HEPHY Institute of High Energy Physics

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Diplomarbeit

Beam test data analysis and resolution studies for the Belle II Silicon Vertex Detector

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom Ingenieurs unter der Leitung von

Doz. DI Dr. Christoph Schwanda
Institut für Hochenergiephysik
Betreuer: DI Dr. Thomas Bergauer
Institut für Hochenergiephysik

eingereicht an der Technischen Universität Wien
Fakultät für Physik

von

Benedikt Würkner, BSc.

Matrikelnummer: 0826633
Körblereck 26
8380 Jennersdorf

Wien im Sommer 2015

Contents

Abstract

The Institute of High Energy PHYsics (HEPHY), which is a part of the Austrian Academy of Sciences (ÖAW[1]), contributes to several high energy physics experiments around the world. One of these experiments is Belle II stationed at the KEKB particle accelerator in Tsukuba, Japan. Belle II is the upgrade to the Belle experiment which ran successfully from 1999 till 2009 and experimentally confirmed CP[2]-violation. The upgrade of the accelerator to a 40 times higher luminosity requires an equal increase in detector performance to cope with the higher particle rate.

One part of the upgraded detector is the SVD, the innermost vertex tracking detector, with sensor layout, readout electronics design and manufacturing, ladder assembly and sensor testing being contributed by HEPHY.

As part of my masters thesis I improved and extended the database keeping track of all the parts that will make up the detector as well as determining the resolution of the silicon strip detectors that will be used in the SVD. To this aim beam test at the SPS at CERN were performed, where the sensors were measured with a beam telescope, the AIDA telescope, to gain insight on the sensor resolution.

[1] Österreichische Akademie der Wissenschaften

[2] Charge-Parity, short for charge-parity-conjugation

Kurzfassung

Das Institut für HochEnergiePHYsik (HEPHY) der Österreichische Akademie der Wissenschaften beteiligt sich an etlichen Hochenergiephysik Experimenten weltweit. Eines dieser Experimente ist Belle II das am KEKB Teilchenbeschleuniger in Tsukuba, Japan stationiert ist. Belle II ist das Upgrade des Belle Experiments, das von 1999 bis 2009 erfolgreich lief und den experimentellen Nachweis zur CP[3]-Verletzung lieferte. Das Upgrade des Beschleunigers auf eine 40-fach höhere Intensität benötigt auch eine gleichzeitige, entsprechende Verbesserung der Detektorleistung um mit der erhöhten Anzahl an Teilchen umgehen zu können.

Ein Teil dieses verbesserten Detektors ist der SVD, der innerste Spurdetektor, für den unter anderem Sensor Layouts, Ausleseelektronik (sowohl Design als auch Produktion), Ladder Montage und Sensortests vom HEPHY beigesteuert werden.

Als Teil meiner Masterarbeit habe ich Verbesserungen und neue Funktionalitäten für die Datenbank, die zur Verwaltung aller Teile des Detektors verwendet wird, beigesteuert. Desweiteren habe ich die Auflösung der im Detektor verwendeten Silizium-Streifen-Detektoren untersucht. Um diese durchzuführen nahm ich an Strahltests am SPS, CERN teil. Hierfür wurden die Detektoren mit einem Strahlteleskop, dem AIDA Teleskop untersucht und ihre Auflösung bestimmt.

[3]Charge-Parity-conjugation, auf Deutsch Ladungs-Paritäts-Erhaltung

Chapter 1

Belle II and its new SVD

1.1 Physics motivation

Parity[1] was believed to be one of the fundamental conservation laws (along with energy and momentum conservation) until 1956 when a review of existing experimental data revealed that it had only been verified for strong and electromagnetic interactions. This led to experiments investigating parity conservation for the weak interaction with the first one (based on the beta decay of ^{60}Co) already showing that parity was not conserved for the weak interaction.

Following from quantum mechanics, symmetry can be restored if a second symmetry can be found leading to the combined symmetry not being broken. The property that was postulated was the charge of a particle, theoretically leading to the assumption that if one replaces all particles in a mirrored reaction with their antiparticles the reaction should have the same behavior.

This was proven wrong in 1964 in the K_0 meson system where it was observed, that the likelihood of a neutral kaon transforming into its antiparticle, by replacing each quark with its anti-quark, is not equal to the likelihood of the anti-particle transforming into the particle. This led to a series of experiments in search of this effect with other particles than kaons. Part of this new generation of experiments was the Belle experiment with its B-factory where CP[2]-violation for B mesons[3] was investigated. CP-violation within the standard model is predicted by the CKM matrix[4] by a complex phase that describes quark mixing but requires the presence of (at least) three generations of quarks.

According to current theories in the early days of the universe a process called baryogenesis is the reason for the asymmetry between matter and anti-matter and leads to the existence of the universe as we know it. Since matter and anti-matter annihilate on contact only an asymmetry during the creation of particles from energy

[1] Parity symmetry: Mirrored reactions of particles occur at the same rate as the original reactions.

[2] Charge-Parity conjugation

[3] Mesons containing the b-quark

[4] Cabibbo–Kobayashi–Maskawa matrix

could have led to more matter than anti-matter, thereby leading to an amount of matter that cannot be annihilated due to a lack of anti-matter counter particles.

To investigate CP-violation for B mesons, the probabilities of different decay paths are analyzed. The experimental challenge is that the branching ratio for decays containing CP-violation are rather low e.g. $B^0 \to k^+\pi^- \sim 2 \times 10^{-5}$. It is therefore necessary to have a high number of collisions resulting in reasonably good statistics regarding the events with low occurrence.

The experimental lifetime of B mesons is around $\tau_B \approx 1.5 \times 10^{-12} s$ which is equivalent to a maximal flight distance of $c\tau \approx 450\,\mu m$. As a consequence only the decay products of a B meson reach the detector, therefore requiring a very high precision measurement to be able to reconstruct the primary interaction. [1]

1.2 Belle

The Belle experiment, which was recording data from 1999 till 2009, was designed to precisely measure the decay vertices of B mesons to confirm the prediction of CP-violation. To achieve this, the Belle detector consisted of multiple parts, each fulfilling a specific role in the reconstruction of the particles. The inner components were:

- Silicon Vertex Detector (SVD): Aiming to precisely reconstruct the decay vertices of B mesons
- Central Drift Chamber(CDC): Main tracking detector measuring the tracks as well as the momenta of the particles.
- Silica-Aerogel Cherenkov Counter (ACC): Distinguishes mainly between pions and kaons.
- Time-of-Flight Counter (TOF): Identifies particles by their velocity and provides precise timing signals for triggering and gating.

The experiment was constructed and operated at the KEKB particle accelerator, which is operated by the **High Energy Accelerator Research Organization** known as KEK[5] in Tsukuba, Japan. KEKB is an asymmetric electron-positron collider with a center of mass energy equivalent to the $\Upsilon(4S)$ meson excitation level. The asymmetry of the particle beams leads to a moving center of mass of the created particles, thereby allowing for a measurement of the B meson decay times by determining the distance between the decay origin, as calculated from the decay vertices, and the collision point, which is known.

Following from the improvements planned and currently in progress for the KEKB, which will lead to an increase in luminosity by a factor of 40, the detector needs to be upgraded as well, to cope with the increase in decays and background radiation and other effects. Therefore the decision was reached to upgrade the Belle detector to Belle II with all new detectors.

[5]KEK is the abbreviation of its Japanese name

1.3 Belle II

The main design changes introduced in Belle II are:

- Addition of two layers of pixel detectors as the two innermost layers of the detector.
- Extension of the four silicon strip detector layers to a greater radius since the inner space is now occupied by the pixel detector.
- The readout chip of the silicon strip detector is changed from the VA1TA chip to the APV25 chip which has a much shorter shaping time.
- The drift chamber surrounding the silicon layers is replaced with a new one containing smaller drift cells and extending to a larger radius.
- A new data acquisition system that meets the requirements of the higher event rate was developed.

The full description can be found in [2] chapter 1.3.

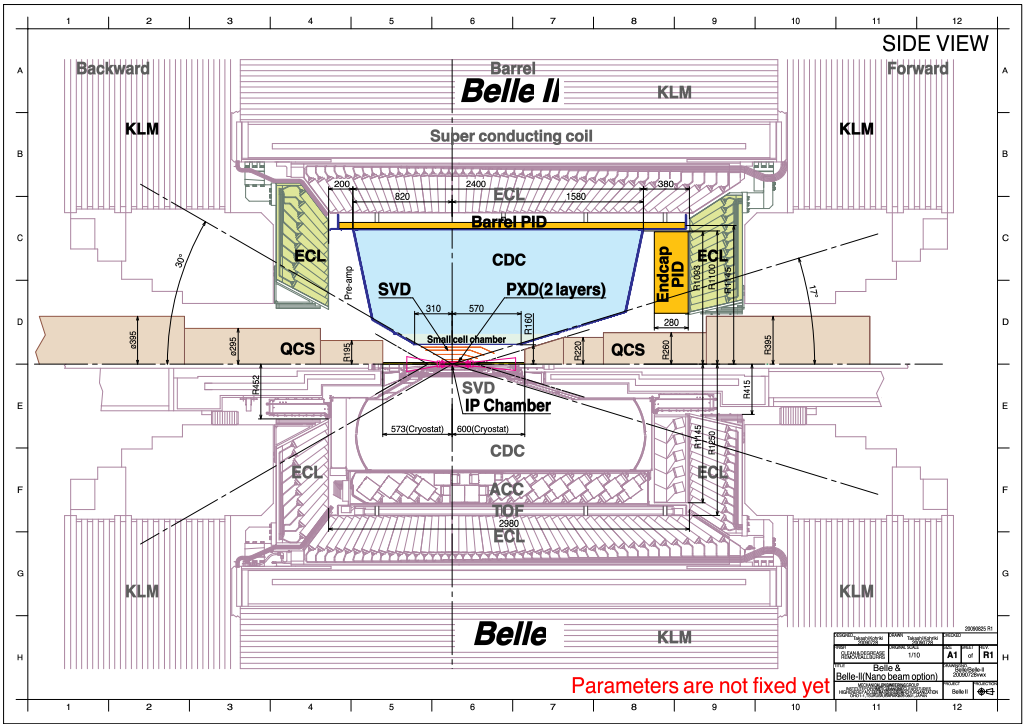


Figure 1.1: Comparison of the Belle II detector (top) with the Belle detector (bottom) taken from [2]

The detector is built by a collaboration around the world with institutes participating from the following countries (in alphabetic order): Australia, Austria, Germany, India, Italy and Japan.

The participation of HEPHY includes: construction and assembly of parts for the silicon strip detector, readout electronics design and main programming work, development of the data acquisition and online analysis software as well as integration of the relevant parts into the slow control structure.

Furthermore HEPHY also helps develop the mechanical mounting structure, performs sensor testing in the lab as well as during beam tests including analysis of the results and provides infrastructure like the construction database.

1.4 The SVD of Belle II

SVD is the abbreviation of silicon vertex detector. A vertex detector is designed to determine the point in space where a particle decayed by reconstructing the tracks particles took as they made their way through the detector back to their origin. This SVD consists of four layers of DSSDs[6] with strips on each side which are orthogonal to each other.

1.4.1 Requirements

The increase of luminosity with the upgrade of the KEKB leads to a higher event rate, thereby enabling a higher trigger rate for the detector. The maximum trigger rate of a system is influenced heavily by the time it takes to read out all the sensors, since a trigger can only be used if the system is ready to be read out. Any particle that would activate the trigger during the readout of the detector cannot be measured and is lost.

In the design of Belle II the old readout chip was replaced with a different one, that was previously developed for the CMS experiment, called APV25. The shaping time of the input channel is only 50 ns and with an implementation of hit-time-finding it can even be reduced to 3 ns with a time over threshold of only 20 ns[3]. The integration time of the SVD of Belle II is about 1/100st of the integration time of the old SVD used in Belle which should be sufficient for the expected event rate. This is also required to keep the occupancy at an acceptable level. Occupancy is the amount of strips that registered a signal within one integration period.

1.4.2 Layout

The whole design of the SVD for Belle II only consists of three different types of sensors [4], one type required for the innermost layer, with a smaller pitch for a higher resolution, and the other two types for the outer layers (fig. 1.2).

