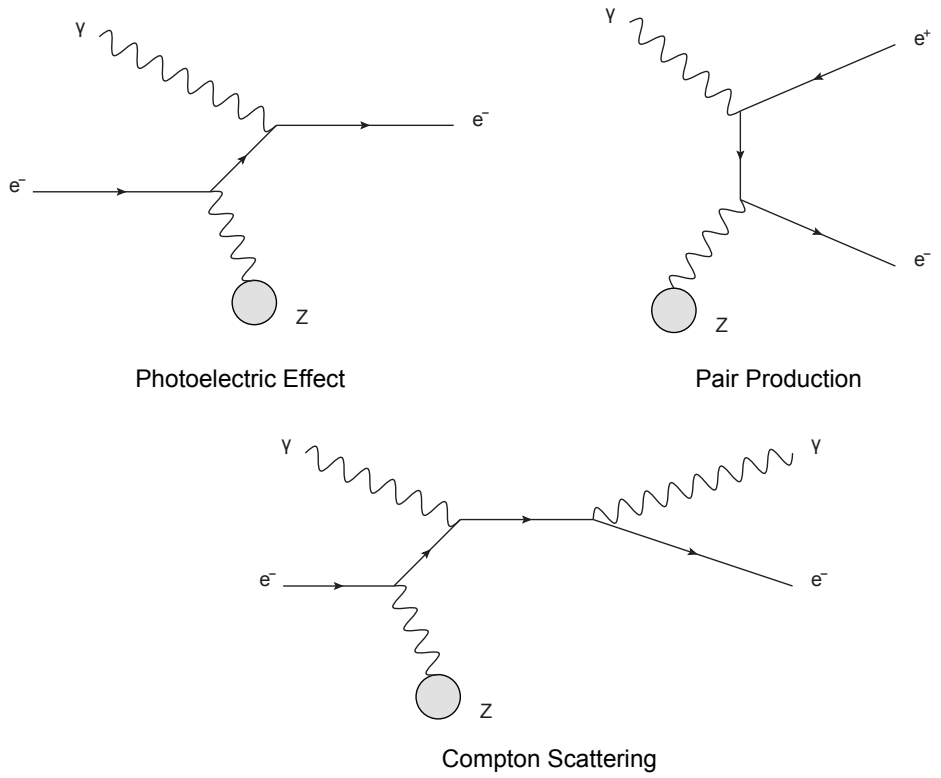

Figure 1.13.: Feynman diagrams for interactions of photons with matter, Z depicts a nucleus.

1.3.2. Signal Generation in Silicon

In semiconductors, the creation of electron-hole pairs, which induce a signal on electrodes when drifting in an electric collection field, is the key mechanism used in particle detection.

While the band-gap in silicon is 1.1 eV at room temperature, it requires a mean deposited energy of 3.67 eV for the formation of an electron-hole pair. This is due to a large fraction of deposited energy going into lattice vibrations because of momentum conservation. This has also effects on the variance of the number of released charge carriers N by a deposited energy E. It is reduced by the so-called Fano factor F, compared to the default Poisson process, i.e. $\sigma_N = \sqrt{FN}$. Using $N = E/E_i$ where E_i is the required energy for the creation of an electron-hole pair, this yields an energy resolution of $\sigma_E/E = \sqrt{FE_i/E}$.
For the purpose of charge separation, silicon is doped and operated as a reversely biased pn-diode. This provides an electric field for charge collection via drift in this field. Furthermore, it removes intrinsic charge carriers from the depletion zone, allowing the detection of the induced charges. Additionally, the doped regions can be manufactured and used as the required electrodes. By applying a bias voltage, the depletion zone in the bulk is increased until the sensor is fully depleted. The depletion depth is given by [1]:

$$W = \sqrt{2\varepsilon \left(V + V_{\text{bi}}\right)/Ne} \tag{1.4}$$

In Equation 1.4, ε is the dielectric constant of silicon, V is the applied bias voltage and V_{bi} the built-in voltage, N the doping concentration and e the elementary charge. The built-in voltage is obtained by integrating over the electric field in the intrinsic space charge region of the pn-junction.
Using typical values for a 300 µm silicon detector, a MIP releases about $29 \cdot 10^3$ electron-hole pairs when passing through the sensor, which requires roughly 70 V to fully deplete [26].

1.3.3. Signal Collection in Silicon Detectors

Once charge carriers are released in the silicon bulk, they start to move due to two processes: diffusion and drift. Diffusion is a random walk process and thus leads to a Gaussian broadening of the charge cloud. Drift is due to the total electric field present in the sensor.
The drift velocity of charge carriers is given by $\vec{v} = \mu\vec{E}$. Using a larger electric field than required for full depletion allows faster charge collection. The important material constant is the charge carrier mobility μ, which differs for electrons and holes. Via the drift velocity and the electric field, the drift time can be computed. Using the Einstein-Smoluchowski relation to relate the diffusion constant D to the charge carrier mobility $D = \mu k_B T$ (where k_B is Boltzmann's constant and T the temperatur) it is possible to estimate the broadening, given the drift time. Using the previous example of a 300 µm

silicon sensor and a bias voltage twice the required one for full depletion, this yields a maximum drift time of roughly 5 ns for electrons and 16 ns for holes. The Gaussian broadening due to diffusion can be computed to be about $\sigma \approx 4\,\mu\text{m}$ [26].
An important theorem is the Shockley-Ramo theorem [27, 28], which states that the current induced in an electrode is given by the instantaneous drift of the charge carriers in the field of the electrode, and not the charge collected by it:

$$i = q\vec{v}\vec{E}_v \tag{1.5}$$

Where q is the charge of the charge carrier, $\vec{v}$ is the instantaneous velocity and $\vec{E}_v$ the weighting field [28].

1.3.4. Semiconductor Sensor Design

For spatial reconstruction of tracks, detectors need to provide spatial resolution. This is achieved by implementing electrodes in one or two dimensions in the sensor, yielding either strip or pixel detectors. Having a segmented pixel structure allows to reconstruct

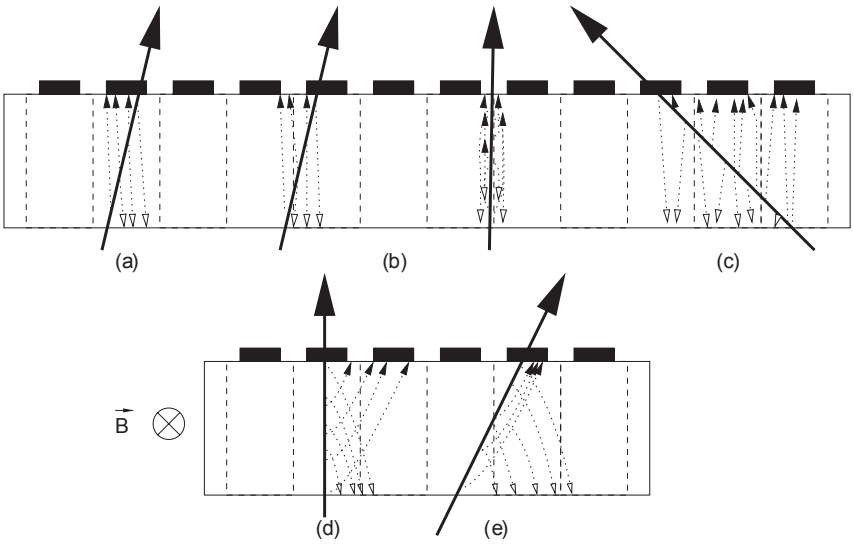

Figure 1.14.: Cluster formation with and without magnetic field [26].

hit positions in two dimensions. The important figure is the pixel pitch d. It is important to note, that the pixel pitch is not equal to the electrode size, but the distance between two pixels. If a pixel detects a hit, the spatial resolution is at worst $\sigma = d/\sqrt{12}$. This is due to the fact that there is no additional information where the particle actually hit the pixel and the hit probability density is assumed to be flat across the pixel. The resolution

can be increased if additional information is available. This is the case when more than one pixel is hit, yielding a cluster of hit pixels. In the case of a two pixel cluster, the hit position probability distribution is not flat, but peaks somewhere between the hit pixels. Figure 1.14 (a-c) shows cluster formation due to hits in-between pixels as well as due to inclined tracks. Additionally, a magnetic field can deflect the tracks of the charge carriers via the Hall effect (Fig. 1.14 (d-e)). The angle of this deviation is known as the Lorentz angle.
Reconstructing the exact hit position from the cluster is not trivial. In the simplest treatment, the hit position is simply derived by a centre of gravity (COG) calculation. If additional information on the collected charge is available, this can be used to weigh the hit positions:

$$X_{\mathrm{COG}} = \frac{\sum_{\mathrm{cluster}} S_i x_i}{\sum_{\mathrm{cluster}} S_j} \tag{1.6}$$

Where x_i and S_i are the position and charge signal of hit pixel i. Corrections to this clustering algorithm can be done via the η-correction algorithm. It is based on the fact, that charge sharing is non-linear due to the shape of the diffusion cloud. As the η distribution has to be obtained from data, it is a data-driven correction algorithm. Additionally, when incoming tracks are heavily inclined, other algorithms than the COG should be used, for example the head-tail position finding algorithm [29].

1.3.5. Signal Processing

As soon as the signal has been induced, it must be processed. Semiconductor detectors can be divided into monolithic and hybrid detectors, depending on whether the read-out electronics are located in the same bulk of semiconductor (monolithic) or attached via some sort of connection, usually bump bonds, to the sensor (hybrid). While the upside of hybrid pixel sensors is that the read-out electronics per pixel can be quite sophisticated, allowing fast parallel read-out of all pixels at once, they introduce more material into the detector and feature a larger pixel size, limited by the size of the read-out pixel and the interconnection technology. Monolithic sensors on the other hand can be easily manufactured with a very small pixel pitch, but have a slower read-out.
Of course, different technologies can be more advanced than others which is reflected in the cost and yield a supplier can deliver in a given time. Ultimately, every choice has to be economically viable and is bound by financial as well as time constraints and the current state of research.
Therefore, the different types are used in different applications. For the LHC experiments,

the read-out time has to be very fast which is a major reason hybrid pixel sensors are used in the ATLAS pixel detector.

1.3.6. The ATLAS FE-I4 Read-Out Chip

The read-out electronics used in the IBL, the Front-End-I4 (FE-I4), is the successor of the one used in the existing pixel detector (FE-I3 [30]). An upgrade was required since the FE-I3 is incapable of dealing with the higher hit rates which are present in proximity of the interaction point. Additionally, it is not able to deal with the higher fluences in terms of radiation hardness.
The FE-I4 features a smaller pixel size, $250\times50\,\mu m^2$ compared to $400\times50\,\mu m^2$ for the old FE. In total 26 880 pixels are read out by the FE-I4, arranged in 80 columns along the direction of the long pixel edge, and 336 rows, yielding a roughly quadratic active area. Despite having smaller pixels, the FE-I4 has a larger active area (in total as well as fractional terms) than the FE-I3. This is since the total pixelated area is larger, whereas the inactive area housing the additional logic has remained the same size.

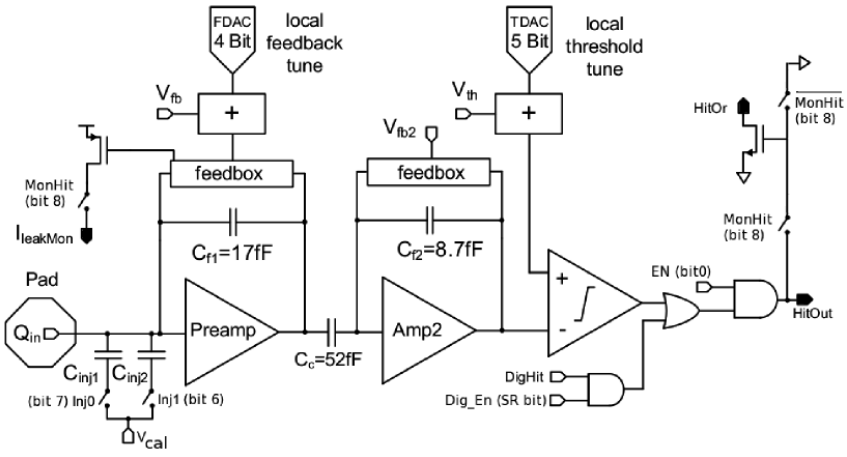

Figure 1.15.: Analogue stage of an FE-I4 pixel [31].

The analogue section of each pixel of the FE-I4 (Fig. 1.15) consists of a two stage charge sensitive amplification circuit followed by a discriminator stage. Several digital to analogue converters (DAC) allow tuning of the individual pixels, e.g. they allow to steer the discharge current or set the discriminator threshold. This discriminator threshold is essential for understanding the signal value which will be ultimately read out by the FE. The signal after the discriminator can either be in a high or a low state, depending if the input voltage is below or above the threshold value. The duration of this pulse in clock

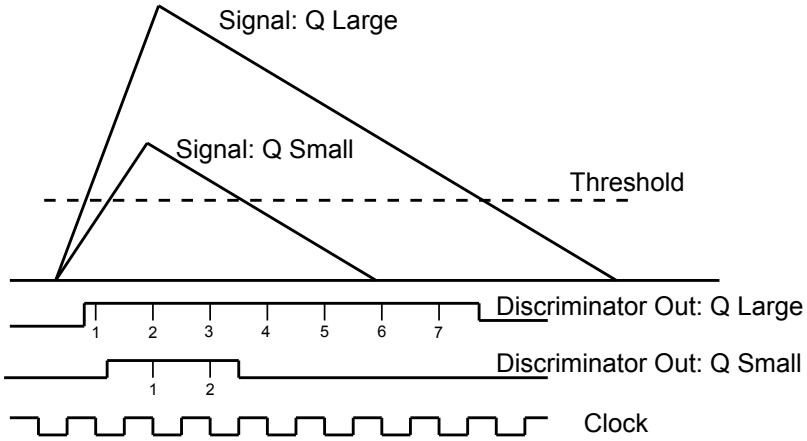

Figure 1.16.: ToT for different charges, the high charge results in a ToT 7 hit, whereas the small charge results in a ToT 2 hit.

cycles is related to the charge collected by the pixel. This value is called ToT, which is short for time over threshold (Fig. 1.16). The threshold value, but also the discharge current (which controls the steepness of the falling slope of the signal), influences the final ToT value. Thus, a ToT tuning and calibration is required to obtain a correct charge read-out.

For tuning, but also for testing purposes, a charge injection circuit is located in front of the amplification stages, allowing a well defined charge to be injected into the amplifier.

The digital sections of the read-out circuits are arranged so that four adjacent pixels share some of the digital circuitry. The output of the discriminator of the analogue section is fed to a hit processing stage which computes the ToT. While this part of the digital circuitry is unique, the part for triggering and transmission is shared by the four adjacent pixels. If any of the pixels in the four pixel block detects a signal, all four pixels are read out. Given that they are located side by side, this reduces the read-out load, since often clusters of more than one pixel in close proximity detect a hit, and if it is in the same block of four pixels, only the read-out of one digital block is necessary.

Additionally, the FE-I4 was designed to have a lower material budget and power consumption than the FE-I3. Furthermore, the read-out speed was increased from 40 Mb/s to 160 Mb/s. These modifications make the FE-I4 a feasible candidate for use in the outer layers of the ATLAS inner tracking detector for the Phase-II upgrade planned for the ATLAS detector during the upgrade phase of the LHC to the High-Luminosity-LHC (HL-LHC) [32].

2. Testbeams

To determine the properties of newly developed detectors as well as read-out chips, a beam telescope at a testbeam facility can be used. A beam telescope is a hardware set-up, used to instrument a beam line for the purpose of track reconstruction at a level of individual tracks for each charged particle crossing the telescope. With those tracks, various analyses can be performed. E.g. expected hit positions can be extrapolated and efficiency studies carried out.

2.1. Hardware Set-Up: The Telescope

The crucial components of the telescope are the reference pixel sensors and their read-out hardware. To achieve a good spatial resolution, they have a relatively small pitch size. While in principle the number of reference planes is arbitrary, usually six are used. This is a trade-off between resolution and material budget. While increasing the number of planes increases the resolution by contributing with additional spatial information, the extra material increases multiple scattering, and thus acts against this improvement.
In an actual test beam set-up the so-called device under test (DUT) is also placed inside the telescope, such that it intersects the particle trajectories. Via the reconstructed tracks, the hit positions can be extrapolated and different types of analyses can be performed. Figure 2.1 shows a schematic of a complete testbeam set-up, whereas a picture of the telescope planes with four inserted DUTs is shown in Figure 2.2.
In addition to the reference sensors and their read-out hardware, further components dedicated to temporal synchronisation as well as triggering are necessary.
While in principle any sensor could be used as a reference, the current generation of DESY pixel telescopes, which originated from the EUDET telescope, all use the Mimosa26 pixel sensor.