The design of the sensors themselves is determined mainly by the requirement for a minimal material budget. To achieve this the largest commercially available sensors are used, thereby reducing the amount of support structure required per active sensor area. The sensors are double-sided because this enables measuring a particle's position in two dimensions without increasing the thickness of the sensor. On the other hand this increases the cost per sensor as well as the complexity of the design of the surrounding structure like the readout electronics. In this case a method, called origami method, is used to have the readout electronics of each central ladder

[6] Double Sided Silicon Detectors

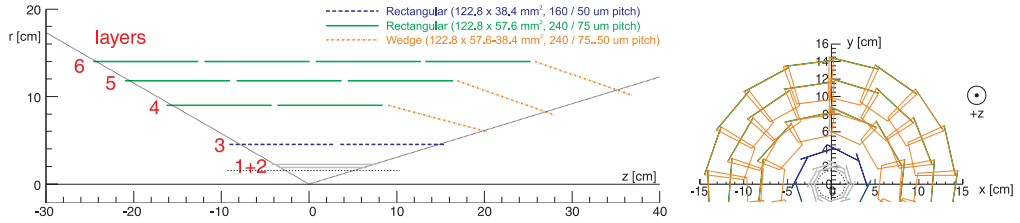


Figure 1.2: Layout of the new silicon detector for Belle II, including the two pixel layers (1+2) and the four strip detector layers (3-6) commonly dubbed SVD [4]

sensor on the side which is further away from the collision point to reduce the material budget in front of the readout area.

The standard wafer size for current production runs is 150mm in diameter[7]. This enables sensor sizes with outer dimensions of $12 \times 6\,\mathrm{cm}^2$ while using most of the available area. The wafer is standard high resistivity n-bulk material, as it is common in the semiconductor industry, and has a thickness of $300 - 320\,\mu\mathrm{m}$.

The sensing strips are implanted into the n-type bulk either with acceptors or donors, depending on the sensor side, thereby creating an n-side (donor in n-type bulk) and a p-side (acceptor in n-type bulk). The implanted strips are then coupled capacitively to an aluminium strip placed on the surface of the substrate for readout. Between each of the coupled strips there is an additional implanted strip which is not read out but helps to increase the resolution by also attracting charges and then coupling capacitively to the neighboring strips, thereby increasing the charge there. These strips are also not considered when discussing strip numbers, pitch and other strip relevant parameters since they don't need to be wire bonded or considered in the readout software.

The innermost SVD layer, known as Layer3 consists of ladders that contain only two sensors. They are identical but differ slightly from the rectangular sensors used in the layers four to six. The main difference is that the sensors are only 38.4 mm (and not 57.6 mm) wide but contain the same amount of strips as the wider sensors, leading to a smaller pitch of only 50 µm and thereby to a higher resolution.

1.4.3 Construction

The construction of parts for the Belle II detector is performed by different institutes all over the world. For example HEPHY builds the ladders for the fifth layer which includes two rectangular modules in the so-called "origami" configuration, one backward rectangular module and one forward wedge module. All forward and backward modules for all ladders are constructed in Pisa and then sent to the assembly sites where they are combined with the other modules and fixed onto the support structure.

Once they are finished and tested, all the parts are sent to KEK for storage until

[7] 150 mm = 5.91 inches leading to a wafer of this size commonly being dubbed 6 inch

the detector is put together.

1.5 Silicon strip sensors

1.5.1 Working principle

When an ionizing particle passes through silicon substrate it deposits energy within the material, thereby creating electron hole pairs. An intrinsic silicon substrate contains several orders of magnitude more free charge carriers than are generated by this particle. Therefore the number of free charge carriers needs to be reduced to enable usage as a detector. One way of doing this would be cooling to very low temperatures which is not feasible for large detector applications. The second possibility is to use p- and n-type silicon in a *reverse-biased pn-junction* configuration to deplete the silicon volume of free charge carriers.

Silicon detectors consist of a bulk material, in this case of type n and implanted strips which are doped the other way, in this case p+. A high bias voltage is applied between the p+ strips and the back of the sensor creating a charge free zone within the bulk and a strong electric field moving the charges (electrons or holes) to the respective attracting side creating an electrical signal in the strips. This signal is transferred over the capacitive coupling to a parallel strip on the surface of the sensor which is present for every readout strip and the coupled signal is then received by the readout chip's input.

The capacitive coupling prevents the dark current within the silicon substrate from entering the readout chip's input since a direct current is not capable of passing over the capacitor. Still fluctuations within the dark current can couple over the capacitor contributing to the noise of a strip.

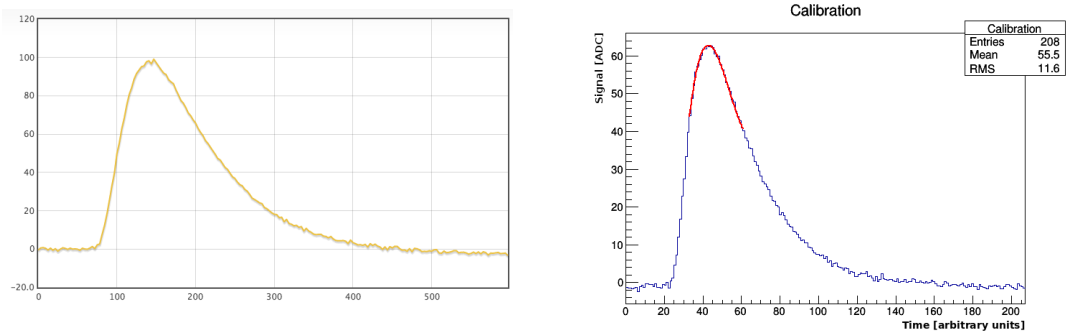
1.5.2 Readout Chip

A readout chip measures the charge deposited on each connected strip and transmits the signal on request.

The chip used in Belle II is called APV25 and is the same as used in the CMS experiment. It has 128 analog input channels where the charge that is received at the input is integrated over a shaping time of 50 ns and then shaped by a CR-RC shaper into a pulse, where the amplitude is proportional to the collected charge. The processed signals are then stored into a per-strip 192 cell big ring buffer. Via an I^2C command the APV25 can be instructed to transmit 1,3,6,9...32 of the stored values. This method of readout allows for a considerable trigger delay since the APV25 can return previous values of each strip on request. [5]

To have a reference of the charge deposited by one particle the APV25 is capable of calibrating itself by injecting a defined amount of charge into the electrode connected to the pre-amplifier. The amount of charge is configurable by an I^2C command. The

calibration command is generally used to determine defects in the readout channel and the connected sensor.



(a) taken from the database

(b) taken from [6]

Figure 1.3: Examples of calibration curves of one strip

The data from one APV25 is transferred using a multiplexer before transmitting the signals in analog peaks, preceded by a header and closing with a signaling bit (see fig. 1.4). The signals are not transmitted in consecutive order, but follow a different scheme which prevents high signals from influencing the signal of strips directly adjacent on the sensor during transmission which would create false clusters otherwise.

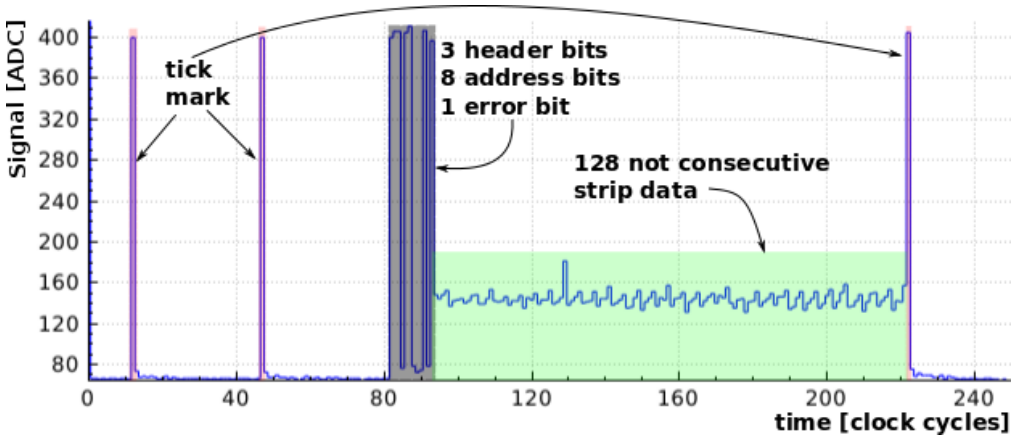


Figure 1.4: Example of one data packet transmitted from one APV25 upon request. Taken from [6]

The DAQ[8] system is capable of performing a multitude of different commands on the connected sensors. One of these commands is the calibration run whose goal it is to find defective strips and also estimate the gain of each strip. To that aim a predefined voltage, usually corresponding to 22400 electrons, is injected in the preamplifier of the APV25. The result is a shaper pulse with the maximum proportional to the injected charge and a resolution of one eighths of a clock cycle.

Another important piece of information required about the sensor before one can start measuring particles with it is how the sensor behaves without particle interac-

[8]Data AQisition

tion. This so-called "Software Run" is performed using software generated random triggers, thereby on average not coinciding with a particle hit.

At the beginning of every run a number of events are taken using the software trigger to calculate the pedestal of each strip. The pedestal is calculated from the mean signal height of each strip and subtracted from each strip signal to compensate for the differences between the strips.

After the pedestal correction the common mode correction is calculated. Common mode is the name for the fluctuations of the 128 strips of one APV25 around their pedestal values. To compensate for this, the strips are usually split into four groups of 32 strips whose pedestal values are then sorted, the top and bottom 25% are cut off and the remaining values are used to calculate a mean for common mode correction. The calculated value is then also subtracted from the signal received.

To calculate the signal to noise ratio which, amongst other things enables the removal of faulty strips from the analysis automatically, the noise of each strip is calculated. It is defined as the standard deviation of the gauss distribution after already removing the pedestal and common mode correction from the signal. Strips with a noise of 2.5× the mean noise of the corresponding APV25 are marked as bad and removed from the analysis.

For a more detailed explanation of these corrections performed on the signal refer to [6].

1.5.3 Clustering and charge sharing

The free charges created by a particle are in most cases not attracted by only one strip but create a signal on multiple strips, thereby forming a cluster. Since the charge of a cluster is spread over multiple strips only clusters below a certain size can be detected because the separate charges fall below the limit of hit recognition. Once a strip is recognized to be over the hit limit, a lower limit is applied to neighboring strips to improve cluster recognition.

Intermediate strips

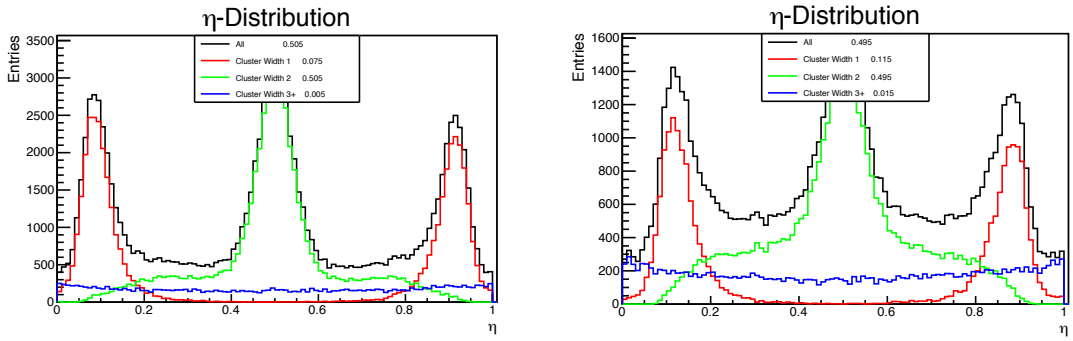
The number of channels that can be read out simultaneously is limited by the amount of chips that can be physically placed near the sensor as well as the amount of cables required for the data to be transmitted to the readout system. To help minimize the number of electronic channels required intermediate strips are implanted into the sensor while omitting the readout strip. The implanted strip then couples capacitively to its neighboring readout strips giving an improved resolution without the requirement of additional readout channels.