2.1.1. Mimosa26

The Mimosa26 is a monolithic active pixel sensor (MAPS) developed by IPHC in Strasbourg. It features 576×1152 pixels with a pitch of 18.4 µm in both directions, giving an

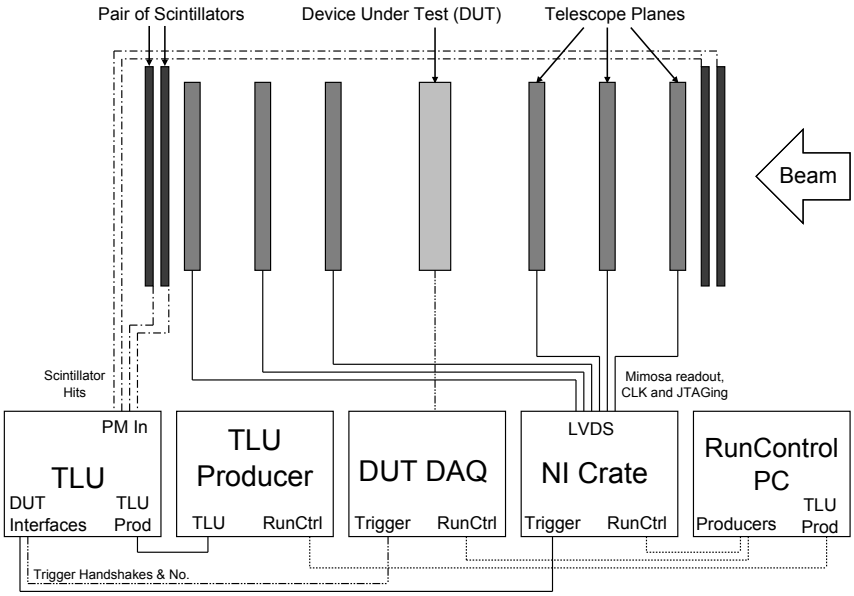

Figure 2.1.: Schematic of the telescope layout depicting read-out hardware (top) as well as software (bottom) and their interconnection.

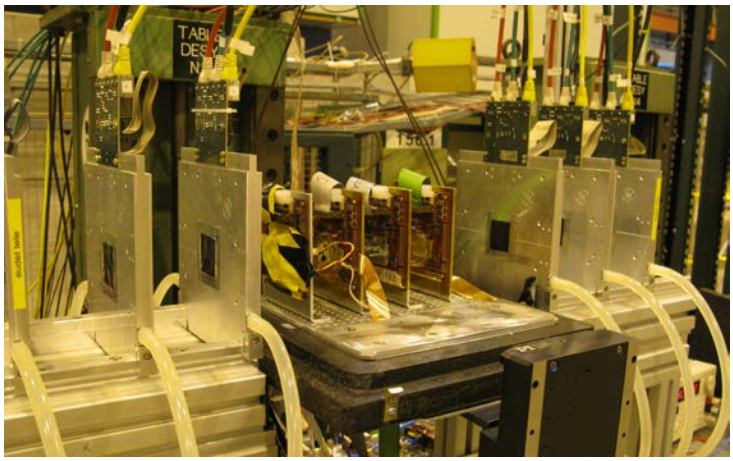

Figure 2.2.: Telescope with four ATLAS FE-I4 DUTs in the centre.

active area of roughly 13.7×21.5 mm^2 (Fig. 2.3). It is read out by a rolling shutter read-out, processing columns in parallel, row by row. One complete read-out takes 115.2 µs. Each pixel has its individual amplification stage and features correlated double sampling (CDS) circuitry. CDS is a technique in which the signal after the hit is recorded, as well as the signal after resetting the pixel. This allows to subtract the base value from the signal, which eliminates any offset and thus reduces noise.

At the end of each column, zero-suppression circuitry is implemented, so only events that contain a hit can be stored. Afterwards, the signals from the columns get processed in

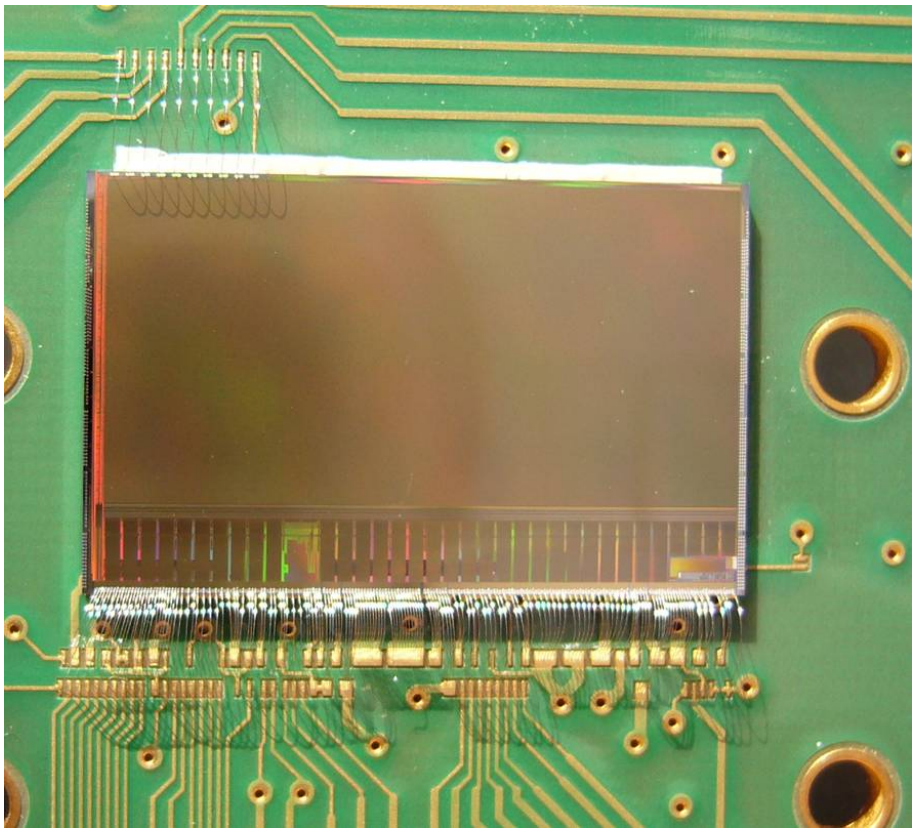

Figure 2.3.: Mimosa26 sensor as used in the telescope [33].

18 banks of logical circuitry. Configuration of the individual registers in the Mimosa26 is done via JTAG. The analogue as well as the digital layout can be reviewed in Figure 2.4.

2.1.2. Trigger Hardware and Trigger Synchronization

The Trigger Logic Unit (TLU) is a piece of hardware dedicated to two tasks: triggering as well as synchronisation of the data streams [35]. Data must only be recorded when a charged particle crosses the telescope. Usually, scintillator fingers read out by photomultiplier tubes (PMT) are used for triggering. To limit the active trigger area to a rectangular shape, two crossed scintillator fingers are used on both ends of the telescope, thus yielding four scintillators per telescope in a usual testbeam set-up.
The PMT signals are sent to the TLU (which can process up to four of those signals), which then triggers a telescope read-out if the required coincidence criterium is matched. This criterium can be set by the operator. In principle, any logical operation of the four inputs can be set, but the typical requirement is a coincidence of all four scintillators.
The TLU is connected to all read-out systems and issues a signal on a specified line to

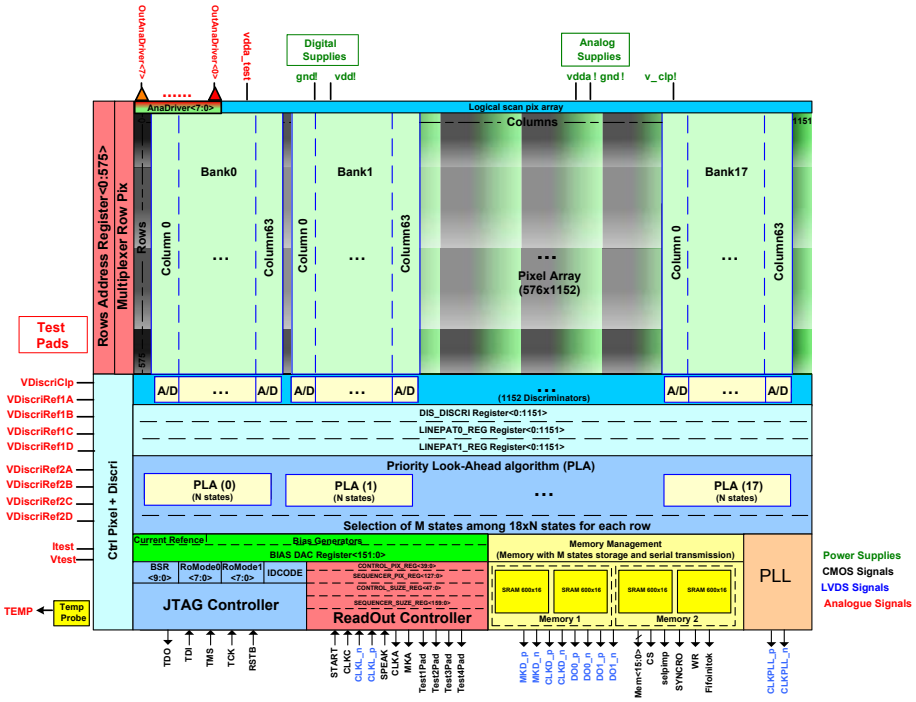

Figure 2.4.: Schematic layout of the Mimosa26 [34].

trigger a read-out. While there are a total of three modes, the most important and most powerful one in terms of functionality is the trigger-data-handshake mode.
In this mode, all recorded events receive a consecutive trigger number, which allows synchronisation of all the data. The other two modes either do not exchange a trigger number (simple-handshake) or feature no handshake at all (no-handshake).

2.2. EUDAQ: Data Acquisition

EUDAQ is the software framework used for data acquisition with the telescope [36]. The main program of EUDAQ is Run Control, to which other parts of EUDAQ can connect (via a TCP/IP connection) to be steered by Run Control. Run Control is therefore used to initiate configurations as well as to start and stop the actual data acquisition. This modular design layout is depicted in Figure 2.5, where the individual data paths to other programs of EUDAQ can be retraced.
Other very important parts of EUDAQ are the so called Producers. Those are the programs which read out all the individual sensors, reference planes, as well as DUTs, and thus produce the data streams. Therefore, in a typical EUDAQ operation, usually more than one Producer is connected to Run Control at the same time.

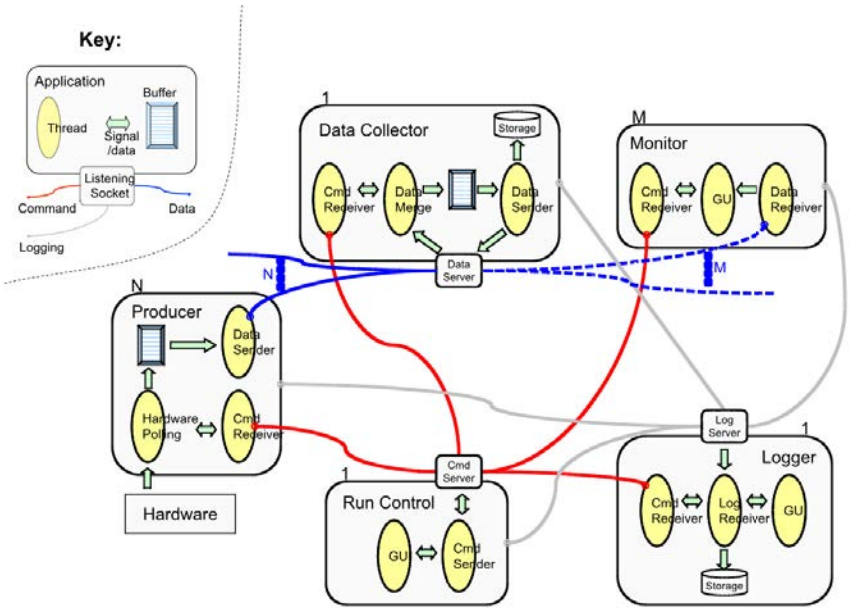

Figure 2.5.: Modular layout of EUDAQ [36].

A special type of Producer is the TLUProducer, which is used to configure and interface the TLU. For example the previously mentioned coincidence requirement is set this way. Configuration is done via one central configuration file, which has sections for all the individual Producers. These settings get parsed to the Producers, which then process values from this file.
The Data Collector is the part of EUDAQ, which acquires all the raw data and writes it to disk. Additional programs exist to monitor log messages as well as on-line plots of the different sensors (Logger and Monitor).
Since EUDAQ is openly available and Producers for the telescope reference sensors are provided, users have the possibility to write their own Producers to include their devices into the telescope data stream. Integration at this early point allows easy reconstruction without the need to merge data in retrospect.

2.3. EUTelescope: Event Reconstruction

After data acquisition, the data has to be reconstructed and analysed. EUTelescope is the framework designed for this purpose.
EUTelescope itself is a collection of Marlin[1] processors and thus, like Marlin, part of

[1]Modular Analysis and Reconstruction for the Linear collider

the ILCSoft software package. Versions prior to this work only use Gear[2] for geometry description of the telescope layout and the pixel geometry. Gear is very restrictive when it comes to pixel layouts, allowing only regularly arranged rectangular pixels.
Reconstruction can be divided into the following steps: data conversion, clustering and hitmaking as well as alignment and track fitting (Fig. 2.6).

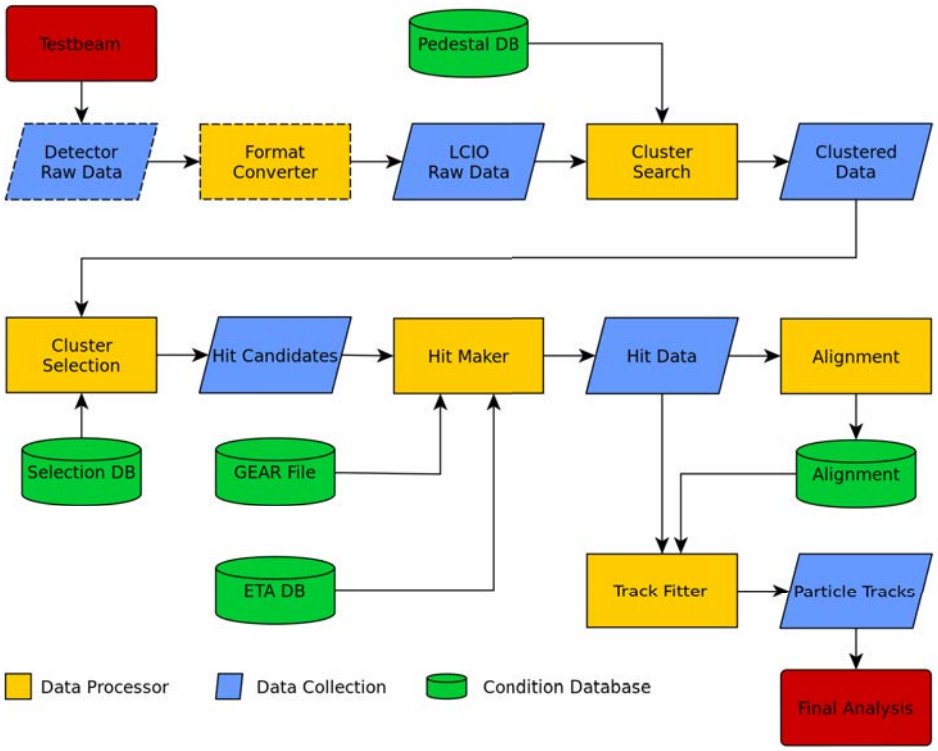

Figure 2.6.: Schematic overview of data processing in EUTelescope [37].

Data conversion (handled by the Format Converter processor in Fig. 2.6) is necessary to bring the data of all different read-out systems into one compliant format used in the reconstruction framework. After this is done, hit pixels in every event have to be determined and grouped into clusters of which the hit position has to be derived (the Cluster Search, Cluster Selection as well as the Hit Maker processor). Subsequently, an alignment (Alignment processor) has to be determined. This is required, as the knowledge of the position of the sensors has to be determined with a precision that cannot be reached using mechanical measurements alone. With those alignment values, the final reconstruction step can be carried out and tracks can be fitted (Track Fitter processor). These tracks can then be used to perform an analysis, either with EUTelescope or

[2]Geometry API for Reconstruction

some other framework, like it is in the case of the ATLAS pixel community, which uses TBmonII.
Additionally, several condition databases can be used in the reconstruction. Some of them are only of importance where non-zero-suppressed data is provided. In this case, signal processing requires to perform cuts on signal to noise ratios of individual pixels and clusters. This requires the pedestal and selection database. With zero-suppressed output, they boil down to one noisy pixel collection, which can be used to mask or remove noisy pixels in one of the early steps. The GEAR and alignment databases store the geometrical information. The GEAR file contains information from the user on the set-up whereas computed corrections are stored in the alignment database. The η-database can be used for η-algorithm corrections to cluster positions.
While this is a rough layout of any EUTELESCOPE analysis, this procedure is not carved in stone but highly modular and flexible. Further steps (e.g. region of interest selection or applying cuts on certain parameters) can be included while other steps could be omitted (e.g. no dedicated cluster selection). Additionally, recent developments change some of these concepts and partly get rid of non-zero-suppressed data processing in later steps, meaning that data of this type has to be preprocessed and zero-suppression has to be done in the initial steps.

2.4. Track Analysis - TBmonII

While no changes to TBmonII were made during this work, it is the final part of the analysis for ATLAS pixels and thus will be briefly introduced. The whole TBmonII analysis is configured via external configuration files. After loading those configurations, the tracks from the EUTELESCOPE reconstruction are loaded and processed.
Various preprocessors are responsible for tasks like clustering, alignment or applying selection criteria. A schematic overview of the different parts of TBmonII is given in Figure 2.7. TBmonII always requires at least two DUTs to be present in the dumped data. This is necessary, since the active times of DUTs and telescope planes do not match. Tracks seen by the telescope might have happened outside the DUTs active time window and are thus not recorded. Using multiple DUTs featuring the same active time allows selection of tracks which lie within this time window.
While at least two DUTs are required, TBmonII supports as many DUTs as included in the testbeam set-up. It is also possible to combine multiple runs with the same set-up for increased tracks and therefore higher statistics.

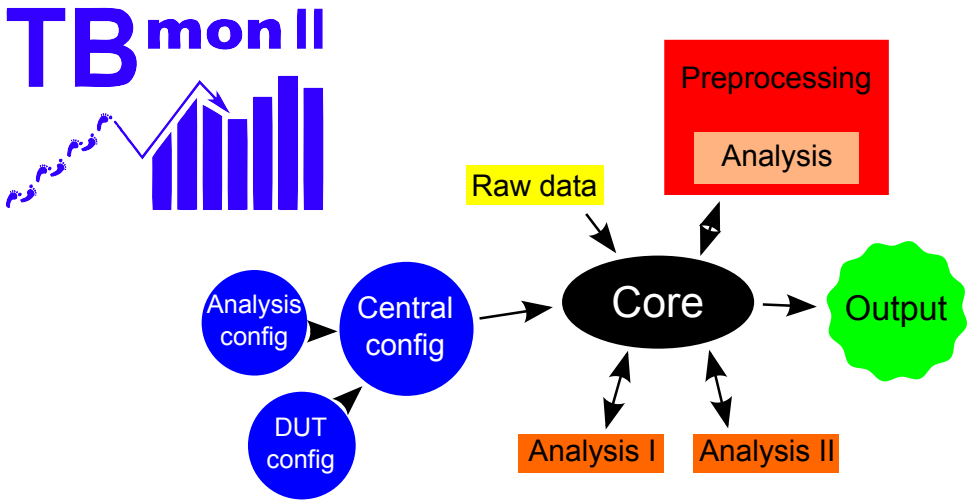

Figure 2.7.: Schematic overview of the mode of operation of TBmonII.

After preprocession, TBmonII allows to perform various analyses on the selected tracks. For example, hit inefficiencies can be computed from selected tracks and their intersection on the various DUTs.

2.5. USBpix - An ATLAS Pixel Read-Out System

For the purpose of operating and testing the FE-I3 and FE-I4 in the lab or at testbeams, a portable data acquisition (DAQ) system has been developed [38]. It consists of the USBpix board, which forms the hardware of this DAQ system, and a read-out software called STControl. It is based on the PixLib libraries, which is the code to interface the ATLAS Pixel Detector read-out drivers (RODs). The software package also provides a graphical user interface (GUI) for users to interface with the hardware, i.e. load and store configurations, run scans, perform tuning and start testbeam data.

USBpix

The USBpix hardware (Fig. 2.8) is based on the S3 MultiIO USB Card (or Board), featuring a Xilinx Spartan3 FPGA, an 8051 microcontroller, as well as a USB interface and 2 MiB[3] of SRAM [39].
Different adapter cards can be connected to the MultiIO Board, allowing to interface many different kinds of sensor boards (FE-I3/I4, 4-chip-modules etc.). This makes the

[3] 1 MiB = 2^{20} bytes

USBpix DAQ system a versatile and cheap in-the-lab solution to operate the supported FEs, without the need to acquire an expensive copy of the actual DAQ system.

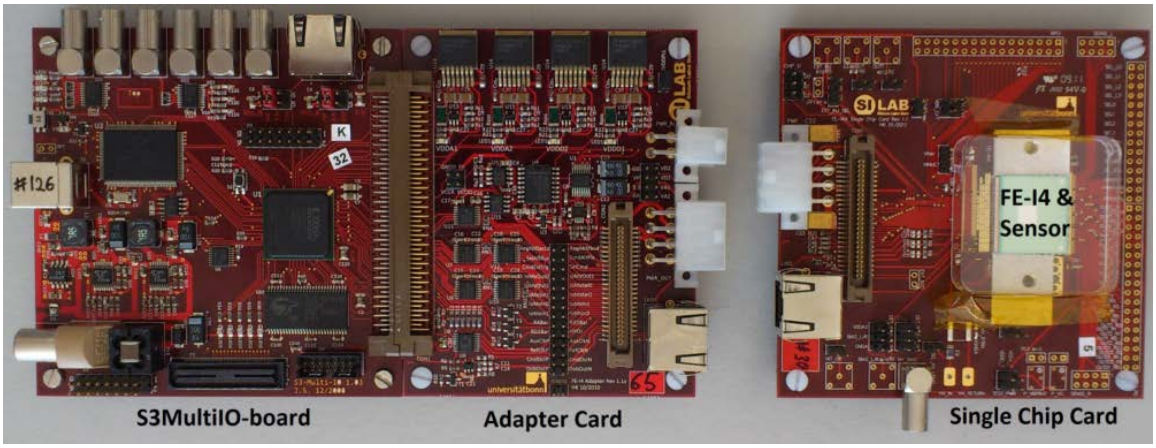

Figure 2.8.: MultiIO Board with attached adapter card for FE-I4 as well as a FE-I4 on a lab board [40].

STControl

STControl is the high level part of the DAQ software, providing the GUI and processing user input to run scans or tune the FE. It does that by interfacing the FE via the PixLib classes.
STControl uses the Qt4 library for the GUI, but also heavily relies on the signal/slot mechanism provided by Qt for inter-thread communication. Aside from being able to read out the ATLAS FEs during testbeam, the actual EUDAQ producer is part of STControl when compiled against EUDAQ (which is necessary for testbeam operation).

3. ATLAS Four Chip Modules for the HL-LHC

3.1. Novel Sensor and Module Layouts

Typical sensor layouts usually consist of rectangular pixel implants on one side of the sensor and a biasing electrode implant on the other (Fig. 3.1, left). This has the disadvantage that the bias voltage required for full depletion scales with the electrode distance. In addition to that, the charge collection distance (CCD) is fixed by the sensor thickness. As higher bias voltages are required by irradiated samples due to radiation damage (and thus a change in effective doping concentration), pixel implants as pillars inside the silicon bulk (Fig. 3.1, right) with a shorter electrode distance would require a smaller bias voltage. Furthermore, pillar electrodes have the advantage that the CCD is decoupled from the sensor thickness, which is advantageous for some sensor technologies. Thus, these pillar electrodes are one approach currently investigated for future pixel modules for ATLAS.

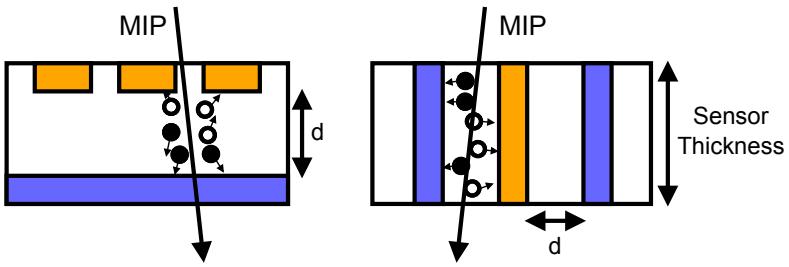

Figure 3.1.: New electrode implants, planar sensors (left) compared to sensors with pillar electrodes (right), indicated is the electrode distance d.

Aside from new sensor layouts and technologies, cheap module layouts are investigated for the upcoming ATLAS upgrades [32]. Especially larger size modules with reduced production cost are of interest. ATLAS four chip modules (Fig. 3.2) are larger sensors

read out by four FE-I4 read-out chips. This reduces costs, as the cost of bump bonding mainly scales with the number of handling steps during module bonding, not module size.

3.2. Problems with the current Set-Up

New sensor and module layouts pose a challenge in read-out as well as in the description of their geometry in any software package which processes data from them. For example, in testbeam operation, four chip modules require the simultaneous read-out of four FEs, which then need to be merged and put into the datastream. They also exhibit a sensor layout which deviates from simple rectangular pixel patterns. This is partly due to the fact that they are composed of two double chip modules with additional space in between, as well as having prolonged pixels between the two FEs of one double chip module (Fig. 3.3). While treatment of this geometry was already supported by TBmonII prior to this

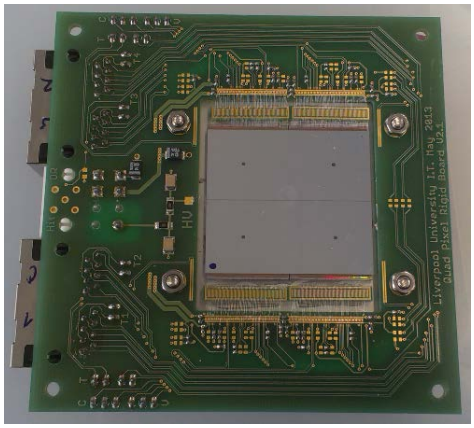

Figure 3.2.: New module layout, ATLAS four chip prototype.

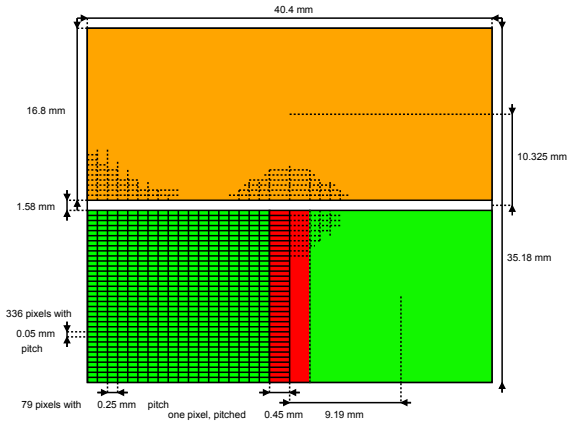

Figure 3.3.: Sketch of the pixel layout of a four chip prototype.

work, the USBpix read-out system did not support the read-out of multiple chip modules. Furthermore, EUTelescope did not allow such pixel layouts.
Thus, the aim of this work was to implement the necessary functionality for DAQ with the USBpix framework, as well as the correct reconstruction for this data with EUTelescope. In consideration of future pixel layouts which might even feature staggered pixels or similar, the framework was adapted to support even more generic geometries. Additionally, verification of correct operation as well as test cases for test-driven development (TDD) were implemented to ensure proper functionality and quality control during the future development process.

4. Modifications to STControl and EUDAQ

As USBpix had been recently updated to support FE-I4 four chip modules with the burn-in cards by the work of J. Agricola [41], this functionality had to be added into the EUDAQ producer of STControl.

4.1. Introduction

The burn-in card (Fig. 4.1) is an adapter card for the USBpix MultiIO board, featuring support for up to four read-out channels. For this purpose, the FPGA firmware (FW) had been adapted by J. Agricola, introducing four identical read-out sections for the four FEs. Since external memory is required (the SRAM on the MultiIO board) a memory arbiter had been implemented (Fig. 4.2), dealing with SRAM reading and writing. In the mode of raw data writing, as in the case of testbeam mode, the SRAM is divided into four equally sized parts, one for each read-out channel.

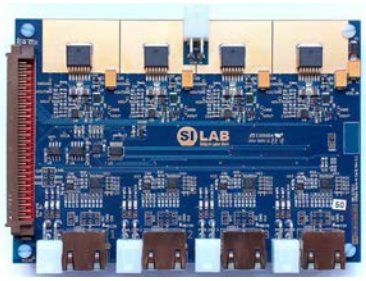

Figure 4.1.: The burn-in adapter card for the MultiIO board.

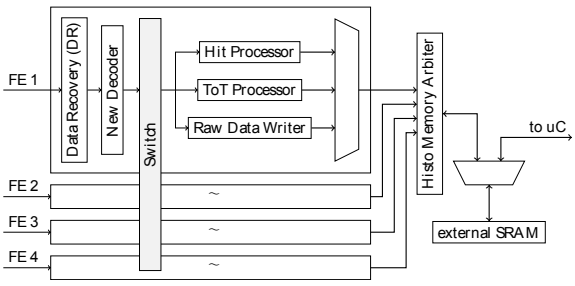

Figure 4.2.: Block overview of the FW read-out section [41].

4.2. EUDAQ Producer integration

As the EUDAQ Producer is integrated in STControl, one Producer is running per instance for STControl. Since STControl supports more than one MultiIO board per in-

stance, as well as more than one FE being read out by one board, multiple FEs produce data which ultimately has to be integrated into the EUDAQ data stream.
Previously, only multiple boards with one FE per board were supported. It was assumed that all individual FEs belonged to separate sensors. If two MultiIO boards were used to read out a double chip module (i.e. one sensor, read out by two FEs), EUDAQ had no knowledge about this and some workarounds had to be implemented in the final reconstruction. As this is a very error prone and user unfriendly approach, a new mechanism was incorporated to circumvent this problem. Since a burn-in adapter can also be used to read out a double chip module and two single chip modules, this had to be considered as well.
The modified Producer requires the following additional parameters to be passed via the configuration file: `boards` defining which boards are used and per board `modules[i]`, where `i` is the board ID, describing the module layout. All these entries have to be comma separated values. `boards` is the list of all MultiIO boards which are read out by this instance of STControl. This field was already needed in some cases. Since it is now required for the Producer to work, it is mandatory.
The `modules[i]` field is there to specify the module configuration for this board, in order of read-out channels. In the trivial case of a single chip board, the entry will be always `modules[i] = 1`, which tells the Producer that there is only one FE belonging to a single module. In the case of a four chip module attached to the board, the entry would need to look as follows: `modules[i] = 1,1,1,1` which will tell the Producer that there are four channels to be read out, all belonging to the same module. If instead only three read-out channels were used, reading out a double chip and a single chip module, `modules[i] = 1,1,2` would be the correct entry, given that the double chip is attached to read-out channel one and two, and the single chip module to three, respectively. While it is not necessary to use all read-out channels, they have to be used in ascending order. E.g. if only two of them are used, they have to be one and two.
Entries in `modules[i]` always start with a 1. The absolute sequence (i.e. the `sensorID`) in which modules are reconstructed in EUTelescope is given by the order of the entries in `board` and subsequently the entries in `modules[i]`.
A possible example of the relevant section in a EUDAQ configuration file can be seen in Listing 4.1 and the schematic layout is depicted in Figure 4.3. This will result in a four chip sensor with a `sensorID` of 20 (by default the first `sensorID` for an FEI4), two single chip modules (21 and 22) and a double chip module (23) in EUTelescope. This is done by passing the module configuration via the *begin of run event* (BORE), which

is an event entry in the data format, allowing additional information to be stored in the data.

```
[Producer.USBpixI4]
        ...
        boards = 10,137
        modules[10] = 1,1,1,1
        modules[137] = 1,2,3,3
        ...
```

Listing 4.1: Possible section of the EUDAQ USBpixI4 producer configuration.

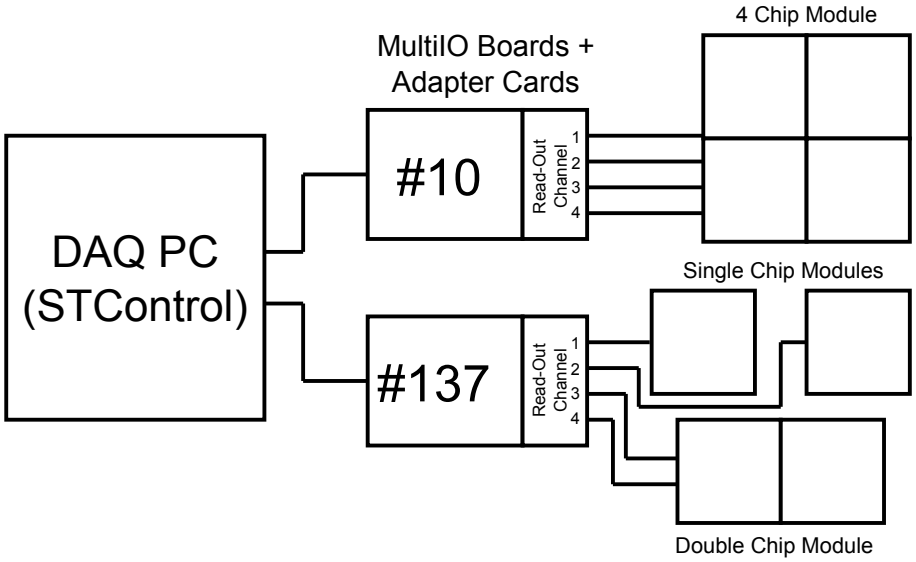

Figure 4.3.: Schematic of the USBpix set-up, as described by Listing 4.1.

4.3. EUDAQ Converter Plug-In

By calling the appropriate EUDAQ converter plug-in, the framework is able to process any foreign data format. Therefore, the converter plug-in has to be provided by the user and could be seen as an extension to the Producer. This design was motivated to reduce the risk of data corruption, as this allows the original data to get stored without further processing and the later offline reconstruction is responsible for the error prone interpretation of the data.

As the different revisions of the FE-I4 use slightly varying bit-masks in their data, there were two different converter plug-ins to process FE-I4A and FE-I4B data. They only varied in their included preprocessor directives, mainly defining macros for interpretation

of data.
To improve this, the converter plug-ins have been refactorised. The main difference is that the preprocessor define macros have been replaced by a templated class, featuring inlined methods to interpret the data. It is templated in the varying bitmasks, exploiting compile time function execution (CTFE) to compute required bitshifts for data processing, which depend on the variable bitmask. This way, multiple converter plug-ins could be merged into one abstract USBpix FE-I4 converter plug-in, which is implemented and instantiated with different bitmasks for the multiple revisions.
The converter plug-in reads in the module configuration attached to the BORE and creates appropriate sensors according to the number of chips belonging to a module. Since the data attached to the raw events is raw FE data, offsets have to be applied to create a single, continuous sensor in the reconstruction. In terms of the internal FE pixel numeration, the origin (i.e. pixel 0|0) is located on the upper left corner on the FE. As EUTelescope has a convention of setting the origin on the lower left corner, the pixels indices need to be shifted. The situation for a four chip module is shown in Figure 4.4.