Eta distribution

To determine the position of a particle that passed through the sensor, thereby creating a cluster, a center of gravity distribution is used. The charges detected

at each strip of the cluster are weighted with respect to their distance and then normalized. The result is called η-distribution and gives insight on the behavior of a sensor and its resolution.

The shape of the η-distribution depends on the pitch, the existence (and amount) of intermediate strips and the precision and energy of the particle beam.



(a) BW module, n-side, 240 µm pitch (b) BW module, p-side,75 µm pitch

Figure 1.5: Comparison of two η plots at different pitch sizes. The colors denote the different cluster sizes. One can see, that the central peak is created exclusively by cluster size two. When looking at the sum of the parts, the influence of the pitch becomes quite apparent.

The η-distribution is used to increase the resolution of a sensor by means of a correction factor. For a detailed description of the calculation of the factor please refer to [6], chapter "position of the particle".

1.5.4 Hybrid

The readout chips are mounted on a PCB known as hybrid(fig. 1.6). Each strip is connected via a wire bond to a pitch adapter whose main purpose is to change the pitch of the strips on the sensor to the pitch of the inputs of the readout chip while also bridging varying distances between the connection pads on the sensor and the pads on the hybrid. The connection from the pitch adapter to the chip input is again a wire bond. Additionally the pitch adapters are also used to connect strips from the one side of the sensor to the readout chips on the other side since they are quite flexible. Ideally the distance between the strip and the chip input should be as short as possible to reduce the series resistance.

1.5.5 Signal to noise ratio

A fully depleted sensor bulk volume still contains sources of statistical free charge fluctuations. Since they are statistically distributed they hinder signal recognition by adding noise to the signal. As mentioned before the signal of a silicon sensor depends solely on the thickness of the sensor and it is therefore in most cases more reasonable to reduce the sensor noise than to increase the thickness which has additional negative implications such as a higher material budget.

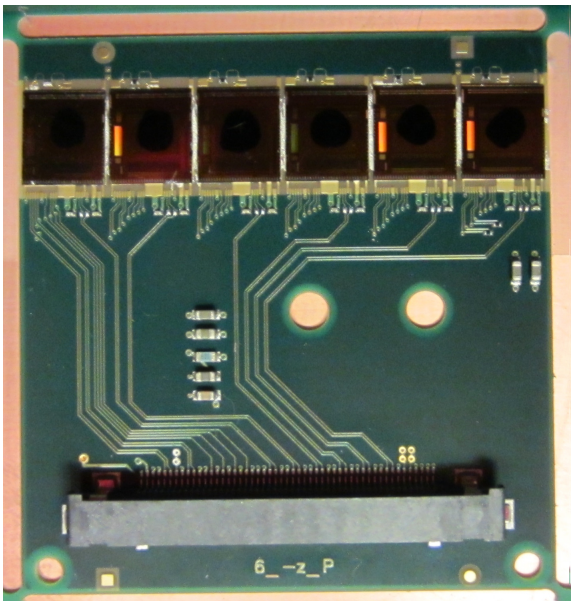


Figure 1.6: Image of a hybrid board with all six APV25 chips already in place.

The main contributions to a sensors noise are load capacity C_d, leakage current I_L, parallel and series resistances R_P and R_S. Within this the load capacity is mostly comprised of the capacity between the implanted and readout strips and the strip to backplane capacitance.

Noise is commonly expressed as Equivalent Noise Charge (ENC) giving the number of electrons contributing to the noise.

$$ENC = \sqrt{ENC_C^2 + ENC_{I_L}^2 + ENC_{R_P}^2 + ENC_{R_S}^2}$$

The most significant contribution to the combined noise tends to be the capacitive contribution wich can be represented by

$$ENC_C = a + b \cdot \mathbf{C_d}$$

with a and b preamplifier-specific parameters.

For example the readout chip used for Belle I was the VA1 with parameters $a = 165$ and $b = 6.1$ which leads to only a weak dependance on the capacitance. Since b is inversely proportional to the shaping time a low shaping time, such as the APV25s 50 ns leads to a high dependance on the charge based noise wich can be seen in the parameters for the APV25: $a = 250$ and $b = 36$. Therefore the noise depends strongly on the capacitance which in turn depends on the strip length. As a consequence the sensor specifications need to be adapted to the used readout chip to reduce the capacitive noise, in this case limiting the strip length and thereby the strip capacity if such a short shaping time is required.

As an example the difference in noise using the APV25 for a strip with a length of 38.42 mm is 420 e^-. For comparison, a strip length of 57.59 mm causes 504 e^- in capacitive noise, a 20% increase in noise for a 34% increase in strip length. The strip length values taken for the example are used in the Belle II detector.

Chapter 2

Construction database

As described in the previous chapter, the detector consists of many parts manufactured by many different institutes at many different locations. To keep track of all these parts and also the information associated with each, a database was started for the project[7]. The basic system and main structure was developed by Bernhard Leitl as a part of his master's thesis. Over the course of my work I added some additional functionality, improved some of his and implemented a lot of bugfixes.

2.1 Requirements

The main requirement for the database was the possibility to track items that are used throughout the project and the actions performed on them. Furthermore it needs to be possible to combine items to composites (virtual assembly) as well as create items (e.g. wafers) that can be disassembled and then attached separately to other items.

Another aspect is the logistics part of the database keeping track of the items and their locations, transferring them between locations, thereby trying to (as closely as possible) mimic the actions done in the real world. The whole database is centered around this logistics aspect, always presenting the user with the item inventory of their location right after login.

Additionally it was proposed to also store measurement data in the database, thereby making it the central point for all information relevant for assembly. This was my main task for the continued development of the database.

The basis of the database already existed and uses a php framework (cakePHP[1] with MySQL) and the above mentioned requirements were mostly already implemented, with the exception of the measurement storage system.

Over the course of this master's thesis, especially before the beam test, I took over the work of implementing the measurement storage(2.2.1) as well as tags for items(2.2.2). Additionally I completely reworked two existing features, namely the stock items(2.2.4) and the transfers(2.2.5).

[1] Cakephp Framework, website `http://cakephp.org`

2.2 Implemented features

Next to the larger feature implementations, I also implemented many bugfixes and helped an additional developer, Federico Pilo, from another institute implement another feature, the checklists.

During the final phase of my feature development an additional developer reworked the search selector, the main interface of the item overview which also required my participation on the backend.

2.2.1 Measurements

During the development and construction of a big physics experiment many measurements need to be performed on the components to make sure they are working correctly, weren't damaged during transport and so on.

Depending on the item type, this leads to many different measurements performed using different probe stations and acquisition software. This therefore requires a flexible as well as intelligent system to recognize different file formats and associate them with the correct location and measurement type.

Importing of measurement data

Each measurement performed generates an ASCII[2] file, usually using the file suffix *.txt* which is easy to open with php using simple file reading commands. Following from this, the whole import procedure was implemented completely in php. If data files had been in a raw data format, external (i.e. non-PHP-)code would have been required to interpret them.

Although all files are simple ASCII files, the data format within the files varies greatly between the different probe stations, requiring a rather complicated code to differentiate between the different formats and how to interpret them. This is due to the fact that the file formats were already in use for some time when the programming for the measurement import in the database was started. As a consequence, the current implementation of the import system is sometimes hardcoded and requires some advanced knowledge and access to the source code to be adapted. There have been thoughts about an implementation to enable the definition of file layouts using the database interface but this was put on hold due to manpower issues.

The current implementation uses a combination of string comparison and regular expressions to identify a file using its first few rows. In some cases these differ only slightly which makes long comparison clauses necessary. This is a rather rudimentary approach and can fail easily if somehow an unexpected character is inserted somewhere in the header of the file leading to the file not being recognized.

For newly created measurement files, a standard was defined using the popular CSV[3]

[2] American Standard Code for Information Interchange
[3] CommaSeparated Values

format with section markings and a standardized two-row header of key-value pairs. This format is mainly used for data recorded with APVDAQ which is still being updated and therefore it was easy to work together with the developer. Currently there are plans that a future functionality of measurement exports will also use this format to reduce the code required for a new file format.

The definition for this default file format looks like this:

```
[info]
ID,Type,Operator,StartDateTime,StopDateTime,MeasurementParameter1
3048-2_Wedge,measurement_name,UserName,'2014-01-27 11:47:05','2014-01-27 14:08:22',1234

[tags]
n-side,tested in Vienna

[parameters]
ItemParameter1,ItemParameter2,ItemParameter3
85, 35, 0

[measurement_name]
col1, col2, col3
146, 0, 0,
```

In the case of the APVDAQ there are three different measurement types (software, hardware and calibration) that each require some special treatment. Especially the calibration run creates large text files (about 5 MB uncompressed) that contain a lot of data. Importing these calibration files requires a lot of time due to the amount of database requests required to store the data. The current database scheme is relational leading to each value being inserted separately. Since the calibration measurement of one strip gives a signal for 208 times three parameters (chip, strip and time) the required number of insert operations for one strip is $208 \times 4 = 832$ (four because an additional insert is required to define each data row). One calibration measurement is usually performed on a sensor with 512 or 768 strips leading to around 425 000 and 639 000 insert operations respectively. This calculation shows that for a high amount of measurement values or parameters the amount of insert operations becomes rather high. On the current server hardware the import of one calibration measurement takes between 15 and 20 minutes which already uses all optimizations (like parallelization) I could think of.

Data storage and problems

At first this didn't seem to be too much of a problem, but then the realization came that requesting that many elements from the database also takes a long time, not just the insertion. Therefore I implemented a caching for the measurements so the request only has to be performed once. Later on I also added, that the cache is already generated by the import, thereby not requiring the user to wait on the first view (although the import now of course takes longer).

The current system also uses an external daemon, written in Python, that handles the import, thereby not blocking the user and the web server and enabling parallel processing as a benefit. This on the other hand makes the system even more complex than it already is.

All these negative side effects were overlooked or not considered important enough during the design of the program. Since the problems became apparent, some additional improvements were implemented, like the caching, but due to time constraints they are still not resolved.

One possible solution would be to not store the data in the database at all and always read it out from files stored on the file-system. This would limit the code to be only php and, given a database flag that is set during the import that defines the file type, would only require the execution of the file interpreter and not the whole recognition chain when a measurement is requested, thereby speeding up the process. This method seems feasible because most of the import time is actually required by the insertion into the database and not by the readout of the measurement file. A caveat of this system is that the data is not stored in the same format for all measurements and is also not automatically backed up with the database but requires special treatment. This could be dealt with by writing a container class that always returns the same data format and handles the differences between file formats internally.

Another possibility would be to convert each uploaded file to a generic format, preferably the already used CSV format and use these files as storage. This again creates the problem of backing up the data but it removes the differences between files and the measurement display would be more generic. Additionally downloading of the measurement data would always return a file with the same overall format without additional conversion.

The differences between these approaches should be weighted before implementation and also some time should be invested into future-proofing the system and backward compatibility. This should prevent a redesign requirement of the system again after only a couple of months.

Displaying of stored data

To plot the stored data directly in the browser, an external library was used. After some investigation the choice fell on the jqPlot library[4], mainly because it uses the jQuery framework which is already used throughout the project. Furthermore it allows for plugins to be added to enable additional functionalities and different plot types like bar, line, stacked and others.

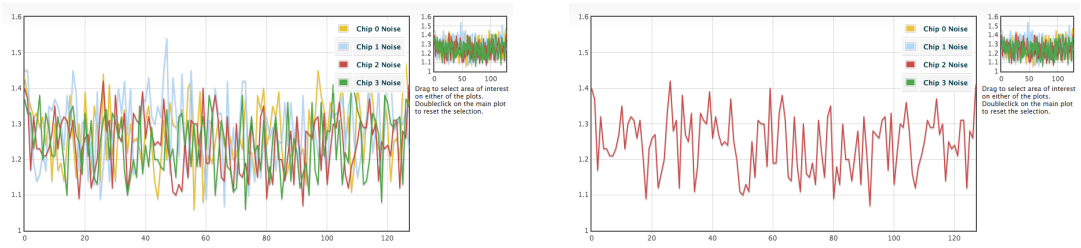
Using a javascript plotting library also has the benefit of the plotting being done in the browser which enables interactivity in the plotting window like zooming and changing the axis to logarithmic or absolute values.