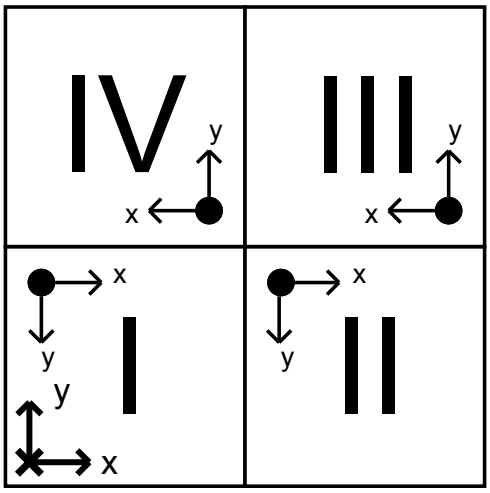

Figure 4.4.: FE layout on a four chip sensor, FE origins are indicated by a dot, the EUTelescope origin by a cross. The x direction is also referred to as *columns*, whereas y corresponds to *rows*.

The case of a single and double chip module can be traced back to the four chip layout by merely taking FE I or FE I and II into account. This has been exploited in the converter plug-in, and thus allows treatment of multiple sensor layouts with just one converter plug-in.

4.3.1. Laboratory Tests

To test if the producer and converter plug-ins work, a test with a radioactive source was carried out in the laboratory. A ^{90}Sr source was used to irradiate the sensor, it mainly decays via β^- decay, yielding an electron with an average energy of 0.546 MeV. The decay product ^{90}Y continues to undergo β^- decay, emitting a 2.28 MeV electron [42]. The higher energetic electron can penetrate the sensor and read-out chip, and can thus be detected by a scintillator below the sensor to trigger a read-out. For triggering, a TLU was used. The coincidence criterium was set to trigger a read-out for every hit in the scintillator. Data was collected via the Producer with help of EUDAQ's Run Control to test the Producer's functionality. It was later imported in EUTelescope to verify proper operation of the converter plug-in. In Figure 4.5, the hit map after the clustering step in EUTelescope

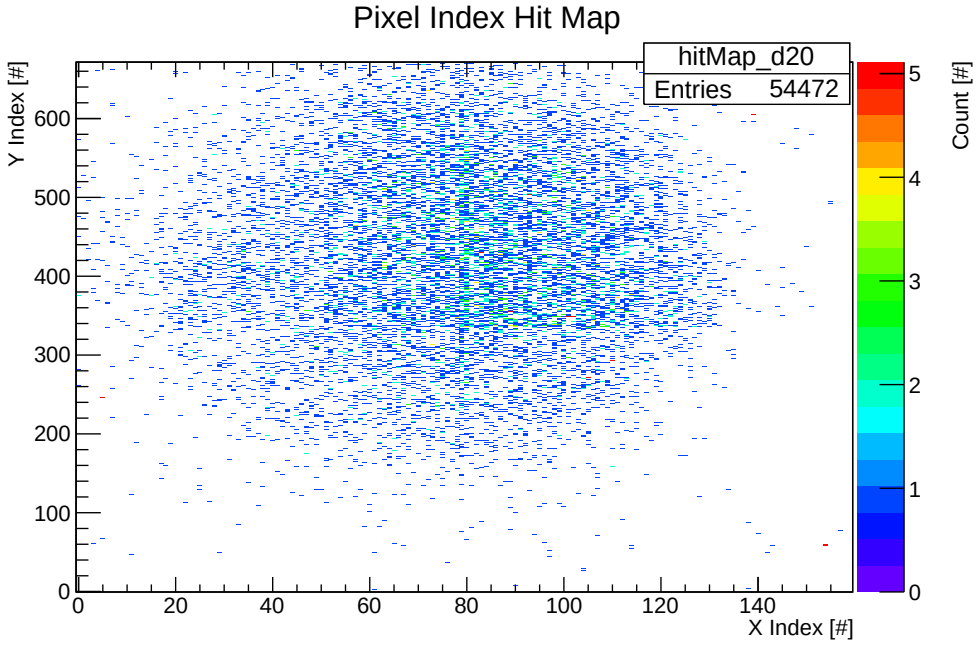

Figure 4.5.: Hits of a ^{90}Sr source on a four chip modul, using EUDAQ as the DAQ framework with a scintillator to issue triggers to the TLU. The USBpix converter plug-in was configured to read out a `modules[i] = 1,1,1,1` board.

can be seen. As the FEs were not tuned, FE III and IV (cf. Fig. 4.4) detect more hits than I and II. The increased hits in column 79 and 80 are due to the prolonged pixel implants in those pixels.

To test the functionality when multiple modules are attached to one burn-in card, the same four chip sensor set-up was used. However, the module entry was set to `modules[i]`

`= 1,1,2,2` or `modules[i] = 1,1,2,3` in the configuration. The resulting hit maps for the case of `modules[i] = 1,1,2,2` are shown in Figure 4.6 and 4.7.

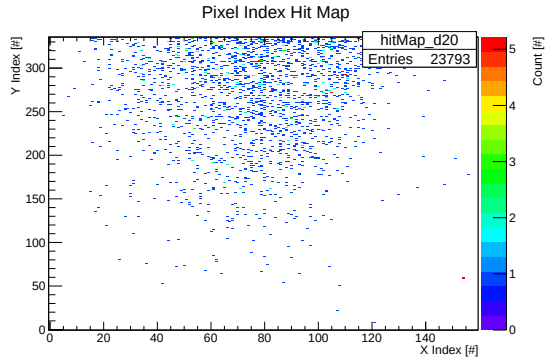

Figure 4.6.: Hit map after clustering for sensor 20, corresponding to FE I and II.

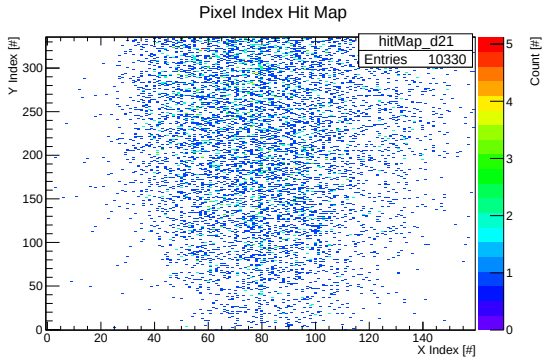

Figure 4.7.: The same for sensor 21, corresponding to FE III and IV.

The data is processed as expected. It can be seen that in comparison to the initial case, where they were treated as part of the four chip module, FE III and IV on sensor 21 are rotated. This is due to their rotated alignment on the four chip module (Fig. 4.4), which is ignored when only treated as a double chip. Aside from the source spot, the noisy pixels match their expected position. The same test has also been carried out with a configuration of `modules[i] = 1,1,2,3`. The results again match the expectation and can be reviewed in Appendix A.1.

5. EUTelescope

5.1. Introduction

The EUTelescope framework provides a collection of Marlin processors which can be adjusted to the needs of individual reconstructions and analyses. Figure 5.1 shows the sequence of a standard ATLAS pixel analysis in EUTelescope. Each of the steps represents one execution of Marlin with a well defined steering template. The steering template is an XML file which tells Marlin which processors have to be called and contains sections with configuration parameters for all those processors.

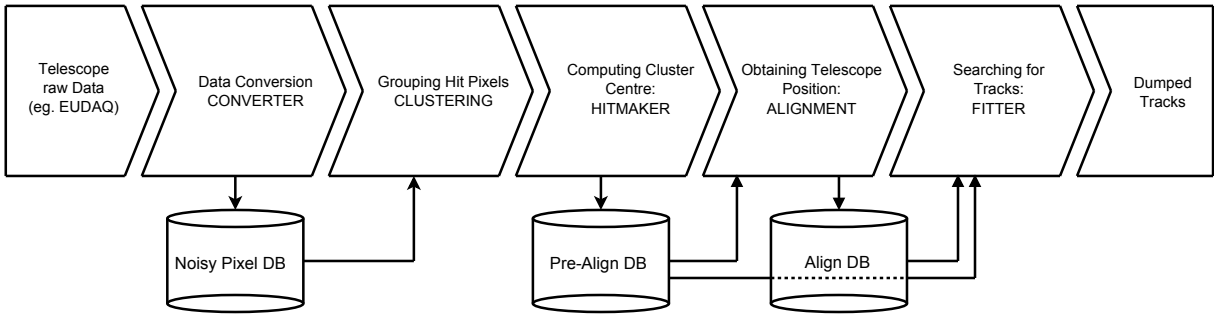

Figure 5.1.: Typical EUTelescope analysis for ATLAS pixel testbeams.

5.1.1. LCIO Data Format

The underlying data structure used is the LCIO (Linear Collider I/O) framework, it is again part of ILCSoft. A C++ as well as Java implementation (with corresponding APIs) exist and additionally, a Fortran interface to the C++ implementation.
It is important to understand that LCIO is a format on an event basis, meaning that all the information of one trigger belongs together in this format. This event data can be stored in various collections, the most important one being `TrackerData` which stores zero-suppressed raw data, the `TrackerPulse` used for storing clusters, the `TrackerHit` storing hits and ultimately the `Track` collections storing the final reconstructed tracks.

For detectors which have non-zero-suppressed data there is also the `TrackerRawData` collection, but all processors introduced during this work only deal with zero-suppressed data by convention. Therefore other means (e.g. some processing steps before the actual analysis) of converting the original data to zero-suppressed one have to be taken.
Each entry in a collection contains a 64 bit field, the `cellID` field, which can be used to encode additional information. Maintaining this 64 bit field is internally done, meaning that the user does not have to worry about little or big endian[1] encoding as well as differences in 32 and 64 bit architectures. The `cellID` field is used to store information on the detector ID (i.e. which detector plane the raw data, cluster or hit belongs to) as well as masking noise or other quality parameters. This is necessary due to the event-based structure of the format, where all data of one trigger is bunched together. Aside of the collections, every file also contains a header which stores various run related values, e.g. run number or time stamps.

5.1.2. Geometry API for Reconstruction (GEAR)

Via GEAR it is possible for users of EUTELESCOPE to provide geometric data on the telescope set-up. This does not only include the absolute positions and orientations of sensors, but also their geometric pixel layout.
For each plane in the telescope set-up, there is a `ladder` and `sensitive` node in the XML file. While this should in principle allow to specify a plane and the sensitive volume in it, this additional `ladder` information is not used in EUTELESCOPE.
GEAR itself will then parse this file and via a manager class provide this information globally throughout all parts of EUTELESCOPE. Individual MARLIN processors will then be able to obtain all this information via the specified ID in the GEAR XML file.

5.2. Previous way to interface LCIO Tracker Data in EUTelescope

EUTELESCOPE provides classes to interface the different collection types. The `TrackerData` stores all its information in a C++ `std::vector<float>`. The LCIO framework does not dictate in any way how information is stored in there, this is done by EUTELESCOPE. For this purpose EUTELESCOPE introduces classes representing pixels. Every

[1] Endianness is the convention of which byte is stored at which address in memory. In big endian the most significant byte is stored at the lowest memory address, whereas in little endian the least significant byte is stored at the lowest memory address.

pixel can for example contain three entries: one for the X index, one for the Y index and one for the signal recorded by that pixel. Each hit pixel therefore stores three values in the vector in a well defined order. When this information is later read back, this ordering is crucial and therefore also the pixel type used to encode this data is stored in the `cellID` field.
To easily read information from this collection EUTelescope provides a templated class for interfacing `TrackerData`, namely the `EUTelSparseDataImpl<T>`. The template parameter `T` is the pixel type, allowing to interface the data with any pixel type.
Clusters in EUTelescope use a `TrackerPulse` collection, the raw hits belonging to the cluster are additionally stored in a `TrackerData` collection. This means that also clusters need to be interfaced by a pixel class. There are several clustering algorithms, the sparse clustering algorithm is the most natural and thus will be discussed in more detail later on. For this algorithm, two templated classes `EUTelSparseClusterImpl<T>` and `EUTelSparseCluster2Impl<T>` had been introduced.
The other collections do not need an interfacing class because they do not have an object which needs to be interpreted in a certain way, contrary to the `std::vector<float>` in the `TrackerData`.
As mentioned, there are specific pixel classes used to interface hits. The most used pixel class is the `EUTelSimpleSparsePixel`, it is a description of a pixel with X and Y index as well as a discrete signal value.

ATLAS Pixel (APIX) Classes in EUTelescope

The APIX community introduced several new processors and several classes related to ATLAS pixel data processing. For that purpose the `EUTelAPIXSparsePixel` had been introduced replacing the `EUTelSimpleSparsePixel` for ATLAS data. It additionally contains fields for the time information of the pixel hit as well as the FE ID on which the hit took place. To process this additional data, several specific classes, some of them processors, have been introduced. For example a `EUTelAPIXSparseClusterImpl` was implemented as a cluster collection for APIX clusters.

Redesign of the Pixel Implementation in EUTelescope

As the specific APIX pixel and cluster classes required distinct subroutines in various processors, if not even individual processors, this introduced a lot of duplicated code which made refactorisation and maintenance a very tedious if not impossible task. As a consequence all this specific code was removed and the required functionality was fully merged into the already existing framework or generic new processors which can also be

used to operate on other data than APIX.
For this two new pixel types were introduced, the `EUTelGenericSparsePixel` replaces the

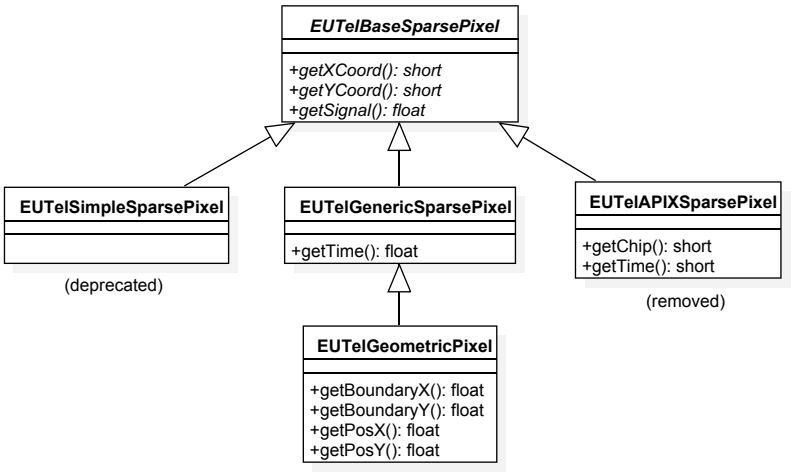

Figure 5.2.: Unified Modeling Language 2 (UML) class diagram for pixel classes in EUTelescope with their most important mutator methods to process the pixel's data.

`EUTelSimpleSparsePixel` as well as the `EUTelAPIXSparsePixel`, and can be considered as the basic pixel type in EUTelescope (Fig. 5.2). It maintains entries for the pixel indices (X,Y), the charge, as well as the hit time. Moreover, a pixel class used in the new geometric processors was introduced, the `EUTelGeometricPixel`. It inherits from the `EUTelSimpleSparsePixel` and has extra fields for a pixel's dimensions and position in space.
While `EUTelAPIXSparsePixel` was removed from the framework completely, the `EUTelSimpleSparsePixel` should be considered deprecated and not used anymore. It is merely there for backward compatibility and intended to be removed in future versions.

Redesign of Tracker Data interfacing and the Cluster Classes

Originally, different cluster classes could only be used with certain pixel types, as the interpretation of the `TrackerData` was done by the cluster class itself, rather than reusing any code. Therefore the `EUTelSparseDataImpl<PixelType>` class was refactorised and the `EUTelTrackerDataInterfacer` interface as well as the implementation `EUTelTrackerDataInterfacerImpl<PixelType>` were introduced as classes templated in the various pixel types (Fig. 5.3).
This allows clusters to be independent of their underlying pixel type. It is achieved via object composition, combining the previously mentioned tracker data interfacing class and

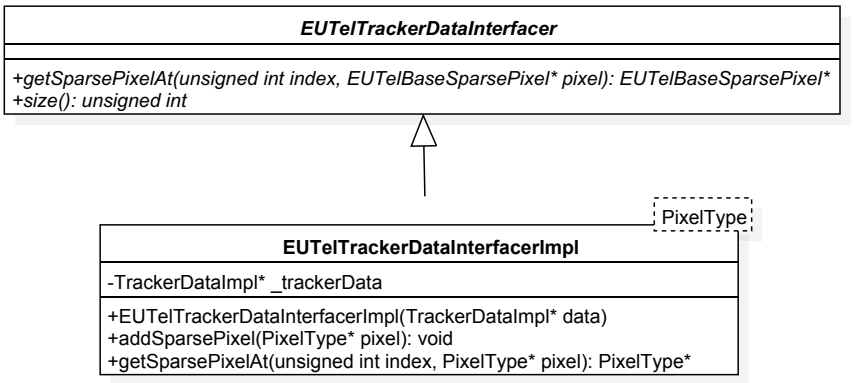

Figure 5.3.: UML 2 class diagram for the classes interfacing LCIO tracker data.

the cluster implementing class. The cluster implementation (implementing the `EUTelSimpleVirtualCluster` interface) is therefore also templated in the pixel type (Fig. 5.4).

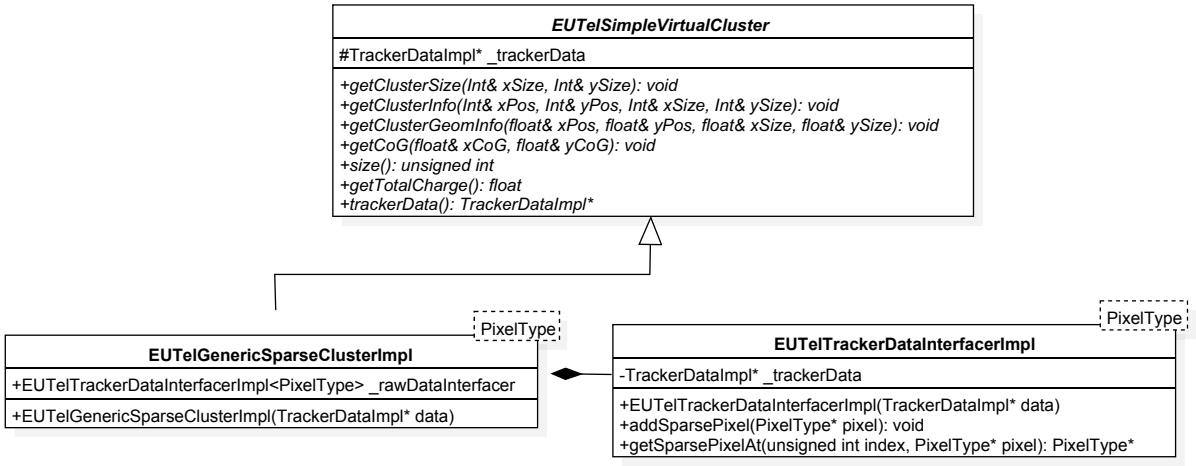

Figure 5.4.: UML 2 class diagram of the new cluster classes featuring a composition rather than an inheritance design approach.

The new cluster classes are designed to contain a collection of hit pixels which are considered to belong together. Unlike in the old approach there are now three key elements, all dedicated to one simple task: raw data interfacing with `EUTelTrackerDataInterfacerImpl<PixelType>`, cluster administration (`EUTelGenericSparseClusterImpl`) and cluster finding by the clustering processor itself. Originally, the cluster finding algorithm was implemented in the raw data interface and the clustering processor mainly called the specific method and stored the result in a cluster object.

Old Cluster Classes

While the old cluster classes have been updated to use the `EUTelTrackerDataInterfacerImpl<PixelType>`, they do not have the same superclass as the new cluster classes (`EUTelSimpleVirtualCluster`). This is due to the fact that their parent class requires implementation of signal to noise ratios which are ill defined for zero suppressed clusters as dealt with in this work.
The way to go would be to subclass from `EUTelSimpleVirtualCluster` and define an interface for non-zero-suppressed clusters. As this goes far beyond the scope of implementing a new pixel geometry, this has not been done. The lack of this unification results in the necessity for two different hit making processors and is one of the big open issues in the EUTelescope development.

5.3. The new Geometry Framework

An additional layer to the geometry description has been introduced in EUTelescope. An extension to Gear already existed for the upcoming General-Broken-Line (GBL) fitter, the `EUTelGeometryTelescopeDescription`. It uses ROOT's TGeo geometry package to create a representation of telescope planes which are used in GBL. As for GBL only the spatial material distribution was necessary, the whole sensor was represented by one box but no detail on the substructure was stored.
To achieve this, the functionality of `EUTelGeometryTelescopeDescription` was extended so it can store pixel layouts of sensors while retaining the functionality required by GBL. The `EUTelGenericPixGeoDescr` was introduced as a purely abstract class to be implemented for the various different pixel layouts, e.g. for an ATLAS four chip module or a Mimosa26 layout. There are two abstract methods which have to be implemented. The `void createRootDescr(char const * name)` as well as the `getPixName(Int X, Int Y)`. The first creates the corresponding nodes and volumes in ROOT's geometry framework[2], whereas the second method needs to return the name of the node corresponding to the pixel in the ROOT geometry model.
Additionally, the interface dictates to implement the inverse map of the above operation, retrieving a pixel's indices from a node. As this functionality is not yet used anywhere in the EUTelescope framework, it is not actively implemented in the included cases. The `EUTelGenericPixGeoMgr` is the responsible class to maintain and take ownership of the individual `EUTelGenericPixGeoDescr` objects, i.e. it will create, maintain and destroy

[2]For more information on this, refer to the chapter on geometry in the ROOT user's guide.

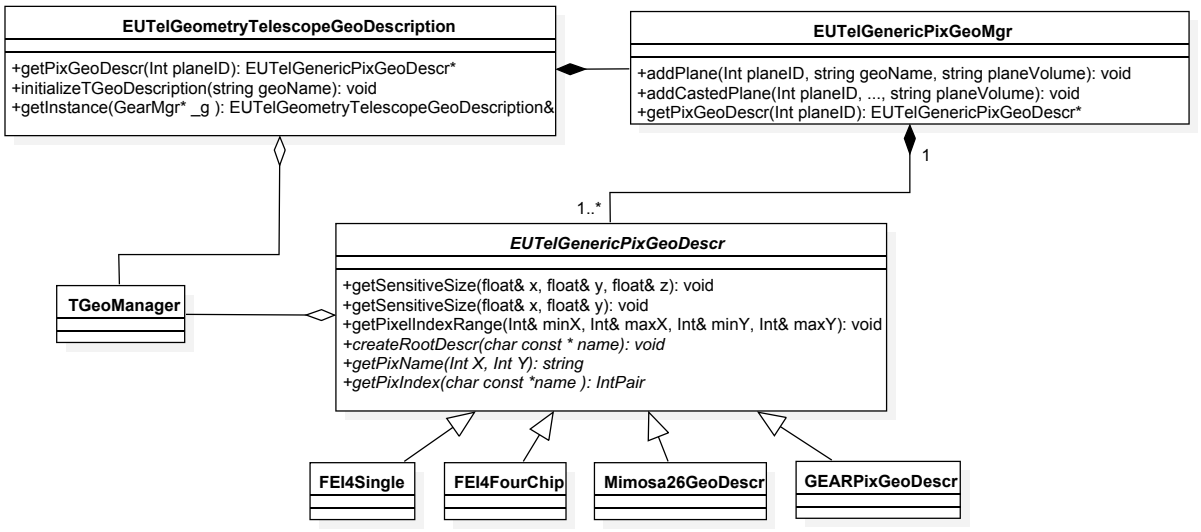

Figure 5.5.: UML 2 class diagram of the new geometry manager and classes connected to it.

them when necessary. A special implementation is the `GEARPixGeoDescr` which is used within the new framework when no other description is provided. It creates a sensor layout consisting of rectangular pixels as described by the GEAR file. While all other `EUTelGenericPixGeoDescr` objects could be considered as a stateless utility class, a utility design pattern was not possible as the GEAR wrapper description does have a state, governed by the GEAR description. Therefore the `EUTelGenericPixGeoMgr` was designed to reuse objects if the same geometry is loaded for multiple planes (Fig. 5.5).
Users developing EUTELESCOPE processor never need to work with the `EUTelGenericPixGeoMgr`. They interface the individual `EUTelGenericPixGeoDescr` via the `getPixGeoDescr(int planeID)` method of the `EUTelGeometryTelescopeDescription`.
An example of an implementation of a `EUTelGenericPixGeoDescr` is given in Appendix B. It is based on the ATLAS four chip module prototype layout.

5.4. Modified Reconstruction Chain

Due to the new geometry framework and cluster management, several processors were modified or newly introduced. This includes mainly processors in the early stage of reconstruction, as in the later alignment and fitting steps hit objects are used and their implementation remained unchanged. For detailed information on all new or heavily modified processors, refer to Appendix C.

5.4.1. The Converter Step

In this first step, the data is converted from its native format, which is unique to any DAQ systems, to the LCIO format. For this purpose there exists a MARLIN processor, namely the `EUTelNativeReader`. This processor is closely related to EUDAQ and uses certain EUDAQ classes. Every data associated to an event stored in EUDAQ has a specific type, which can be processed via so-called `DataConverterPlugin` managed by the `PluginManager` in EUDAQ. The `EUTelNativeReader`s can make use of that and use those converters to process the EUDAQ raw data and store them in LCIO files. This is the step in which the detector raw data is interpreted and converted.
Once the data is translated into the LCIO format, processors to mask noisy pixels are executed. Originally two processors existed, the `EUTelHotPixelKiller` and the `EUTelAPIXHotPixelKiller`. The APIX specific processor was necessary due to the different pixel implementation of APIX. The APIX processor was removed from the framework and the `EUTelHotPixelKiller` is deprecated. Profiling the CPU usage of `EUTelHotPixelKiller` revealed improper memory allocation leading to unnecessary memory reallocation in this processor. Both these processors have been replaced by the `EUTelProcessorNoisyPixelFinder`. The `EUTelProcessorNoisyPixelFinder` counts the firing frequency of each pixel, defined as the amount of events the pixel fired divided by the total event count and cuts on this value. Pixels above the threshold are considered noisy and written out in a separate database.

```
<execute>
   <processor name="AIDA"/>
   <processor name="UniversalNativeReader"/>
   <processor name="NoisyPixelMaskerM26"/>
   <processor name="NoisyPixelMaskerAPIX"/>
   <processor name="Save"/>
   <processor name="EUTelUtilityPrintEventNumber"/>
</execute>
```

Listing 5.1: Possible `execute` node of a converter steering template for telescope and APIX data.

In Listing 5.1 an example of which processors are called during the converter step is shown. The `AIDA` processor is responsible for histogram generation within the framework and typically called in every reconstruction step. Afterwards the converter and noisy pixel treatment processors are called. While there is only one call of the converter which deals with all the data, the noise treatment processor is called for APIX and telescope data individually. The `Save` processor saves the output LCIO file which will be read in

in the next step of the reconstruction chain. The `EUTelUtilityPrintEventNumber` will just print a message after having processed a certain amount of events to keep the user informed on the progress. It does not do any reconstruction work and can be omitted if this information is not required.

5.4.2. The Clustering Step

In this step, the previously created noisy pixel collections are read in. Explicit loading of the noisy pixel collection is necessary, since they are stored in an external LCIO file. After that the clustering processors are called. Originally two clustering processors existed, the `EUTelClusteringProcessor` and `EUTelAPIXClusteringProcessor`. While the APIX processor was removed, the `EUTelClusteringProcessor` was slightly modified. The algorithm for sparse clustering was initially implemented in two different ways, related to the `EUTelSparseClusterImpl<T>` and `EUTelSparseCluster2Impl<T>`. As this was modified, also the corresponding algorithms were updated. The duplicated implementation `EUTelSparseCluster2Impl<T>` was removed and the actual cluster finding routines were rewritten. `EUTelClusteringProcessor` provides multiple clustering routines for various types of data (zero-suppressed as well as non-zero-suppressed). This violates the design concept of EUTelescope, where processors should do simple and single tasks only and not be large, monolithic structures which cover every possible task.
For this reason the `EUTelProcessorSparseClustering` and `EUTelProcessorGeometricClustering` were introduced. The `EUTelProcessorSparseClustering` does sparse clustering for zero-suppressed data in the same way as the old clustering did. The `EUTelProcessorGeometricClustering` is also a type of sparse clustering, but uses the actual pixel volumes to search for adjacent hit pixels.

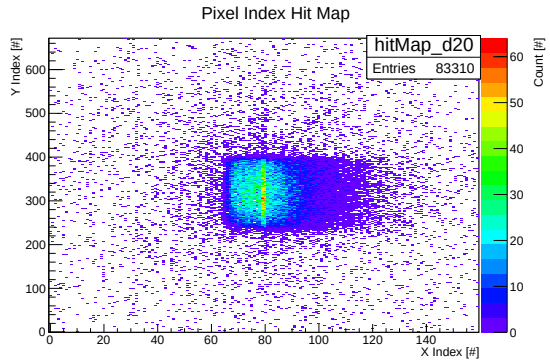

Figure 5.6.: Hit map in the space of pixel indices.

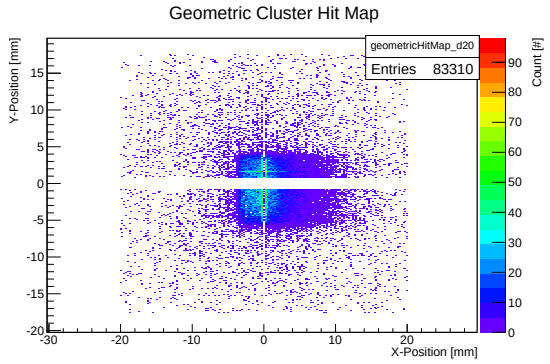

Figure 5.7.: Same hit map showing spatial cluster positions.

Figure 5.6 depicts a hit map from `EUTelProcessorGeometricClustering`, it shows the cluster position on the sensor in units of pixel indices, regardless of their actual spatial position. As the geometric clustering also obtains the actual spatial layout of the pixels, a hit map showing the spatial position on the sensor is also filled, it is given in Figure 5.7[3]. The given example hit maps are taken from the newly introduced Aconite-4chip example in EUTelescope and show an ATLAS four chip module. This explains the space between the two double chip modules as well as the increased hit rate in the prolonged pixels in the middle.
Any future ATLAS testbeam reconstruction should refrain from using the old clustering processor. For telescope data the `EUTelProcessorSparseClustering` should be used in addition to the `EUTelProcessorGeometricClustering` for more complex pixel layouts of the DUTs.

```
<execute>
   <processor name="AIDA"/>
   <processor name="LoadNoisyPixelDB"/>
   <processor name="ClusteringMimosa"/>
   <processor name="ClusteringAPIX"/>
   <processor name="NoisyClusterMaskerM26"/>
   <processor name="NoisyClusterMaskerAPIX"/>
   <processor name="NoisyClusterRemoverM26"/>
   <processor name="NoisyClusterRemoverAPIX"/>
   <processor name="Save"/>
   <processor name="EUTelUtilityPrintEventNumber"/>
</execute>
```

Listing 5.2: Possible EUTelescope processors executed in the clustering step.

After execution of the clustering processors, noise treatment processors are invoked (Listing 5.2). The noisy pixel treatment has been updated so that only dedicated noisy pixel processors care about noisy pixels, i.e. no other processor applies cuts on noisy pixels, but the cuts are applied prior to them. This is done by `EUTelProcessorNoisyClusterMasker` which masks any cluster containing a noisy pixel by setting a flag and `EUTelProcessor-NoisyClusterRemover` which removes masked clusters and provides a new collection of clusters which are noise-free.

[3]Note that the spatial histogram does not have any information on the exact pixel layout, therefore the binning does not correspond to pixels and binning artefacts can occur. As this is a control plot to check during reconstruction, this poses no problem but should be kept in mind.

5.4.3. The Hitmaker Step

In the hitmaker step actual hit positions on the sensor are derived from the created clusters (see Listing 5.3 for an example of MARLIN processors executed during the hitmaker step). As previously discussed, the new cluster classes require a different hitmaker, the `EUTelHitMakerTwo`. This processor simply retrieves all the hit pixels from a cluster object and uses their spatial information to derive the correct hit position. It exploits its knowledge of the pixel's dimensions, allowing for a modified determination of hit position compared to the simple COG approach as given in Eq. 1.6.

```
<execute>
   <processor name="AIDA"/>
   <processor name="HitMakerTwoM26"/>
   <processor name="HitMakerTwoAPIX"/>
   <processor name="Correlator"/>
   <processor name="PreAligner"/>
   <processor name="Save"/>
   <processor name="EUTelUtilityPrintEventNumber"/>
</execute>
```

Listing 5.3: Possible EUTELESCOPE processors executed in the hitmaker step.

Output of the hitmaker are hit objects, this is the point where all the different cluster implementations are merged into one common object used in the further reconstruction. In the hitmaking step also the physical orientation of the sensor is applied. That means if the sensor is mounted mirrored or the x and y direction are flipped, the hit is transformed into the global system. In Figure 5.8 the output of the Aconite-4chip example is given for sensor 21, an ATLAS FE-I4 single chip module. As the sensor is mounted in a way that the x axis is flipped, this is specified in the GEAR file and results in a hit map as given in Figure 5.9 in the global telescope coordinate system. Any shift would have also been applied in this step.
As hits have a spatial position associated to them, it is also possible to make a rough alignment by assuming that tracks are not inclined with respect to the normal direction of the telescope planes. This is done by the `EUTelCorrelator` and `EUTelPreAlign` processors. Examples of such correlation plots can be seen in Figure 5.10 and 5.11 for the four chip module of the Aconite-4chip example (sensor 20). It shows the correlation of hits (in the global telescope frame of reference) from the first telescope plane and from the investigated plane, in this case sensor 20. Again the characteristics of this layout can be seen, the prolonged pixels in the centre cause the two prominent entries in the x correlation histogram and the missing entries in the y one are due to the space between the two double chip modules.

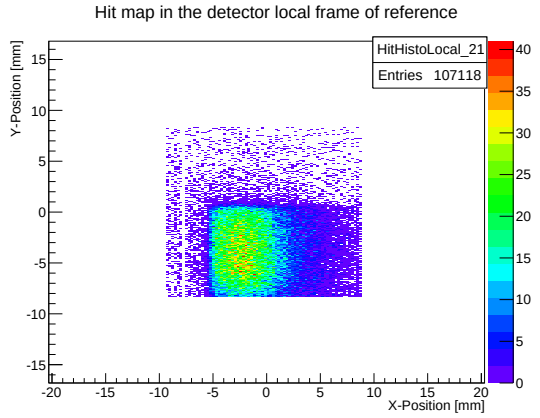

Figure 5.8.: Hit in the local frame.

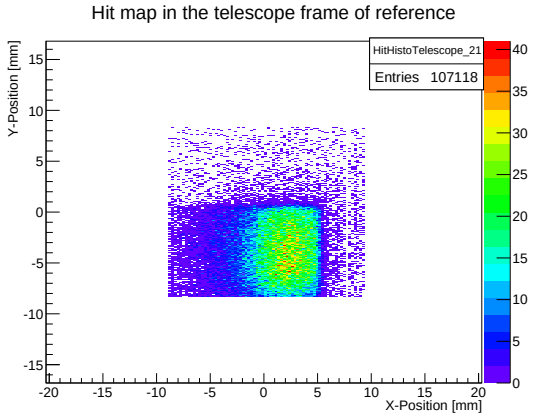

Figure 5.9.: Transformed hits into global frame.