Since many different measurement types are available, different presets for measurements are required. For example an APVDAQ measurement is usually performed on a sensor using multiple chips. It is therefore reasonable to plot all chips at once in different colors and to separate by the channel number of the chip, thereby enabling a comparison of different chips over the same sensor(fig. 2.1a).

Interactivity is given using right and left mouse button interactions on multiple objects. For example in the data table one can select the parameter for the y-axis (which is usually of interest) using the left mouse button, likewise using the right mouse button one can select the parameter for the x-axis.

[4]jpPlot homepage: `http://www.jqplot.com`

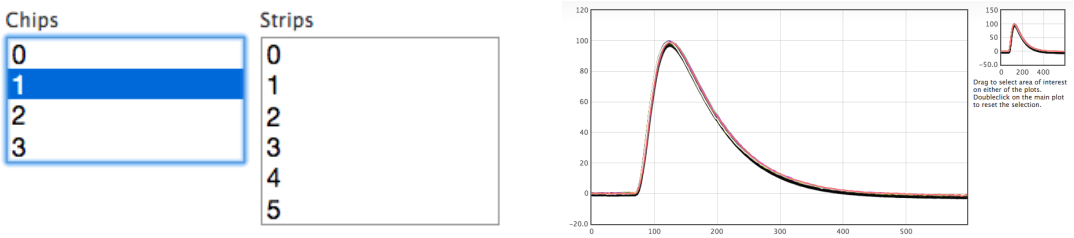
In the plot window one can select the option to show only one chip (left click on the name in the legend) or toggle the visibility of one specific plot (right click on the name in the legend) (fig. 2.1b).



(a) All four chips (b) One chip

Figure 2.1: Examples of an APVDAQ measurement with different amounts of chips plotted. The x-axis is the number of the channel on each chip while the y-axis is selected by the user to be in this case the Noise value.

Another special case is the APVDAQ calibration measurement which generates a graph for each channel. Therefore a selector was implemented for these measurements to allow selection of chip and strip before plotting them. It is, of course, also possible to plot all strips of a chip at the same time to see in one glance if there is a faulty calibration on this chip(fig. 2.2).



(a) Chip and strip selector (b) Plot of all 128 strips

Figure 2.2: Calibration scan plot method

Using stored data with parameters to generate additional information

Strip scans are performed on the sensors to gain information about each strip. This information is given in electric values (e.g. resistance, current, capacity, ...) which vary slightly between strips. To make this information easier to understand, especially since one is usually only interested in strips behaving differently, a functionality was implemented that allows checking if strip values are in a defined range. The result of this process is an additional measurement that contains a list of strips outside the range and the range definition. This makes it easier to get information about different errors (e.g. pinholes) at a glance.

The grading can be done for one or multiple items at once, the only requirement is that they are of the same subtype version. The current implementation allows this action only for administrators since it is a process that requires a lot of knowledge of the sensors.

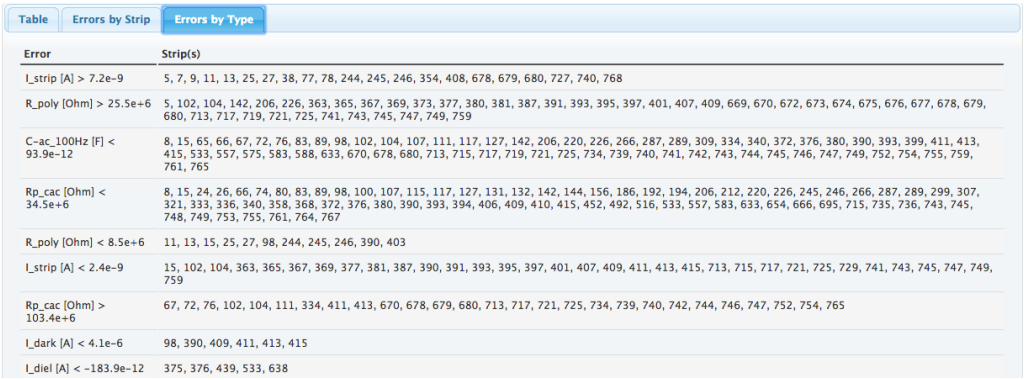


Table | Errors by Strip | Errors by Type

Error	Strip(s)
I_strip [A] > 7.2e-9	5, 7, 9, 11, 13, 25, 27, 38, 77, 78, 244, 245, 246, 354, 408, 678, 679, 680, 727, 740, 768
R_poly [Ohm] > 25.5e+6	5, 102, 104, 142, 206, 226, 363, 365, 367, 369, 373, 377, 380, 381, 387, 391, 393, 395, 397, 401, 407, 409, 669, 670, 672, 673, 674, 675, 676, 677, 678, 679, 680, 713, 717, 719, 721, 725, 741, 743, 745, 747, 749, 759
C-ac_100Hz [F] < 93.9e-12	8, 15, 65, 66, 67, 72, 76, 83, 89, 98, 102, 104, 107, 111, 117, 127, 142, 206, 220, 226, 266, 287, 289, 309, 334, 340, 372, 376, 380, 390, 393, 399, 411, 413, 415, 533, 557, 575, 583, 588, 633, 670, 678, 680, 713, 715, 717, 719, 721, 725, 734, 739, 740, 741, 742, 743, 744, 745, 746, 747, 749, 752, 754, 755, 759, 761, 765
Rp_cac [Ohm] < 34.5e+6	8, 15, 24, 26, 66, 74, 80, 83, 89, 98, 100, 107, 115, 117, 127, 131, 132, 142, 144, 156, 186, 192, 194, 206, 212, 220, 226, 245, 246, 266, 287, 289, 299, 307, 321, 333, 336, 340, 358, 368, 372, 376, 380, 390, 393, 394, 406, 409, 410, 415, 452, 492, 516, 533, 557, 583, 633, 654, 666, 695, 715, 735, 736, 743, 745, 748, 749, 753, 755, 761, 764, 767
R_poly [Ohm] < 8.5e+6	11, 13, 15, 25, 27, 98, 244, 245, 246, 390, 403
I_strip [A] < 2.4e-9	15, 102, 104, 363, 365, 367, 369, 377, 381, 387, 390, 391, 393, 395, 397, 401, 407, 409, 411, 413, 415, 713, 715, 717, 721, 725, 729, 741, 743, 745, 747, 749, 759
Rp_cac [Ohm] > 103.4e+6	67, 72, 76, 102, 104, 111, 334, 411, 413, 670, 678, 679, 680, 713, 717, 721, 725, 734, 739, 740, 742, 744, 746, 747, 752, 754, 765
I_dark [A] < 4.1e-6	98, 390, 409, 411, 413, 415
I_diel [A] < -183.9e-12	375, 376, 439, 533, 638

Figure 2.3: Results of one measurement grading. Usually fewer strips are outside of the parameters.

The result of such a grading can be seen in fig. 2.3. It is also possible to not use all measured parameters for a grading. If one wants to re-grade a sensor it is possible to remove the grading measurement by simply deleting it, thereby reactivating the grading checkbox.

2.2.2 Tags

Tags are widely used on the internet as defining properties of things while also making them selectable by their similar properties. This same concept was thought of and then applied to items in the database, thereby allowing the assignment of different properties to items while making them groupable and searchable as well. Tags are now used for a wide variety of properties, including, but not limited to, wafer numbers(e.g. APV wafer XF4B4AT), error descriptions(e.g. broken lines on PA, higher dark current), optical descriptions(e.g. severe scratches) and electrical definitions (e.g. electrically functional). An item can theoretically have an unlimited amount of tags assigned to it but no tag can be assigned more than once.

Shortly after implementing the tags into the database it became apparent that some sort of grouping would have to be applied since not all tags make sense for all items. Mechanical structures, for example, are not really capable of being 'electrically functional'. Therefore the grouping of tags for each combination of an item type and a project was implemented. Admins can select which tags are available for each combination of an item type and a project by means of a drag'n'drop interface dubbed 'Tag Cloud'. In this interface tags can be added and assigned to configurable combinations of projects and item types (fig. 2.4).

In the main search selector, items can also be filtered by tags. Here all tags are always available since it was not deemed worth the effort to remove tags from the list if a project and item type are selected. Tags that are not possible to set for an item are ignored anyways.

Tags can be assigned to items in multiple locations and in different ways. One possibility is to assign them during item creation. Another is of course in the item view where a separate page is opened showing only the available tags for this item(based on its item type and project). There is also the possibility to change

Figure 2.4: Screenshot of the tag cloud interface

tags for multiple items by selecting them in the search selector and using the 'change tags for these items' button to be presented with a drag'n'drop interface that allows fast assigning of tags to multiple items by dragging the tag to the grouping header or just assigning multiple tags to different items in one interface without requiring one to go back and forth all the time. If items of different item types are selected they are grouped accordingly so only possible tags can be assigned.

2.2.3 Item creation

Creating an item in the database is an important step as many properties defined during creation cannot be changed later on in the user interface, especially the item subtype version. Therefore the idea is to reduce the possibility for error by only presenting the user with possibilities that make sense for different items.

When I started working on the database there were three completely separated interfaces that could be used to create different types of items.

1. **Register**: Used to register wafers and all the components on the wafer in the database
2. **Assemble**: Used to create just a single item or put multiple components together to form a new item
3. **Register stock item**: Used to register an amount of stock items following the old system (see 2.2.4)

This nonintuitive separation, especially made worse by the lack of description within the database and the generic names for the menu items, led to much confusion for people. In conjunction with the reworking of the stock items (see 2.2.4) it was decided to streamline this interface reducing the possible confusion.

I thought about the dependences during item creation and decided to split the item creation into four in-order steps that all depend on the previous step(s) and prevent changes to the previous fields when continuing to the next. I used AJAX[5] to prevent having to reload the page constantly, which amongst other things increases the responsiveness and makes the webpage feel more like an application running locally.

The first step allows for the selection of a location, which is independent of the other parameters, and then, in the following order: the selection of project, item type, item subtype and item subtype version. Once an item subtype version and a location have been selected it is possible to continue to the second step, thereby finalizing the first selection.

The second step allows the user to enter either one or multiple unique codes to create distinct items or a number that defines the amount of stock items to create. Additionally it is required to select an item quality from the dropdown. It is also possible to add a comment and select tags, with the displayed tags depending on the selected item type-project combination.

For an administrator there exist some additional options. One allows forcing the register option, thereby allowing to create all items that should normally be attached to an item type instead of selecting them from the inventory. The second one allows changing the naming behavior by adding the short name in the middle of the item names during registration.

For a user the system decides based on the item type what the required course of action is. For wafers the default action is to create a wafer with several objects on it that are automatically named based on their type and can be taken off of the wafer while still maintaining the connection to the wafer number. For almost every other item type the action is to create the item by assembling it from two or more items or by simply adding it by name.

The third step now totally depends on the selections before and mainly differs between the 'registering' and 'assembling' ways of item creation. In both cases a tabbed selector with all item codes entered in the second step is shown.

In case of registering, each item is displayed with a checkbox allowing the user to decide if it should be created, including a collapsable section for the respective subitems. It is also possible to change the name of items to something other than the standard convention, although this is not recommended. Furthermore it is possible to select if the items should still be attached to the wafer or detached, thereby reducing the user effort to do this later on manually.

In case of assembling, each item receives a button that allows one to select a matching item from the inventory and attach it already during creation. This is optional and should match the real world actions. Attaching is, of course, still possible later on and so is registering but the latter requires more effort since the usage of the 'post creation' method is required.

After finalizing the selections for each item code (if applicable) the 'create item' button creates all items and attaches the items (if applicable).

[5] Asynchronus Javascript And XML

The chain differs only slightly for the creation of stock items, since there are no selections possible for them; There is only a message displayed telling the user what will happen. For example clicking on 'Create item' will create 20 Stock Items with the current configuration.

This change replaced all three menu items with a single 'Create' button and has led to way less confusion than previously, especially with new users of the database.

2.2.4 Stock items

Not all items that exists within the detector are distinguishable nor do they need to be. Items that fulfill this description are called 'a stock of items', usually just dubbed 'stock item' and are described by their item subtype version, their quality, the assigned tags and the amount present at the relevant locations. One example for a stock item would be the APV25 chip that is used for sensor readout. The only difference that can be between APVs of the same version is the wafer number. Aside from that they are interchangeable, and if they break, the amount of stock just needs to be reduced by the amount destroyed.

In the previous system stock items were handled completely differently than 'normal' items, requiring additional tables for lots of information that on second thought was redundant but could not be merged at first. This also led to complications while implementing the tags since I basically had to implement the tags twice, once for items and once for stock items. This also would have led to problems in the search selector, since two tables of different definitions would need to be searched at the same time, which would have become even more complicated if the tags were added, since they already required some tinkering with the search algorithm to work.

All these considerations led to the decision that the whole stock item system would need to be redesigned, in principle simplified, by merging stock items into items and just storing the additional information required for stock items in a separate table. This solved a lot of the problems with the surrounding system and streamlined the process. The current system generates a generic name for a stock item to be used as code, making sure that it is unique by using the automatically incremented item id, and just replacing the automatically generated name during display with the generic term 'stock item' and a count of the existing items at the relevant location.

The additional information that needs to be stored for a stock item is saved in a table linking the stock information to the item ID. The required information only contains the columns 'location' and 'amount', everything else is stored (as for every other item) in the items table.

During this work it became apparent that previously, it was not possible to send stock items via a transfer and the necessary changes for that were also planned to be implemented in conjunction with the redesign of the stock items. Details can be found in section 2.2.5.

With the new system treating stock items like items in most regards, the item view had to be adapted to recognize the difference and display the relevant information on request. This includes showing the locations and the amount at each location in

case it is a stock item in the place where usually only the current location of the item is displayed. Furthermore the 'Measurements', 'Components' and 'Check list' tabs are hidden for stock items as they are never necessary.

The result is that for the user the difference in the view is only marginal and the code actually was simplified by reducing the amount of complexity involved with two separate tables that pretty much contained the same information.

2.2.5 Transfers

As mentioned in the previous section transferring of stock items was not possible before the rework. Since this functionality was requested, the rework of the stock items included a rework of transfers. This also led to talks and ideas on how to further improve the transfers and finalized on the following changes.

- Transfer selectable amount of stock items
- Start transfer from inventory
- Allow multiple users to participate in one transfer
- Added user standard location
- Group transfers depending on users standard location
- Group transfers depending on status
- Only display transfers relevant for a user

The first big change was to remove the requirement for a separate inventory page where items for a transfer can be selected. It was decided that it is more reasonable to start a transfer from the inventory since the possibility to select items there was already implemented and working well. Therefore dynamic buttons were added to the inventory that depend on the selection of items and the location selected. If only one location is selected in the search selector, it is possible to start a transfer from said location by selecting items and pressing the 'add to new transfer from "current location"' button. If a user is adding a stock item to a transfer, a dropdown is displayed in the transfer configuration window allowing the selection of the amount of stock items to be added.

Another change is that transfers now have a status defining if they are 'pending', 'in transit' or 'completed' and are stored on the server and not in the users browser, thereby enabling multiple users to add items to the same transfer that is pending from the same location.

Along with these transfer states now defined for a transfer a user now has to 'send' a transfer, thereby preventing the further addition items and changing the state of the transfer to 'in transit'. This intermediate state requires the receiver to actively 'receive' the transfer to change the location of the items in the transfer to the target location. The receivers standard location needs to be set to the transfers destination for the functionality to work. This intermediate status was a much requested feature since before this change, upon submitting the transfer items would be moved to the target location and show up in the inventory of that location although the items

were not already physically there. To enable this functionality, a 'virtual location' called 'In Transfer' was added to the database that hosts all items that are currently in transit from one location to another before they are received. A side-effect of this is that, if one views a stock item where some amount is in a transfer, the table showing the locations of the stock items actually lists 'In Transfer' as a possibility.

Additional to this implementation, the concept of a 'standard location' for each user was invented, giving way to a location that is selected by default in the search selector, displayed in the top right corner of the interface and determining the possible actions for transfers for the user.

The overview of transfers was grouped into four sections:

- **Pending**: Transfers at the currently selected standard location that are yet to be sent. Can be sent
- **In Transit**: Transfers on their way to the selected standard location. Can be received
- **Other transfers currently in Transit**: Transfers to or from other locations assigned to the current user that are not his standard location. No interaction (except viewing) possible
- **Completed**: Transfers to or from locations assigned to the user that are completed. Only viewing possible.

Additionally, two indicators were added in the menu, displaying the current user relevant incoming and outgoing transfers in correspondence with the standard location.

With the addition of the standard location came some confusion for users that were used to working with multiple locations, especially with them all selected by default in the search selector, and therefore some items now not being shown at first due to the standard location being the only selected location by default. Following up on this some additional warning messages were implemented in the inventory telling the user that there are items matching the search criteria in other locations assigned to him. A link is provided allowing the user to quickly repeat the query, this time including all available locations.

2.3 Status and outlook

The current status of the database can be described as stable with a lot of known (mostly design based) issues. The bug tracking system currently lists mostly issues regarding small changes, a few are bigger improvements with the included time requirements. Some problems are not even mentioned there but known to the people responsible.

One of the bigger problems was already described in the Measurement section (2.2.1), a redesign of the way measurements are stored and handled on the file system to improve performance and reduce complexity. This is, in my opinion, a rather big task that would require about four to six weeks to be done properly.

Another issue, which is not really apparent to the user but might become a problem for the admins, is the fact that many parameters cannot be set from the user interface but are hardcoded in the source code and require at least a basic understanding of the system to be changed.

The current system enforces unique item codes even if the items are in different projects. This might become a problem at some later point in the database when multiple projects are running in parallel and a user cannot create an item because of a duplicate item code but has no way of seeing the item due to no access of the project where the duplicate item belongs to.

Chapter 3

Beam test

To gain information about sensor properties beyond the possibilities offered by a typical lab setup, it is common in high energy particle physics to perform beam tests at particle accelerator laboratories. Multiple particle accelerators offer the possibility for research groups to reserve a beam test area and perform experiments with the provided beam.

The type of beam provided depends on the accelerator type and therefore on the location visited.

All measurements discussed in this thesis were performed during beam tests for the Belle II group at the SPS[1] which is located at CERN[2] in Geneva, Switzerland.

3.1 Sensors

For this thesis only sensors designed for layer four and above were tested.

3.1.1 Layout

Rectangular sensor

The rectangular sensor is designed for the barrel of the detector. The parameter specifications for the sensors designated for the layers 4-6 type are given in the tables 3.1 and 3.2. A visual representation of the geometry can be found in fig. 3.2.

For both beam tests the sensors were assembled in Pisa, put into a transport casing and taken to CERN for testing(fig. 3.1).

The strips on each side of this sensor are straight and parallel but perpendicular to the strips on the other side. On the n-side the strips are parallel to the short side of the sensor and have a pitch of 240 µm measured from the center of one strip to the center of the next. Each strip is connected to one input of an APV25 on

[1] Super Proton Synchrotron

[2] Conseil Européen pour la Recherche Nucléaire (European Organization for Nuclear Research)

Quantity	Value
# strips n-side	512
# strips p-side	768
Pitch n-side	240 µm
Pitch p-side	75 µm
Area (total)	7442.85 mm^2
Area (active)	7029.88 mm^2 (94.5 %)

Table 3.1: Geometric parameters of the rectangular sensor[2]

Quantity	Value
Base material	n-type Si, 8 kΩ cm
Full depletion voltage V_{FD}	< 120 V
Breakdown voltage	$\geq V_{FD} + 50$ V
Polysilicon resistor	4 MΩ(min.), 10 MΩ(typ.)
Coupling capacitance	> 100 pF
Breakdown voltage of AC coupling	> 20 V
Bias dark current of V_{FD}	1 µA(typ.), 10 µA(max.)

Table 3.2: Electrical parameters of the rectangular sensor[2]

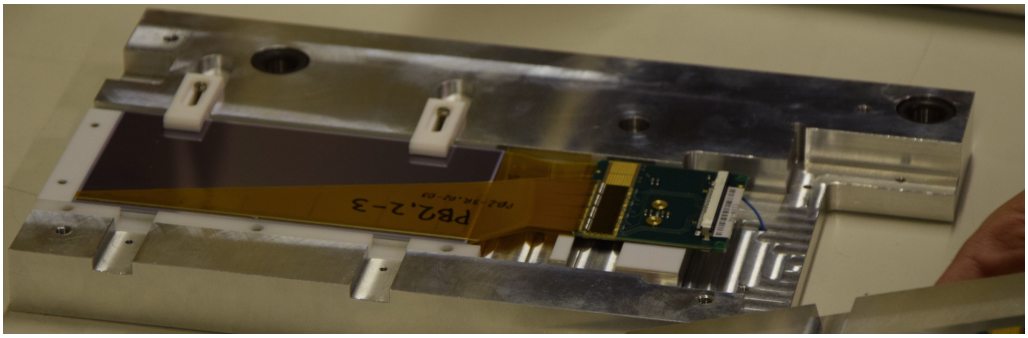


Figure 3.1: Picture of the SFB993 in the transport casing taken at the november beam test.

a hybrid board via wire bonds and a pitch adapter. On the p-side the strips are parallel to the long side of the sensor and have a pitch of 75 µm. Aside from that the configuration is the same as on the n-side.

Trapezoidal sensor

The trapezoidal sensor, sometimes called wedge, is designed for the forward, angled part of the SVD and the same layout is used for all layers with a forward sensor (layers 4-6). The parameter specifications can be found in tables 3.3 and 3.4.

These sensors are also assembled in Pisa and taken to CERN in a transport casing (fig. 3.3).

The strips on this sensor have a more complicated design than the ones of the rectangular sensor. The strips on the n-side are parallel to the short side of the sensor with a constant pitch of 240 µm, but the length of the strips varies from

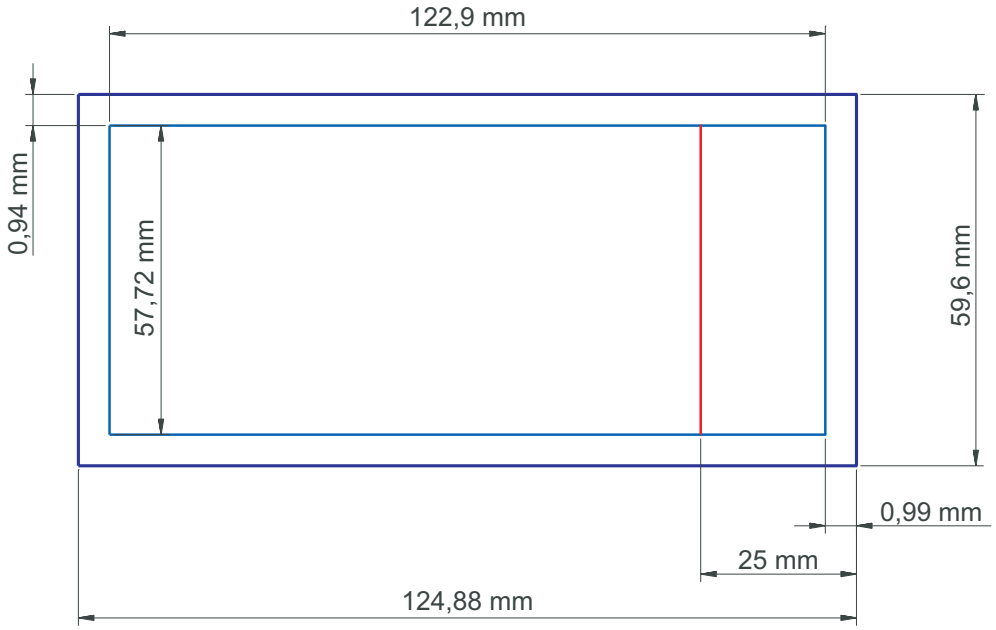


Figure 3.2: Geometric dimensions (not to scale) of the rectangular sensor for layers 4 to 6. Dark blue: outer dimensions. Light blue: active area. Red: additional pad row.[2]

Quantity	Value
# strips n-side	512
# strips p-side	768
Pitch n-side	240 µm
Pitch p-side	75 . . . 50 µm
Area (total)	6382.6 mm^2
Area (active)	5890.0 mm^2 (94.5 %)

Table 3.3: Geometric parameters of the trapezoidal sensor[2]

Quantity	Value
Base material	n-type Si, 8 kΩ cm
Full depletion voltage V_{FD}	40 V(typ.),70 V(max.)
Operation voltage	V_D...2 × V_D
Breakdown voltage	$\geq 2.5 \times V_{FD}$
Polysilicon resistor	10 MΩ(min.), 15(5) MΩ(max.)
Interstrip resistance, p-side	100 MΩ(min.),1 GΩ(typ.)
Interstrip resistance, n-side	10 MΩ(min.),100 MΩ(typ.)