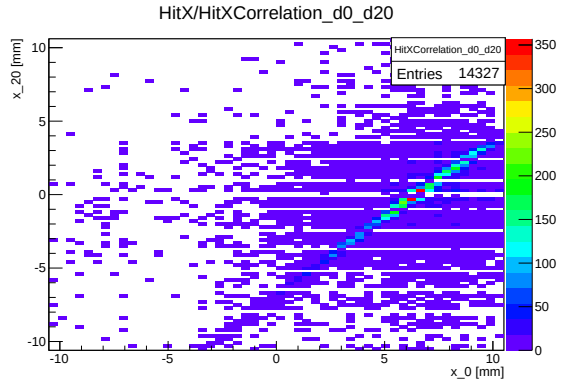

Figure 5.10.: x correlation plot for the four chip module in the Aconite-4chip example.

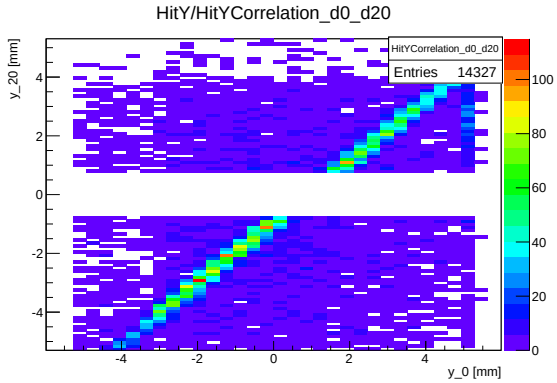

Figure 5.11.: y correlation plot for the same sensor in the same example.

Alignment corrections derived in this step are stored in a separate LCIO file. Alignment files are used in a bottom-up fashion, i.e. even when the final alignment is obtained, the pre-alignment derived in this step has to be applied initially.

5.4.4. Alignment and Track Fitting

The final two steps are alignment and track fitting. As they have not been changed during this work, they will be discussed only briefly.
Alignment is done by applying the previously obtained pre-alignment and then using the MillepedeII framework. This results in a second alignment database. As the hits remain unchanged, no new LCIO collection, aside from the alignment database, is created.
In the final reconstruction step, the track finding, all the alignment collections are applied and a pattern recognition algorithm is invoked on the aligned hit collections. In the case

of APIX the Deterministic Annealing Filter (DAF) fitter is used. A DAF fitter is an iterative Kalman filter with reweighted observations [43].

5.4.5. Track Dump

The final step, and actually part of the fitting MARLIN execution step, is exporting the obtained tracks in a way that TBmonII can process them. The processor is called `EUTelAPIXTbTrackTuple` and has been completely rewritten during this work and a new file format has been introduced. To differentiate the different file versions also a version number has been introduced, allowing TBmonII to correctly process old, new and even future revisions of the file format.
A major difference is that the undoing of the alignment is not done by this processor anymore, but hits have to be unaligned before by a separate processor. Undoing the alignment is necessary, as TBmonII was designed to obtain hits in the local frame of reference and thus does not need to know anything about the telescope set-up and alignment.

6. Results with the updated Version of EUTelescope

6.1. The Aconite-4chip Example

As already mentioned, an example case has been introduced in EUTelescope which uses all the new processors for a correct reconstruction. The example uses data obtained during the November 2013 planar pixel sensor testbeam of ATLAS at DESY. The set-up uses the Aconite telescope with all six telescope planes active and two DUTs in between. One is an ATLAS four chip prototype and the other a single chip module acting as a reference plane. The run chosen for this example is 1085, in which the beam spot was in the middle of the four chip module.

6.2. Geometry Verification Tests

To verify that the new geometry framework provides a correct description, its outputs have been compared to the old clustering. For this purpose, a Python program has been written to compare hits on an event basis.
These tests have been carried out on real testbeam data: one data set with a double chip module and one with a four chip module.

6.2.1. Double Chip Module Results

In x direction, the double chip features 2×79 pixels with 250 µm pitch, and two pixels with 450 µm pitch in the centre, yielding a total length of 4.04 cm. If all of the 160 pixels are assumed to be equally sized, they would have to be 252.5 µm long. This description via Gear was compared to the results obtained with the new geometry framework.
Positions in the old hitmaker are computed the following way:

$$\mathrm{pos}_i = (i + 0.5)\, p_i - \frac{s_i}{2} \tag{6.1}$$

Where i is the pixel index, p_i is the pixel pitch and s_i the total length of the sensor. As the first pixel index i starts with 0, this yields the correct pixel centre position with the origin located in the middle of the sensor.

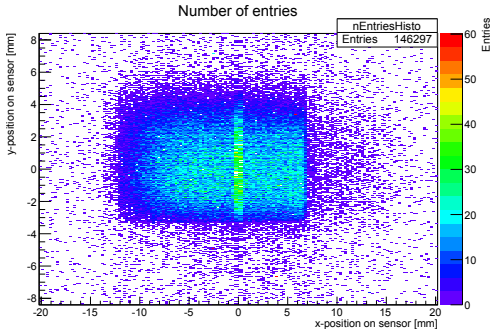

Figure 6.1.: Hit map of the investigated double chip.

As with the four chip modules, the prolonged pixels detect more hits due to their larger area, which is clearly visible in the hitmap of the investigated sample (Fig. 6.1). For hits in those pixels, one expects a difference in reconstructed hit position of half their pixel pitch, i.e. $\frac{450\,\mu\mathrm{m}}{2} - \frac{252.5\,\mu\mathrm{m}}{2} \approx 0.98\,\mathrm{mm}$, which can be seen in the distribution of $\Delta x = x_{\mathrm{geo}} - x_{\mathrm{old}}$ (analogous for Δy) shown in Figure 6.2 (and Fig. 6.3 for Δy).

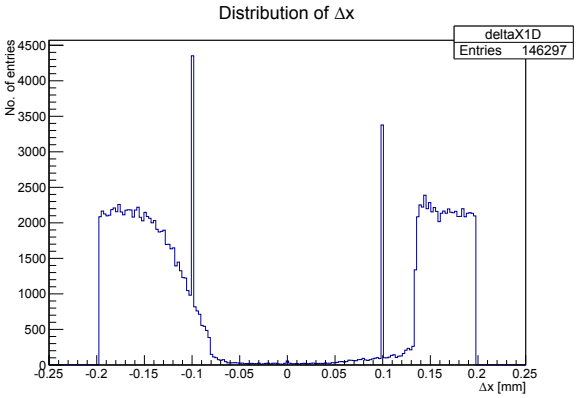

Figure 6.2.: Distribution of Δx for the double chip module.

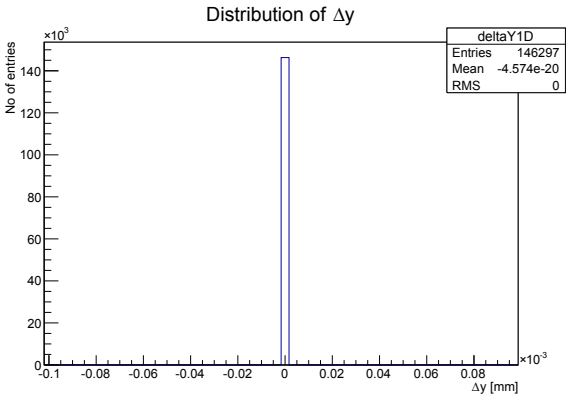

Figure 6.3.: Distribution for Δy for the same data.

While the Δy distribution features no deviations, the shape of the Δx meets the expectation: the difference in reconstructed hit positions is a linear function in pixel index where the slope equals the difference in pixel pitch, i.e. $\Delta p_x = (250 - 252.5)\,\mu\mathrm{m} = -2.5\,\mu\mathrm{m}$

per pixel. This linear offset makes a jump in the centre due to the prolonged pixels present there. The maximum offset is thus for hits in pixel 79, leading to a deviation of $79p_x = 0.1795\,\text{mm}$. The hitmap (Fig. 6.1) reveals a rough cut-off due to the scintillator active area for positive x, whereas in the negative region the beam spot is more washed out, leading to a rough edge in the Δx distribution (Fig. 6.2) for positive values and a more continuous drop for the negative ones. Moreover, the fact that the beam spot is not centred on the sensor can be seen.

To quantify the behaviour of the offset, a 2D histogram showing the correlations of Δx and x_{geo} is shown in Figure 6.4 (and the same for y as shown in Figure 6.5). The deviation is expected to increase as: $\Delta x = (i + 0.5)\Delta p_x$. Combined with Eq. 6.1 this yields:

$$\Delta x = \frac{\Delta p_x}{p_x} x + \frac{s_x}{2}\frac{\Delta p_x}{p_x} \tag{6.2}$$

p_x being the pitch of the correct description, i.e. 250 µm. This yields the following values:

$$\frac{\Delta p_x}{p_x} = -0.01 \tag{6.3}$$

$$\frac{s_x}{2} = 20.2\,\text{mm} \tag{6.4}$$

The correlation plots given in Figure 6.4 and 6.5 exhibit the behaviour discussed above. To quantify the linear slopes, a profile histogram was derived and two fits ($y = p_0 + p_1 x$) were performed in two regions Ri: $[-20.2, -0.7]$ and $[0.7, 20.2]$.

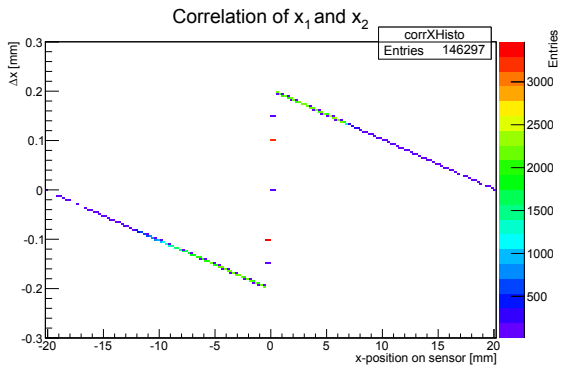

Figure 6.4.: Correlation of Δx and x_{geo} (the actual position on the sensor).

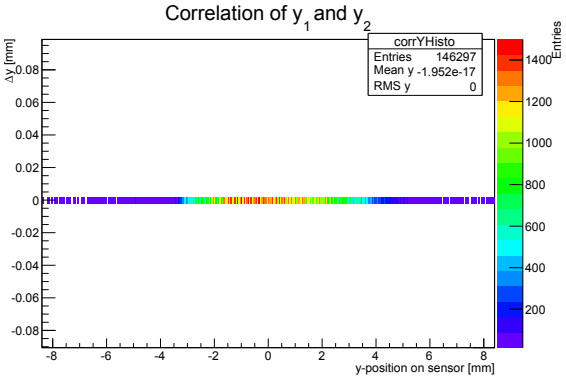

Figure 6.5.: Same correlations as in Fig. 6.4 for y.

The obtained values were:

Region R1:

- offset = (-0.2021 ± 0.0009) mm
- slope = -0.0100 ± 0.0001

Region R2:

- offset = (-0.2019 ± 0.0009) mm
- slope = -0.0099 ± 0.0001

These values agree with the derived expectations of $\text{offset}^{\text{exp}} = 0.202$ mm and $\text{slope}^{\text{exp}} = 0.01$. Additionally, Δx has been determined space-resolved on the sensor. The resulting histogram is given in Figure 6.6[1]. As expected, Δx increases towards the centre of the chip, where it makes a jump and no dependence on the y position can be seen.

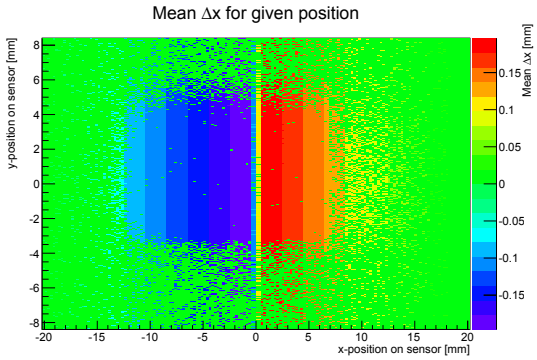

Figure 6.6.: Spatially resolved Δx for the double chip module analysis.

Some features of the Gear implementation of the pixel geometry are surprising: while the pixel pitch p_i, the pixel count n_i and the total sensor size s_i are trivially related, $s_i = p_i \times n_i$, this relation can be violated in the Gear description. As double chip modules mostly consist of 250 µm pixels, it is very tempting to use this value for the pixel pitch, but increase the total sensor size to 40.4 mm, taking into account the prolonged pixels. As the position is reconstructed according to Eq. 6.1, this will retrieve the correct position for any pixels (except the prolonged one) on the first sensor, but lead to an offset corresponding to twice the additional length of the prolonged pixels, i.e. 0.4 mm for any pixels on the second sensor. Performing the same analysis reveals this behaviour. In addition to the two peaks at 0 and 0.4 mm, small peaks corresponding to hits in the prolonged pixels at 0.1 and 0.3 mm are present in the Δx distribution (Fig. 6.7). The spatial and correlation plots exhibit the expected structure, featuring the jump on the second chip, as can be seen in Figure 6.8 and 6.9.

[1]The deviations are only computed for pixels which contain at least one hit. Therefore, the shadow of the beam spot can be seen, and only pixels within feature a deviation different from 0. This is the case for all 2D plots showing Δx (and Δy).

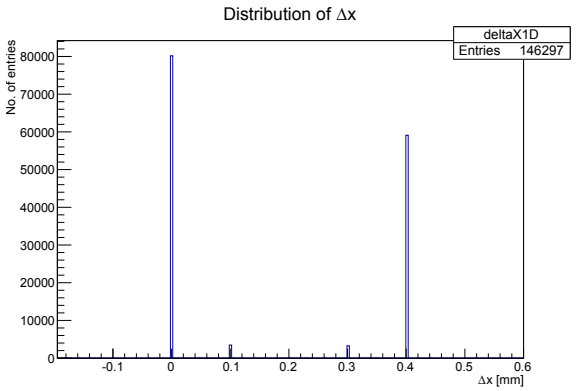

Figure 6.7.: Δx distribution for the modified reconstruction with 250 µm pitch and increased sensor size.

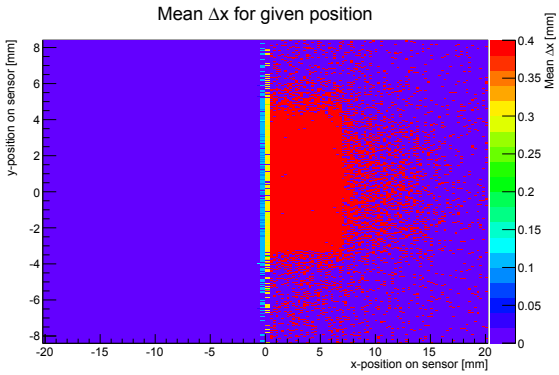

Figure 6.8.: Mean Δx spatially resolved on the sensor.

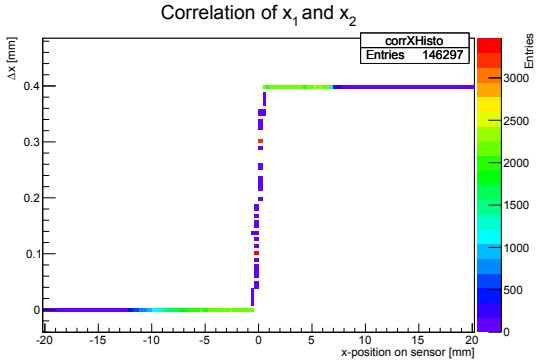

Figure 6.9.: Correlations of Δx and x_{geo}.

6.2.2. Four Chip Modules

The same analyses were also performed for the Aconite-4chip example. As the layout features a 1.58 mm gap between the two double chip modules, a deviation in reconstructed y position was also expected.

The Δx distribution (Fig. 6.10) features the same behaviour as in the double chip module case (Fig. 6.2), which is expected, as a four chip module consists of two double chip modules and thus features the same layout in x direction. In y direction, the same shape of the distribution (Fig. 6.11) is expected without the peaks caused by the prolonged centre pixels. Again, no dependence on the other direction is observed if looking at the spatially resolved mean Δx and Δy (Fig. 6.12 and 6.13).

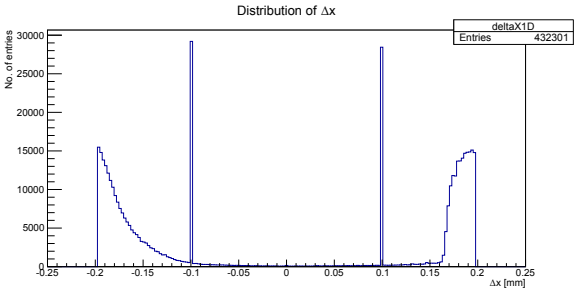

Figure 6.10.: Δx distribution for the four chip module data.

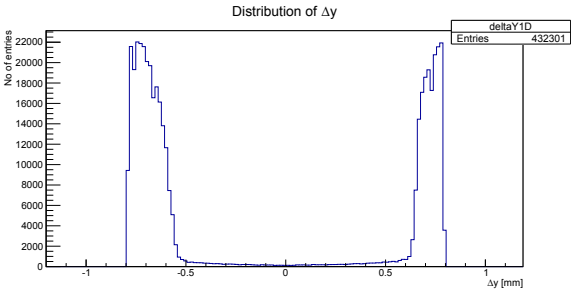

Figure 6.11.: Δy for the same data set.