Table 3.4: Electrical parameters of the trapezoidal sensor[[2]]

38.42 mm to 57.59 mm. On the p-side the pitch of the strips varies from 75 µm on the wider side to 50 µm on the slimmer side leading to the strips not being parallel. This is easily visible in fig. 3.4. Again each strip is connected to the input of an APV25 chip via wire bonds and pitch adapters on both sides.

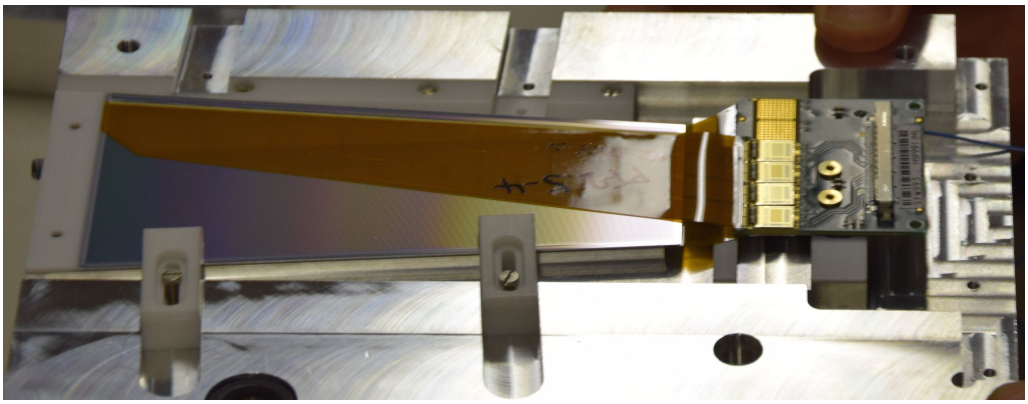


Figure 3.3: Picture of the SFW993 in the transport casing taken at the november beam test.

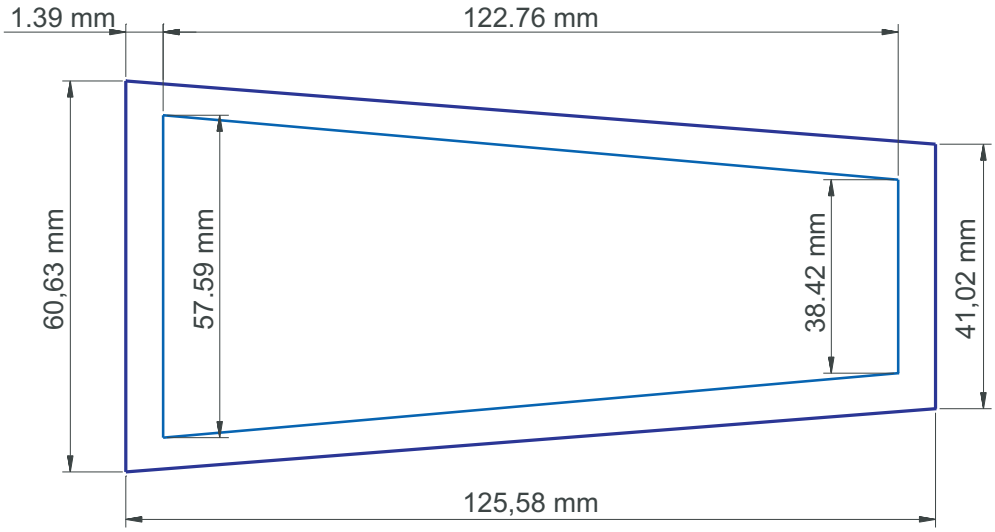


Figure 3.4: Geometric dimensions (not to scale) of the trapezoidal sensor. Dark blue: outer dimensions. Light blue: active area.[2]

3.1.2 Signal readout

The signal is stored analog in the readout chip (APV25) and then transmitted as a series of multiplexed analog signals to the DAQ. For a more detailed description please refer to section 1.5.2.

3.2 Setup

The overall measurement setup during the beam test consisted of the following parts (fig. 3.5):

- EUDET Telescope, provided by CERN
- TLU and control computers running EUDAQ, provided by CERN
- x-y stage, dubbed DESY table, provided by CERN

- DUTs to be tested, provided by the Belle II collaboration
- SVD3 readout system, brought with us from HEPHY
- SVD3 dock box, brought with us from HEPHY
- SVD3 voltage supplies, brought with us from HEPHY

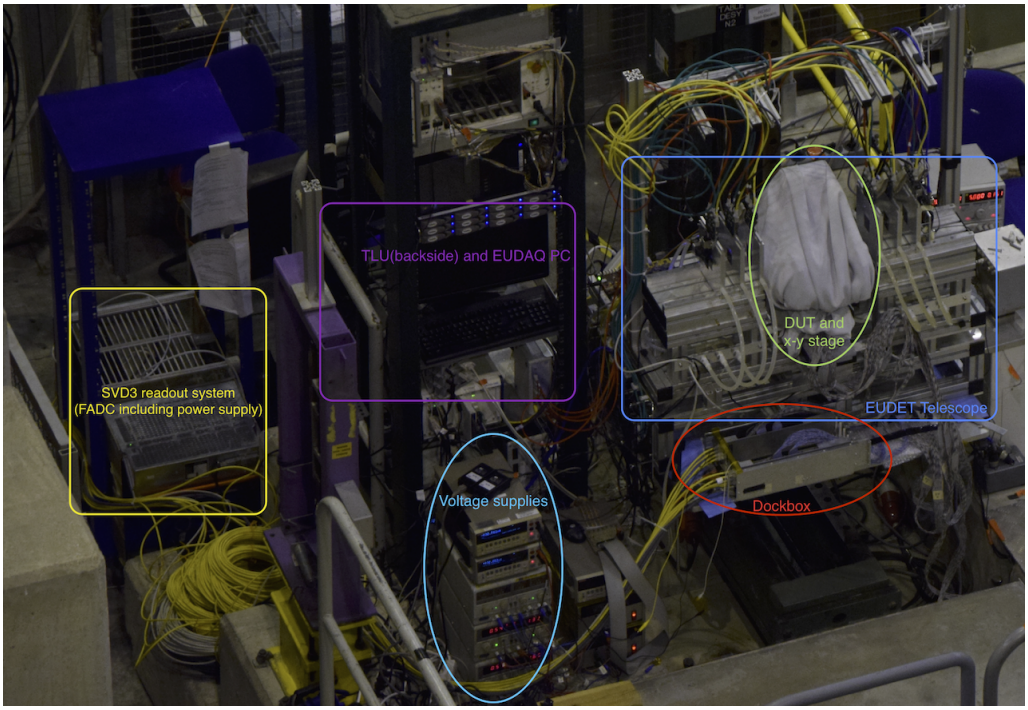


Figure 3.5: Overview of the beam test setup in November 2014.

3.2.1 SVD3

The SVD3 (Silicon Vertex Detector 3) readout system was a prototype system for an upgraded SVD of Belle before a complete redesign of the detector into Belle II was decided. It has been functional for a long time and is still used in smaller variants for sensor tests performed in laboratories and also during beam tests for other APV25 based sensors.

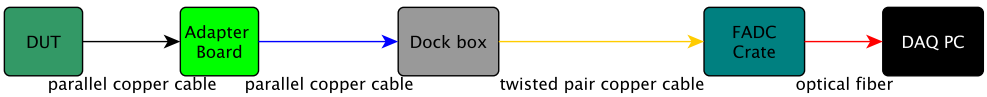


Figure 3.6: The readout chain of the SVD3 setup as it was for the SVD. For beam test setups only one junction box was used which is capable of connecting a maximum of 16 APV25 chips on both n- and p-side.

Since it is only a prototype system it requires some in-depth knowledge of the system and a lot of components to work. The main components are

- VME-crate with one Buffer, one NECO and two FADC (n and p sides)
- Junction Box
- Two standard SMU HV power supplies with cables
- Five standard power supplies for various voltages (8 total voltages required)
- About 12 long cables for data communication
- Custom made cables for voltage supply and HV cables for HV supply

3.2.2 Beam telescope

A beam telescope is intended to be used as a baseline to analyze the behavior of a sensor against. It is usually provided by the beam test location.

EUDET/AIDA Telescope

The development of the EUDET Pixel Telescope is part of the efforts of the EUDET project, supported by the "European Union" in the "6^{th} Framework Program"(FP6), named sub-project JRA1. It is designed as a six-plane telescope with an area for DUTs[3] between each set of three planes. Each telescope plane contains a MIMOSA26 pixel sensor within a water-cooled aluminum structure and a readout hybrid(EUDET Data Reduction Board) which creates zero-suppressed data. For a more detailed description please refer to [8].

The telescope is designed to be a stand-alone solution for beam test analysis and comes with all six sensor planes including the mounting structure, a DAQ called EUDAQ for data storage, an analysis framework called EUTelescope for data analysis and a trigger logic unit(TLU, see next section) for a common trigger for telescope and DUT.

In fig. 3.7 the whole setup is shown with the six aluminum squares being the telescope planes and the DUTs in the center from left to right: Old Wedge, SFW993 and SBW993. The DUTs are mounted on an x-y stage capable of movements with a precision of a few µm.

TLU

The Trigger Logic Unit was developed for the telescope and is designed to act as a common trigger for the telescope planes and any DUTs placed within the telescope.

It can operate in three different modes with the third one being used by the telescope planes and also being recommended for other devices since it allows for the trigger number to be stored along with the trigger.

[3]DUT: Device Under Test

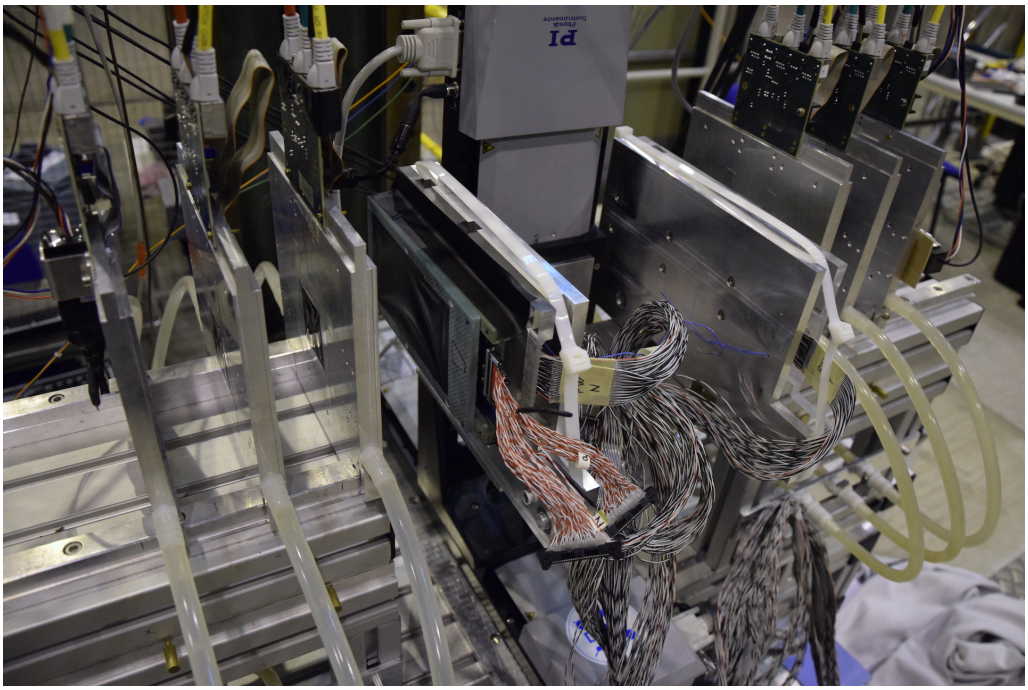


Figure 3.7: Picture of the modules after they were placed inside of the telescope.