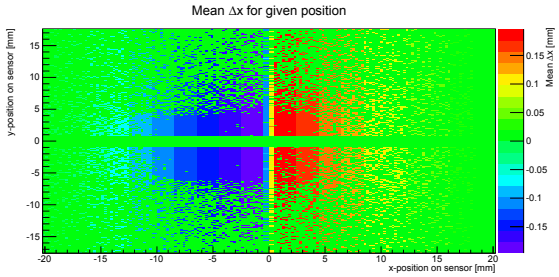

Figure 6.12.: Spatially resolved Δx.

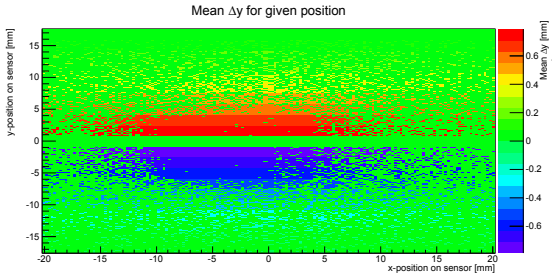

Figure 6.13.: Spatially resolved Δy.

6.3. Test-Driven Development

To verify the functionality of EUTelescope, the most recent version is automatically downloaded, compiled and test cases are executed. As several new Marlin processors were introduced, they had to be included in the test builds. As the Aconite-4chip example was made publicly available, the newly introduced test cases for the new processors are based on this example.

All the reconstruction steps in the example are executed and their proper completion is checked. The results of these tests are uploaded to a web-based testing sever and the development team is informed if any of them fail. As they run on a daily basis and a history of their status is recorded, it is easy to track down a failed test to a specific date and thus to corresponding changes in the software framework at that time. This leads to an easier and more efficient maintenance of the whole EUTelescope software package.

6.4. Toy-Box Straight Line Simulation

A small straight line simulation has been written in pyLCIO. It simulates six telescope planes and two standard ATLAS FE-I4 layouts (without any prolonged edge pixels) which are perfectly aligned with respect to each other. Straight lines are fitted through the

telescope and the hit pixel is geometrically determined. In this simple model, this results in a single hit cluster in exactly this pixel.
One track is simulated per event, each telescope sensor thus detecting exactly one hit pixel per event, as they are assumed to be 100% efficient. For the first ATLAS plane, different pixels had different efficiencies. For every hit, a random number was generated and depending on the efficiency, the hit was written into the data stream or discarded. The base efficiency was set to 0.8, i.e. unless otherwise specified, this was the global efficiency. Pixels were selected to form the TBmonII logo and given an efficiency of 100%. Furthermore, a row under the TBmonII logo was given an efficiency gradient from 0% to 100%. To further probe the reconstruction resolution, pixels for which both indices modulo 30 are equal (i.e. $(x \mod 30) = (y \mod 30)$), were given an efficiency of 0.5.

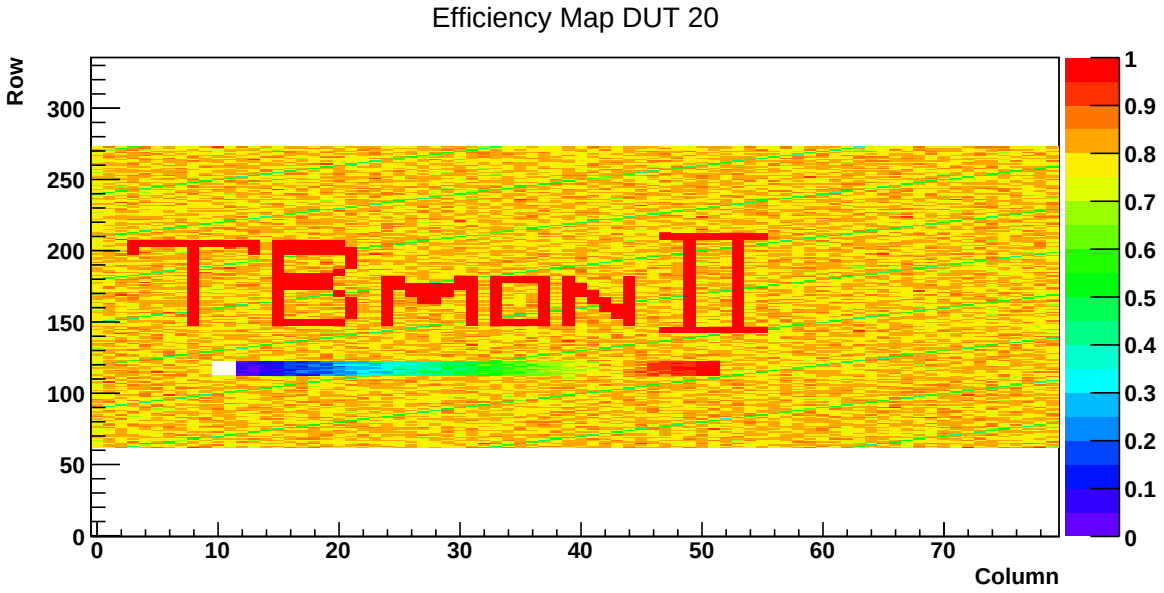

Figure 6.14.: Efficiency map as obtained from TBmonII for the simulated straight line data.

The obtained data was reconstructed via the new geometric clustering processor and the new hitmaker. Ultimately, the data was dumped via the new track dump processor and imported into TBmonII, where it was analysed. The resulting efficiency map is shown in Figure 6.14 and exhibits the introduced characteristics as discussed before.

7. Summary and Outlook

7.1. Summary

During this work, the analysis framework for the ATLAS pixel testbeams has been updated and improved in several aspects. Data acquisition with the USBpix framework has been updated to be able to process data with burn-in adapter cards, allowing to read out up to four FEs with a single MultiIO board. This is especially important for data acquisition with the ATLAS four chip modules. For this purpose, the EUDAQ Producer in STControl was modified and the desired new functionality implemented.
Accompanying these changes, the EUDAQ converter plug-in was updated. Multiple old converter plug-ins were merged into a new, refactorised one. The new converter plug-in is able to process data provided by the updated USBpix Producer, but is completely downward compatible to old data at the same time. Correct operation of the updated DAQ software was verified in the laboratory with a radioactive source.
The reconstruction framework EUTelescope was improved to support more arbitrary pixel geometries, allowing to reconstruct ATLAS double and four chip modules without the need for error prone workarounds. This was achieved by introducing a new pixel geometry framework and expanding the EUTelescope geometry manager, which is based on ROOT's geometry libraries. The pixel geometry manager was implemented with respect to possible further applications, especially providing the possibility for Geant4-like navigation, allowing the geometry to be used in other applications, e.g. in external simulation toolkits or for geometry-based tracking.
To fully exploit the new functionality of the pixel geometry manager, the pixel and cluster implementation in EUTelescope has been reworked, leading to multiple new Marlin processors. Aside from the newly introduced clustering based on geometric proximity, the default clustering processor has been updated and made easier to use. Additionally, a new noise treatment scheme and corresponding processors have been written. The processor interfacing EUTelescope and TBmonII, the track dumper, has been reimplemented to be able to deal with the new classes.
To verify and provide a test case, the Aconite-4chip example has been introduced in EU-

Telescope, giving users a starting point if they want to base their reconstruction on the new processors. This example is also included in the nightly test builds, giving developers immediate warning in case of failure within the new processors.
The geometry description was verified by reconstructing ATLAS double and four chip modules and comparing reconstructed hit positions on an event basis. Hits were reconstructed with an old version of EUTelescope as well as the current version. The resulting deviations can be understood by the various limitations and peculiarities of the old Gear pixel layout description.
Finally, a small straight line track simulation was written for the telescope with the aim to verify the new geometric hit processors, as well as the interface between EUTelescope and TBmonII.

7.2. Outlook

7.2.1. USBpix

The USBpix Producer has been updated to support read-out of the burn-in cards in testbeam operation. Data from each FE is read out in a separate thread (the multiple data producing threads) and processed by the EUDAQ Producer (the single consumer thread). This poses a classical multi-producer, single-consumer multi-process synchronisation problem. Currently, this problem is poorly solved as the producer threads will be put to sleep (i.e. keep the BUSY line raised) until the consumer thread is finished. Data is currently passed via Qt events from the producer threads to the consumer, and a global boolean is used for interprocess synchronisation. Using a first in, first out (FIFO) buffer and semaphores for synchronisation, this dead time could be decreased to a minimum, allowing faster DAQ at testbeams.

7.2.2. Geometry

As the implemented pixel geometry manager was kept as generic as possible, it would be possible to reuse it in other programs, for example in TBmonII, which would result in one pixel layout description during the whole testbeam reconstruction and analysis. Furthermore, it would be advantageous if various other processors in EUTelescope would exploit the features provided by the updated telescope geometry manager, as it supplies easier implementation of geometry related functionality.
EUTelescope needs to be updated to use one central alignment file, which would make exporting the data to TBmonII much easier, since only one alignment file has to be

undone. In particular, this operation would take minimal implementation effort when using the ROOT geometry libraries, as this transformation is provided by the library.

7.2.3. Simulation

Continuing the idea of the straight line track simulation which was implemented to verify the very basic functionality of the newly introduced processors, this idea could be expanded and developed into a more sophisticated simulation. This would allow to verify and test various processors, especially the ones concerning alignment and track fitting. By way of example, testbeam studies aiming at high track rates, to investigate the properties of the tested DUTs under more realistic conditions, would become possible. Adapting the tracking algorithms to match or approximate the ones used in ATLAS would allow to conduct not only hit inefficiency studies, but to exploit the telescope as a vertexing device, operating it similarly to the real pixel detector and performing tracking studies. As such changes must be verified, a proper simulation toolkit becomes necessary. Initial implementations already exist in the *AllPix Simulation Framework*, which is a Geant4 based simulation with the main focus on the implementation of the various digitisers. Expanding such a framework to a general telescope simulation could help improve track reconstruction and enable future studies of tracking in general.

A. STControl Producer and EUDAQ

A.1. Verification with a Four Chip Module

The case for `modules[i] = 1,1,2,3` has been tested in the laboratory, the same way as discussed in Section 4.3. The resulting plots from EUTelescope match their expectation completely.

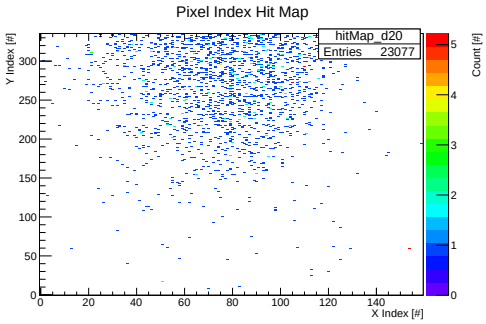

Figure A.1.: Hits on sensor 20, corresponding to a double chip, consisting of FE I and II of the four chip module.

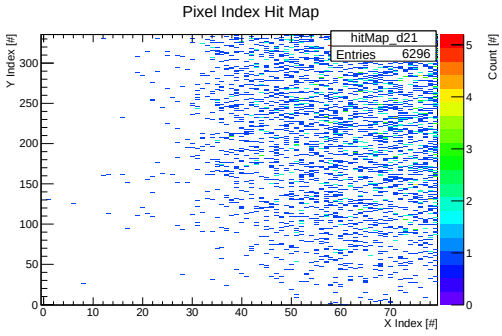

Figure A.2.: Hits on sensor 21, corresponding to a single chip (FE III).

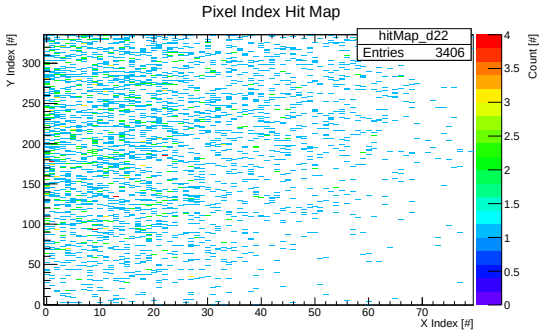

Figure A.3.: Hits on sensor 21, corresponding to a single chip (FE IV).

B. Implementation of a EUTelGenericPixGeoDescr - An Example

The ATLAS four chip module used in this example consists of two double chip modules. Each double chip module has twice 79 normal sized pixels and two prolonged in the middle. The exact layout is depicted in Figure 3.3. Therefore the layout in TGeo is based on two double chip modules.

```
1  FEI4FourChip::FEI4FourChip(): EUTelGenericPixGeoDescr(  40.4, 35.18, 0.025,      //size X, Y, Z
2                                                           0, 159, 0, 671,         //min max X,Y
3                                                           93.660734 )             //rad length
4  {
5          //Create the material for the sensor
6          matSi = new TGeoMaterial( "Si", 28.0855 , 14.0, 2.33, _radLength, 45.753206 );
7          Si = new TGeoMedium("FEI4Silicon",1, matSi);
8
9          /* Make a box for the sensitive area
10         MakeBox takes the half of size in mm as arguments */
11         plane = _tGeoManager->MakeBox( "sensarea_fei4four", Si, 20.2, 17.59, 0.0125 );
12
13         //Create volumes for the double chip
14         TGeoVolume* doublechip = _tGeoManager->MakeBox("fei4double",Si, 20.2, 8.4, 0.0125);
15
16         //Create volumes for the different regions
17         TGeoVolume* normalRegion = _tGeoManager->MakeBox("fei4normreg",Si, 9.875, 8.4, 0.0125);
18         TGeoVolume* centreRegion = _tGeoManager->MakeBox("fei4centreg",Si, 0.45, 8.4, 0.0125);
19
20         //Divide the regions to create pixels
21         TGeoVolume* normalCol = normalRegion->Divide("col", 1 , 79, 0 , 1, 0, "N");
22         normalCol->Divide("pixel", 2 , 336, 0 , 1, 0, "N");
23         TGeoVolume* centreCol = centreRegion->Divide("col", 1 , 2, 0 , 1, 0, "N");
24         centreCol->Divide("pixel", 2 , 336, 0 , 1, 0, "N");
25
26         //And place them to make a doublechip
27         doublechip->AddNode(normalRegion, 1, new TGeoTranslation(-10.325,0,0));
28         doublechip->AddNode(centreRegion, 1);
29         doublechip->AddNode(normalRegion, 2, new TGeoTranslation(10.325,0,0));
30
31         //Place two double chips for a four chip module
32         plane->AddNode(doublechip, 1, new TGeoTranslation(0,-9.19,0));
33         plane->AddNode(doublechip, 2, new TGeoTranslation(0,9.19,0));
34 }
35
36 void FEI4FourChip::createRootDescr(char const * planeVolume)
37 {
38         //Get the plane as provided by the EUTelGeometryTelescopeGeoDescription
39         TGeoVolume* topplane =_tGeoManager->GetVolume(planeVolume);
40         //Finally add the sensitive area to the plane
41         topplane->AddNode(plane, 1);
42 }
```

Listing B.1: Constructor and implemented `createRootDescr(...)` method of an ATLAS four chip module geometry class.

The whole geometry is prepared in the class' constructor. The `plane` will be the `TGeoVolume` holding a representation of the whole geometry layout (Listing B.2, Line 11). The layout itself is composed of double chips (`doublechip`) (L. 14) which by itself is made out of two `normalRegion`, i.e. the regions with 79×336 normal pixels, separated by a `centreRegion` (L. 17.18). The `Divide` calls divide the volumes in a GEANT 3 compliant syntax (L. 21-24). After preparing all the individual components, they are placed to form the double chip (L. 27-29) and finally two `doublechip` are placed into the plane to create the final `plane` (L. 32-33).

```
1  std::string FEI4FourChip::getPixName(int x, int y)
2  {
3          char buffer [100];
4          int doublechip = 1;
5          //If y index is larger than 335 then we are on double chip two (the upper one)
6          if(y > 335)
7          {
8                  doublechip = 2;
9                  y -= 336;
10         }
11         if (x < 79 )
12         {
13                 //in normalRegion_1
14                 snprintf( buffer, 100,
15                 "/sensarea_fei4four_1/fei4double_%d/fei4normreg_1/col_%d/pixel_%d",
16                 doublechip, x+1, y+1);
17         }
18         else if ( x == 79 )
19         {
20                 //in centreRegion_1, col_1
21                 snprintf( buffer, 100,
22                 "/sensarea_fei4four_1/fei4double_%d/fei4centreg_1/col_1/pixel_%d",
23                 doublechip, y+1);
24         }
25         else if( x == 80 )
26         {
27                 //in centreRegion_1, col_2
28                 snprintf( buffer, 100,
29                 "/sensarea_fei4four_1/fei4double_%d/fei4centreg_1/col_2/pixel_%d",
30                 doublechip, y+1);
31         }
32         else
33         {
34                 //in normalRegion_2
35                 snprintf( buffer, 100,
36                 "/sensarea_fei4four_1/fei4double_%d/fei4normreg_2/col_%d/pixel_%d",
37                 doublechip, x-80, y+1);
38         }
39
40         //Return the full path
41         return std::string( buffer );
42 }
```

Listing B.2: Implementation of the `getPixName(...)` method for the example.

This layout results in the given name of the node, representing an individual pixel. It is a file-system-like string, the directories being the different nodes placed into each other. Each pixel node will start with `sensarea_fei4four_1` as this is the name of the uppermost node `plane`. The first subnode will be the one representing the double chip, i.e. `fei4double_i` where `i` is either 1 or 2 for the first or second placed double chip. Afterwards it has to be differentiated if the pixel is in the prolonged region (`fei4centreg_1`) or in the normal region (`fei4normreg_i`). Ultimately the correct `col` and `pixel`, according to the divisions, has to be chosen. This way a given pixel index can be translated into a node, as done by the `getPixName()` method implemented in Listing B.2.