The three modes are:

- **Without handshake:** Trigger is sent by the TLU whenever the trigger condition is fulfilled
- **With handshake:** Allows connected DAQ systems to raise a busy signal to prevent triggers from being sent when they cannot process it.
- **With handshake and data:** Allows, additionally to the blocking signal, for the trigger number to be sent along with the trigger impulse so the DAQ can store it along with the data.

Storing the trigger number (TLUEventNr) along with the data is important to make sure that the events of both data-streams are synchronized.

The SVD3 system contains space in the raw data format for this TLUEventNr and is therefore capable of storing it along with the data. The TLUEventNr is transmitted using a 16 bit field with the first bit being an error bit, therefore only 15 bits are available leading to a maximum trigger value of 32767. After this the TLUEventNr again starts at 0. This needs to be considered when interpreting the data since the values stored in the raw data need to be transformed using something like

$$TLUEventNr = int(previousEventNr/32767) \cdot 32767 + TLUEventNr;$$
$$previousEventNr = TLUEventNr;$$

to actually represent the real value. The corrected value is calculated during the analysis with TuxOA(4.2.1) and stored in the root files.

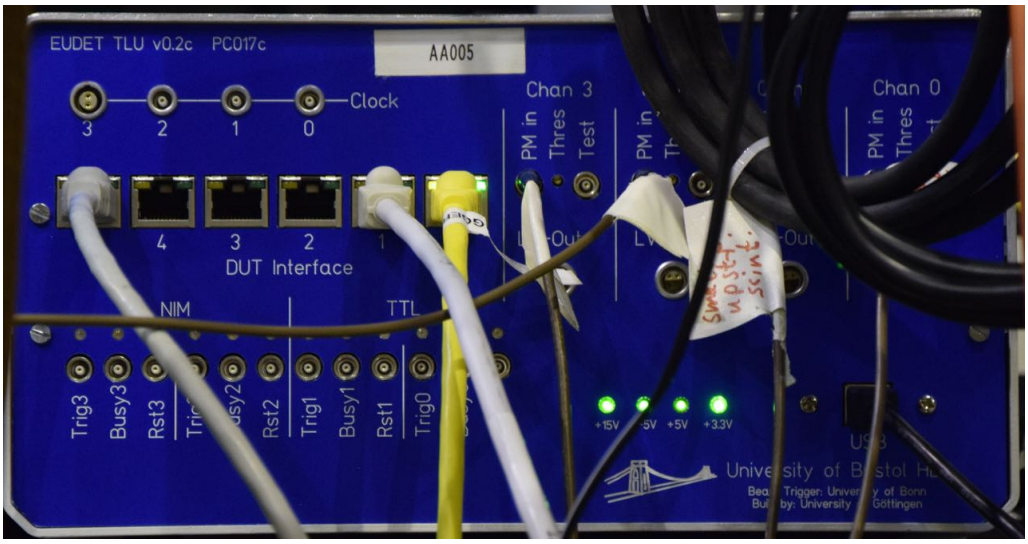


Figure 3.8: Picture of the TLU that was used.

3.2.3 Beam test November 2014

Leading up to this beam test the plan was to measure the two sensors SFW993 and SBW993 using the new readout system that was also planned to be tested during the beam test. Due to complications with the new system the plan was changed for the FW[4] and BW[5] sensors to be tested only with the old SVD3 system but inside of a beam telescope. The beam telescope was only available for one afternoon and as a consequence the amount of data that was taken is limited to one calibration, one pedestal and three hardware runs. It also became apparent during analysis that there are some problems with the recorded runs and that the information that can be gained from them is limited.

The aim was to perform resolution studies on the sensors, especially the wedge sensors where the pitch changes across the sensor and noise studies with respect to the changing strip length.

The expectation would be an increase in resolution with smaller pitch for the p-side while also gaining a better SNR on the n-side in that area. As described in section 1.5.5 a strips SNR increases with strip capacity, mainly length which in turn leads to shorter strips having a lower noise while still having the same signal strength which depends mostly on the thickness of the sensor.

To investigate these phenomena it would have been necessary to record data from different areas on the sensor where the different effects are visible, in this case moving the sensors from the left to the right. Due to time constraints and the short notice of the availability of the telescope, these considerations weren't made during the beam test leading to measurements taken in three different areas while moving the sensors from the top to the bottom.

Furthermore due to the usage of the old SVD3 readout system, which only allows a maximum of four APV25s per sensor side, two of the six APV25s of the p-side

[4]The wedge sensors are usually dubbed ForWard sensor due to their position in the detector.
[5]BackWard

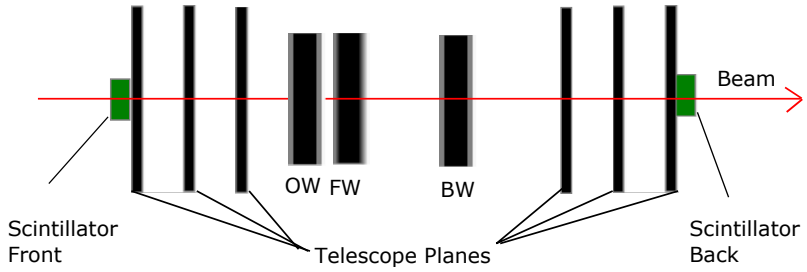


Figure 3.9: Layout of the telescope with DUTs (not to scale) during the first beam test

could not be connected and their data therefore was not recorded. The first of the three runs therefore basically only contains a small slice of particles instead of a full beam spot.

Despite all this, the opportunity was taken to install and use the EUTelescope framework and understand how it can be used to analyze the data taken with our DAQ and to get prepared for future measurements involving the telescope.

The sensors used during this beam test were one of the rectangular $122.8 \times 57.6\,\mathrm{mm}^2$ (SBW993) and one of the wedge 122.8 mm × 57.6 to 38.4 mm (SFW993) sensors. The sensors were classified as "Class B", electrically working with some minor errors. Additionally an older sensor was used for comparison, dubbed "Old Wedge", which is identical to the FW sensors, with the exception of some minor improvements introduced later, hence "old".

As mentioned before, this setup was only available on the last day. This thankfully led to all configuration required to make APVDAQ-FADC[6] work with the TLU and the telescope already being done. It was only required to replace the DUTs and configure them in APVDAQ-FADC.

Measurements conducted

Aside from the default software and calibration runs three hardware runs with TLU triggering and telescope data were recorded.

Run	Position	Number of Events	Useable
03	bottom center	100 000	No, because area not read out
04	middle center	60 000	Yes, best run available
05	top center	10 000	Yes, but very few events

Table 3.5: Runs during the first beam test

During the analysis the flaws with the measurements became apparent and the second beam test was included in the schedule for this sensor analysis.

[6] APVDAQ-FADC is the data acquisition software developed for the SVD3 hardware which consists of FADC boards hence the name.

3.2.4 Beam test June 2015

A beam test scheduled for June of 2015 was taken as an additional opportunity to take data this time as the sole user of the telescope and with lots of time available. This allowed for plenty of measurements of the DUT at many positions and additionally gave the opportunity to use an additional pixel sensor with a readout time equal to the readout time of our DUTs which was thought to maybe improve the resolution calculations since the DUT track would be well defined.

Setup differences

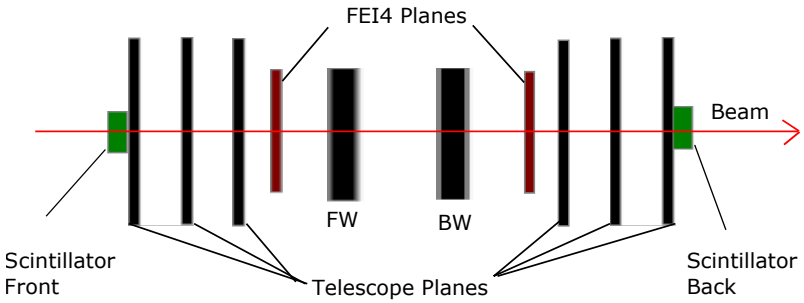


Figure 3.10: Layout of the telescope with DUTs (not to scale) during the second beam test

Fundamentally the setup of the second beam test doesn't differ much from the first one. The readout system and voltage supplies were built up in a different place and some different power supplies were used.

Regarding the DUTs the used modules were different ones but again class B and manufactured in Pisa. Again one FW and one BW module were used in the aluminum casing intended for transportation.

A change to the setup was, that an additional pixel plane was provided. This pixel plane contains one of the ATLAS experiment's pixel sensors (FE-I4) with rectangular pixels. In table 3.6 the geometrical details of the sensor are given.

Item	Value	Units
Pixel size	50 x 250	μm^2
Pixel array size	80 × 336	Col × Row
Hit-trigger association resolution	25	ns

Table 3.6: Selected parameters of the FE-I4 ATLAS pixel sensor [9]

The most important detail is the low readout time of only 25 ns that equals the readout time of the APV25 chip since they were both designed for LHC timing. It should therefore be possible to use the FE-I4 plane to select the track within all the telescope tracks that equals the one stored by the DUT.

Measurements conducted

To have many points of comparison multiple spots on the wedge sensor were defined and then measured. The spots can be found in figure 3.11. Each spot was measured once without using the FE-I4 and then all but 6 and 9 again with the FE-I4.

At least 50 000 Events were recorded for each run, sometimes more overnight and then analyzed starting already during the beam test. The results can be found in the next chapter. The first run from this beam test has the number 49 with the following numbered consecutively.

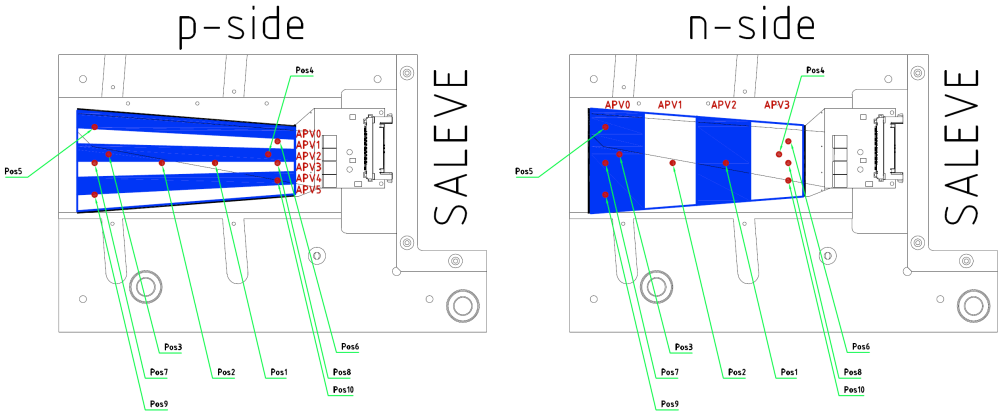
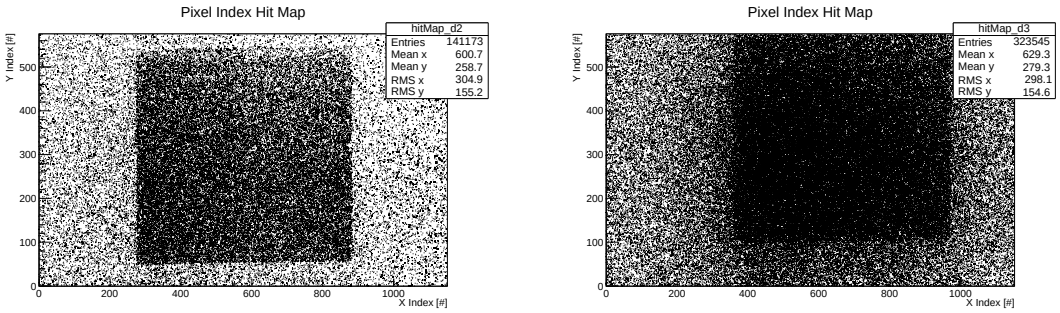


Figure 3.11: Drawing of the forward module with the 10 points measured on it. This picture can also be used to find the reference points on the backward sensor since they were almost perfectly aligned.