C. Newly introduced Marlin processors in EUTelescope

C.1. Noise Treatment Processors

EUTelNoisyPixelFinder

This processor is key to the noisy pixel treatment in EUTelescope, as it creates the databases of pixels which are considered to be noisy. For this purpose it is intended to be called right after data conversion. It counts the amount of registered hits for each pixel over a subset of all events and then normalises it to the subset size. A cut is applied to this overall hit detection frequency, pixels which detect a hit more often than this cut value are considered to be noisy and appended to the created noisy pixel collection.
It should be noted that EUTelescope has no way of knowing anything about the active time of a sensor, thus if working with a sensor which is only active during a small time window during a full telescope read-out, the cut hit frequency will naturally be lower. The processor's parameters can be reviewed in Table C.1.

EUTelNoisyClusterMasker

The `EUTelNoisyClusterMasker` operates on the output of any clustering processor and is the second step in noisy pixel treatment. It requires a noisy pixel collection as an input and then will mask any clusters in the LCIO pulse collection which contains a noisy pixel as defined by the noisy pixel database. It will do this by modifying the original collection, setting a flag for pulses which belong to a noisy cluster, therefore no new output collection is created. The noisy clusters are still present in the collection. Processor parameters can be reviewed in Table C.2.

Variable Name	Description	Variable Type	Default Value
`ZSDataCollectionName`	LCIO collection from EUTelNativeReader containing raw hits	`string`	"zsdata"
`NoOfEvents`	Size of event subset which will be used in this processor	`int`	"100"
`SensorIDVec`	Sensor IDs of detectors for which the processor should run	`vector<int>`	empty
`MaxAllowedFiringFreq`	The hit detection cut off value	`float`	0.2
`HotPixelDBFile`	Name of the LCIO output file containing the noisy pixel database	`string`	"hotpixel.slcio"
`ExcludedPlanes`	Sensors which appear in the input collection (`ZSDataCollectionName`) but should be excluded	`vector<int>`	empty
`HotPixelCollectionName`	Name of the LCIO collection in the output file	`string`	"hotpixel"

Table C.1.: Parameters of `EUTelNoisyPixelFinder`.

Variable Name	Description	Variable Type	Default Value
`InputCollectionName`	LCIO collection from clustering processor containing cluster pulses	`string`	"cluster"
`HotPixelCollectionName`	Name of the loaded noisy pixel database	`string`	"hotpixel"

Table C.2.: Parameters of `EUTelNoisyClusterMasker`.

EUTelNoisyClusterRemover

The final step of noise treatment is the creation of a noise free cluster collection. For this purpose `EUTelNoisyClusterRemover` is used, who will read in a previously masked (by `EUTelNoisyClusterMasker`) cluster pulse collection and create a new collection with only

cluster pulses which have not been flagged as noisy. The new cluster pulse collection will link individual pulses against the original underlying raw `LCIO::TRACKERDATA` (usually named `original_zsdata`), this collection must therefore not be deleted. A summary of processor parameters can be found in Table C.3.

Variable Name	Description	Variable Type	Default Value
`InputCollectionName`	LCIO collection from clustering processor containing cluster pulses in which noisy clusters have been flagged	`string`	"noisy_cluster"
`OutputCollectionName`	Name of the new, noisy cluster free cluster pulse collection	`string`	"noise-free_clusters"

Table C.3.: Parameters of `EUTelNoisyClusterRemover`.

C.2. Clustering Processors

EUTelProcessorSparseClustering

The `EUTelProcessorSparseClustering` processor replaces the old sparse clustering algorithm in `EUTelClusteringProcessor` for zero-suppressed data. Unlike the old `EUTelClusteringProcessor` it does not incorporate any noisy pixel treatment, which therefore has to be done afterwards with the newly introduced noisy pixel treatment scheme and processors. In addition to all other functionality it also allows to apply a time cut, given that your sensor provides this information and it is included in EUTelescope.

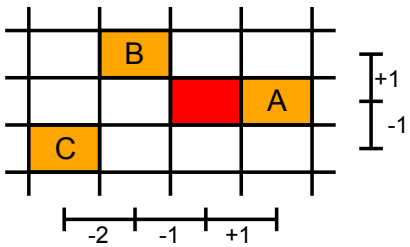

Figure C.1.: Sparse Clustering Distance.

Along with the temporal cut, which is the maximum difference in detector specific time units, the spatial cut is the only user tunable value. It is given in distance of pixel indices squared. Figure C.1 illustrates how this is understood. The distance between the red pixel and the pixel marked with **A** is 1 in horizontal direction and 0 in vertical, the squared distance is therefore $1^2 + 0^2 = 1$. Pixels touching at one corner (case **B**) have a distance of 1 in both directions, the sparse distance squared is therefore $1^2 + 1^2 = 2$. Case **C** has a distance of 2 in horizontal direction and 1 in vertical, giving a squared sparse distance of $2^2 + 1^2 = 5$.
Thus for the typical case of clustering the red pixel with **A** and **B**, the cut value has to be set to 2. If it were set to 1, only pixel **A** would be considered to belong to a cluster with the red pixel. Since the most natural requirement is pixels to be touching at least at one corner, the default value is set to 2. A summary of important parameters is given in Table C.4.

Variable Name	Description	Variable Type	Default Value
`ZSDataCollectionName`	Input raw data collection name	`string`	"zsdata"
`PulseCollectionName`	Name of output pulse collection storing clusters	`string`	"cluster"
`TCut`	Maximum time difference in detector specific time units, for hits belonging to the same cluster	`float`	max(`float`)
`HistogramFilling`	Switch for turning histograms on or off	`bool`	`true`
`ExcludedPlanes`	Planes to be excluded from the clustering	`vector<int>`	empty
`SparseMinDistance-Squared`	The maximum distance in pixel units, two hits are allowed to be apart for them to belong to the same cluster, squared	`int`	2

Table C.4.: Parameters of `EUTelProcessorSparseClustering`.

EUTelProcessorGeometricClustering

The geometric clustering processor is very much like the sparse one, with the major difference that the proximity requirement for pixels is not based on their indices, but arises from the geometric spatial mapping on the sensor. Consider the situation shown in Figure C.2 where the regular pixel pattern is interrupted by a long pixel composed of two normal pixels. Using the index scheme shown in the schematic, the long pixel would not be clus-

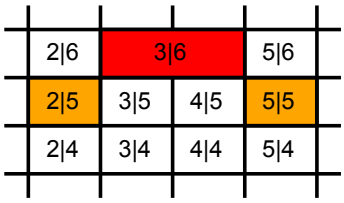

Figure C.2.: Geometric Clustering Proximity.

tered together with the pixel on the right, adjacent to it with a default sparse clustering, assuming a distance cut of 2, which corresponds to the usual touching requirement. This is due to the limitation imposed by assuming a coherent pixel pattern all over the sensor. Other cases where sparse clustering fails, include sensor layouts where there is some inactive material between two pixels. In this case, sparse clustering will wrongly assume that pixels aside of that area are adjacent, despite them being not.
`EUTelProcessorGeometricClustering` circumvents this problem by deriving the clustering proximity requirement via the geometric layout, thus its name. It will cluster together any pixels which touch at least in one point, no user set distance cut has to be applied. The time cut is the same as in `EUTelProcessorSparseClustering`. The important parameters of this processor are summarised in Table C.5.

C.3. Other Processors

EUTelHitMakerTwo

This processor arised from the necessity of treating new cluster objects differently than the old ones, given that they do not derive from a common base class anymore. It is used in the exact same way as the old hitmaker (`EUTelProcessorHitMaker`) and has identical parameters.

Variable Name	Description	Variable Type	Default Value
`ZSDataCollectionName`	Input raw data collection name	`string`	"zsdata"
`PulseCollectionName`	Name of output pulse collection storing clusters	`string`	"cluster"
`TCut`	Maximum time difference in detector specific time units, for hits belonging to the same cluster	`float`	max(`float`)
`HistogramFilling`	Switch for turning histograms on or off	`bool`	`true`
`ExcludedPlanes`	Planes to be excluded from the clustering	`vector<int>`	empty

Table C.5.: Parameters of `EUTelProcessorGeometricClustering`.

EUTelAPIXTbTrackTuple

The `EUTelAPIXTbTrackTuple` is the interface between EUTELESCOPE and TBmonII. It exports the reconstructed tracks and predicted points from any track fitting processor for later analysis in TBmonII. For this three collections have to be provided, the raw tracker data, the tracks and unaligned points for these tracks. `EUTelAPIXTbTrackTuple` will not undo any alignment and the appropriate processor has to be called prior to this one. Table C.6 gives an overview of all the parameters.

Variable Name	Description	Variable Type	Default Value
`InputTrackCollection-Name`	Name of track collection	`string`	"fittracks"
`InputTrackerHit-CollectionName`	Name of unaligned hit collection	`string`	"fitpoints"
`DutZsColName`	Name of raw tracker data collection	`string`	"zsdata_apix"
`OutputPath`	Name of output `ROOT` file	`string`	"NTuple.root"
`DUTIDs`	`sensorID`s of all DUTs which should be dumped	`vector<int>`	empty

Table C.6.: Parameters of `EUTelAPIXTbTrackTuple`.

Bibliography

[1] K. A. Olive, et al. (Particle Data Group), *2014 Review of Particle Physics*, Chin. Phys. C **38**, 090001 (2014)

[2] F. J. Hasert, et al., *Search for elastic muon-neutrino electron scattering*, Phys. Lett. B **46**, 121 (1973)

[3] F. J. Hasert, et al., *Observation of neutrino-like interactions without muon or electron in the gargamelle neutrino experiment*, Phys. Lett. B **46**, 138 (1973)

[4] UA1 Collaboration (G. Arnison et al.), *Experimental observation of isolated large transverse energy electrons with associated missing energy at* $\sqrt{s} = 540$ *GeV*, Phys. Lett. B **122**, 103 (1983)

[5] The UA2 Collaboration (M. Banner et al.), *Observation of single isolated electrons of high transverse momentum in events with missing transverse energy at the CERN* $\bar{p}p$ *collider*, Phys. Lett. B **122**, 476 (1983)

[6] UA1 Collaboration (G. Arnison et al.), *Experimental observation of lepton pairs of invariant mass around 95 GeV/*c^2 *at the CERN SPS collider*, Phys. Lett. B **126**, 398 (1983)

[7] The UA2 Collaboration (P. Bagnaia et al.), *Evidence for* $Z^0 \rightarrow e^+e^-$ *at the CERN* $\bar{p}p$ *collider*, Phys. Lett. B **129**, 130 (1983)

[8] The ATLAS Collaboration, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716**, 1 (2012)

[9] The CMS Collaboration (Serguei Chatrchyan et al.), *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett. B **716**, 30 (2012)

[10] P. W. Higgs, *Broken Symmetries and the Masses of Gauge Bosons*, Phys. Rev. Lett. **13**, 508 (1964)

[11] F. Englert, R. Brout, *Broken Symmetry and the Mass of Gauge Vector Mesons*, Phys. Rev. Lett. **13**, 321 (1964)

[12] J. J. Aubert, et al., *Experimental Observation of a Heavy Particle J*, Phys. Rev. Lett. **33**, 1404 (1974)

[13] J. E. Augustin, et al., *Discovery of a Narrow Resonance in e^+e^- Annihilation*, Phys. Rev. Lett. **33**, 1406 (1974)

[14] TASSO Collaboration (R. Brandelik et al.), *Evidence for planar events in e^+e^- annihilation at high energies*, Phys. Lett. B **86**, 243 (1979)

[15] J. H. Christenson, et al., *Evidence for the 2π Decay of the K_2^0 Meson*, Phys. Rev. Lett. **13**, 138 (1964)

[16] N. Cabibbo, *Unitary Symmetry and Leptonic Decays*, Phys. Rev. Lett. **10**, 531 (1963)

[17] M. Kobayashi, T. Maskawa, *CP-Violation in the Renormalizable Theory of Weak Interaction*, Prog. Theor. Phys. **49**, 652 (1973)

[18] S. W. Herb, et al., *Observation of a Dimuon Resonance at 9.5 GeV in 400-GeV Proton-Nucleus Collisions*, Phys. Rev. Lett. **39**, 252 (1977)

[19] S. Abachi, et al., *Search for High Mass Top Quark Production in $p\bar{p}$ Collisions at $\sqrt{s} = 1.8$ TeV*, Phys. Rev. Lett. **74**, 2422 (1995)

[20] F. Abe, et al., *Observation of Top Quark Production in $\bar{p}p$ Collisions with the Collider Detector at Fermilab*, Phys. Rev. Lett. **74**, 2626 (1995)

[21] Super-Kamiokande Collaboration (Y. Fukuda et al.), *Evidence for Oscillation of Atmospheric Neutrinos*, Phys. Rev. Lett. **81**, 1562 (1998)

[22] The ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3**, S08003 (2008)

[23] M. Capeans, et al., *ATLAS Insertable B-Layer Technical Design Report*, CERN-LHCC-2010-013; ATLAS-TDR-19 (2010)

[24] H. Bichsel, *A method to improve tracking and particle identification in TPCs and silicon detectors*, Nucl. Instrum. Meth. A **562**, 154 (2006)

[25] S. Meroli, et al., *Energy loss measurement for charged particles in very thin silicon layers*, JINST **6**, P06013 (2011)

[26] J. Große-Knetter, *Vertex measurement at a hadron collider - the ATLAS Pixel Detector*, BONN-IR-2008-04 (2008)

[27] W. Shockley, *Currents to Conductors Induced by a Moving Point Charge*, J. Appl. Phys. **9**, 635 (1938)

[28] S. Ramo, *Currents Induced by Electron Motion*, Proc. IRE **27**, 584 (1939)

[29] R. Turchetta, *Spatial resolution of silicon microstrip detectors*, Nucl. Instrum. Meth. A **335**, 44 (1993)

[30] G. Aad, et al., *ATLAS pixel detector electronics and sensors*, JINST **3** (2008)

[31] J. Albert, et al., *Prototype ATLAS IBL Modules using the FE-I4A Front-End Readout Chip*, JINST **7**, P11010 (2012)

[32] The ATLAS Collaboration, *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*, CERN-LHCC-2012-022; LHCC-I-023 (2012)

[33] `http://www.iphc.cnrs.fr/List-of-MIMOSA-chips.html`, visited on October 19th, 2014

[34] *MIMOSA26 User Manual (Preliminary version)*, V.1.5, IPHC (2011)

[35] D. Cussans, *Description of the JRA1 Trigger Logic Unit (TLU), v0.2*, EUDET-Memo-2008-50 (2008)

[36] E. Corrin, *EUDAQ Software User Manual*, EUDET-Memo-2010-01 (2010)

[37] EUTelescope website, `http://eutelescope.web.cern.ch`, visited on October 1st, 2014

[38] M. Backhaus, et al., *Development of a versatile and modular test system for ATLAS hybrid pixel detectors*, Nucl. Instrum. Meth. A **650**, 37 (2011)

[39] J. Schneider, H. Krüger, *S3 Multi IO System - S3 Multi IO USB Card*, V1.03 (2010)

[40] T. Obermann, *Development of a test beam telescope based on the ATLAS front end ASIC FE-I4*, BONN-IB-2012-14 (2012)

[41] J. Agricola, *Development Of A Faster Test System For ATLAS Pixel Front End Electronics*, II.Physik-UniGö-MSc-2014/05, MSc thesis, Georg-August-Universität Göttingen (2014)

[42] E. Browne, *Nuclear Data Sheets for A = 90*, Nucl. Data Sheets **82**, 379 (1997)

[43] R. Frühwirth, A. Strandlie, *Track fitting with ambiguities and noise: A study of elastic tracking and nonlinear filters*, Comput. Phys. Commun. **120**, 197 (1999)

Danksagung

Ich möchte Arnulf Quadt und Jörn Große-Knetter nicht nur für ihre gute und professionelle Betreuung danken, sondern auch dafür, dass sie die Gutachten meiner Arbeit übernommen haben.
Weiters möchte ich Jens Weingarten für die zusätzliche Betreuung, den vielen guten Rat und für das Probelesen dieser Arbeit danken.
Ebenso gilt mein Dank Johannes, welchen ich mit unzähligen Fragen mit mehr oder weniger Informatikbezug gequält habe, aber auch dem Rest der Hardware-Untergruppe möchte ich für die schöne Zeit und viele Hilfe danken: Danke an Julia, Lars und Matze.
Ich hatte die Ehre einige Dienstreisen unternehmen zu dürfen, und eine noch größere Freude war es mein ausgelegtes Geld rückerstattet zu bekommen, danke Lucie für die stets reibungslose Abwicklung dessen.
Für die gute Zusammenarbeit will ich Igor und Hanno danken. Die Gespräche mit euch hatten zwar desöftern nicht das eigentliche Thema als Inhalt, dennoch habe ich durch sie einiges gelernt.
Danke an meine Eltern, Großeltern und meine Schwester, die mir nicht nur das Studium ermöglicht haben, sondern mich auch stets unterstützt haben. Ebenso konnte ich stets mit Unterstützung, Hilfe und Rat meiner Freundin, Katharina, rechnen. Ich möchte dir dafür und für das Probelesen meiner Arbeit danken.
Ebenso danke ich all meinen Freunden die immer für mich da waren, insbesondere Ondřej, der mir bei Programmierfragen immer eine riesige Hilfe war. Desweiteren danke ich Andreas für das kurzfristige Korrigieren meiner Arbeit.

Erklärung nach §17(9) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.
Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 27. Mai 2016

(Tobias Bisanz)