Being more knowledgeable about the telescope software also helped with investigating additional phenomena such as the influence of the aluminium casing on the beam and the amount of secondary particle emission. As can be seen in fig. 3.12 the plane right after the DUTs in their aluminum casing contains more than twice as many hits than the one in front of the DUTs.

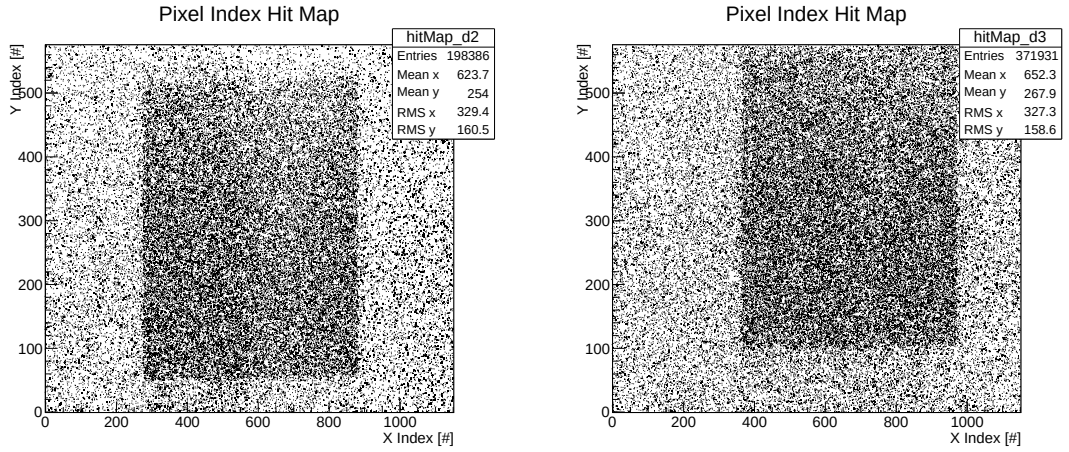


(a) 3rd plane (in front of DUT) (b) 4th plane (behind DUT)

Figure 3.12: Comparison of the hitmaps of telescope planes in front and behind the DUTs and their aluminum casing.

As a comparison a run was taken after the DUTs had been removed again whose results can be seen in fig. 3.13. Here the plane after the DUT area contains only about 85% more hits than the one in front of the DUT area which can be attributed

to the different noise levels of the sensors. Additionally a visual comparison shows that also the shape of the trigger window is better visible without the scattering induced by the DUT casing.



(a) 3rd plane (in front of DUT) (b) 4th plane (behind DUT)

Figure 3.13: Comparison of the hitmaps of telescope planes in front and behind the DUT space without a DUT in between.

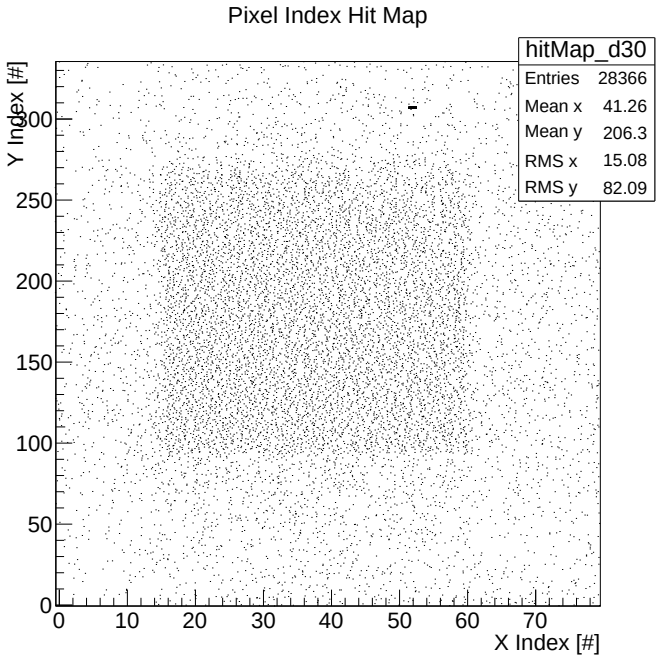


Figure 3.14: The hitmap of the FE-I4 plane with its lower integration time.

Chapter 4

Data analysis

During each beam test two separate streams of data were recorded, one with the EUDAQ of the telescope planes and one with APVDAQ-FADC for the SVD3 system of the DUTs. Each DAQ has different requirements on data processing before the data can be analyzed.

The first section describes the APVDAQ-FADC data and what processing needs to be performed to get actual hit signals.

In the second section I give an explanation of the actions performed by TuxOA and the transformations required before the analyzed data can be used with EUTelescope.

4.1 Basic sensor properties

The results of a run performed with the SVD3 setup are recorded by APVDAQ-FADC and stored in a raw data file. In order to find particle hits on the sensor the raw data needs to be processed, noise reduced and then have limits applied to it to determine if a strip detected a particle.

4.1.1 Signal & Noise

The signals recorded are amounts of charge deposited at an APV25 entrance as described in 3.1.2. The APV25 can mark multiple cells for readout which can be used during a hardware run to gain more data samples. In this case a subset of six samples was used which translates to one particles interaction being recorded in six subsamples with the maximum of the curve calibrated between the second and third peak (fig. 4.1).

Using six data packets allows for a more precise measurement of the maximum peak because the shaping curve can be fitted over the data to calculate the peak from the six data points. Additionally the position of the maximum could be used to distinguish particle hits from ghost hits.

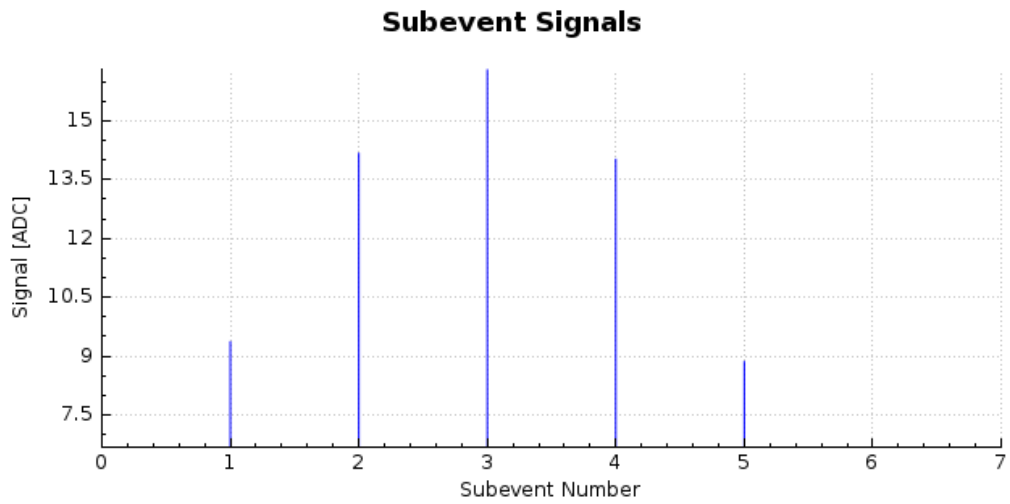


Figure 4.1: Sample plot of 6 subevents with the maximum at the third peak

The analysis of the APVDAQ-FADC data can be performed with multiple programs that were programmed with different goals in the past. The one used by me is called TuxOA and is designed to be used by Belle II for online analysis. It is also capable of offline analysis of two different raw data formats, one of which is the APVDAQ-FADC raw data format.

In fig. 4.6 I give an overview over the steps required to perform a full analysis of the raw data files recorded by APVDAQ-FADC to generate a root file containing only recognized hits for each recorded event. One hit in this context equals a single strip or cluster of strips on one sensor plane identified by the strip number. The data can be plotted using TuxRoot[6] to generate a wide variety of plots like noise maps, pedestal maps and calibration constant maps.

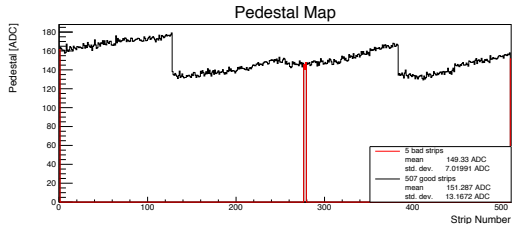
Pedestal map

The first step in the analysis chain is the calculation of the pedestal values for each strip. The pedestal is defined as the mean signal height of a number of events recorded with a random trigger and represents the signal height of a strip without any particle interaction. The pedestal value should not change much over one APV25 but can vary between different APV25s. In fig. 4.2 two pedestal maps are given as examples.

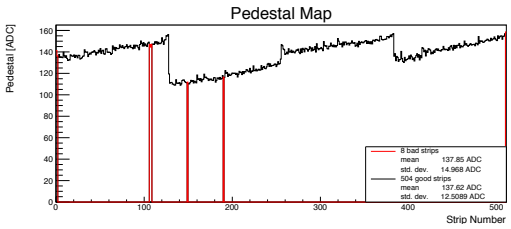
Noise map

To find damaged strips, the noise of each strip is calculated and if it is above a threshold, the strip is marked as noisy and excluded from the analysis. To calculate the noise for each strip, a gaussian fit is performed within the FWHM[1] of the signal after pedestal and common mode correction. If the calculated noise is greater than 2.5× the mean noise of the chip, the strip is marked as noisy and excluded from future analysis.

[1] Full Width Half Maximum

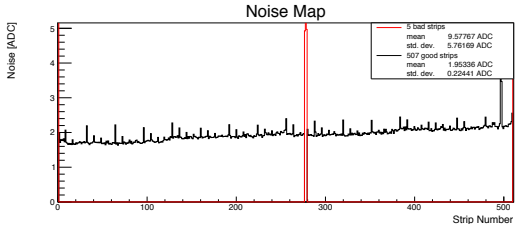


(a) BW module, n-side

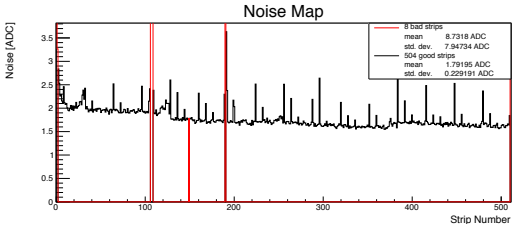


(b) FW module, n-side

Figure 4.2: Two pedestal maps. Note the linearity except the jumps at 128 steps where the APV25 changes. Red vertical lines are strips that have been marked as faulty and are excluded from further calculations.



(a) BW module, n-side

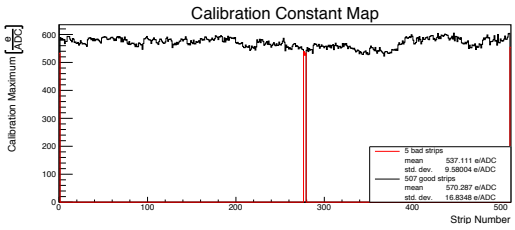


(b) FW module, n-side

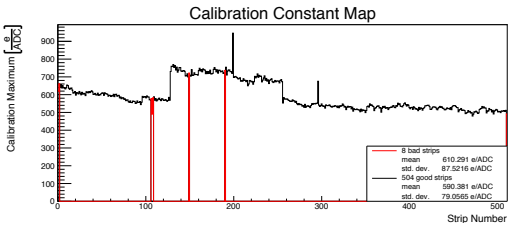
Figure 4.3: Two noise maps. Red vertical strips mark deactivated strips

Calibration constant map

During a calibration run specific charge amounts are injected into each APV25 and the result is recorded generating calibration values for each strip. Using this file TuxOA calculates the calibration values and generates a TuxOA calibration file. The analysis generates a calibration file which contains the calibration curve for each strip in fitted parameters that define the amount of ADC[2] units one defined charge deposit generates on each APV25 input. The calibration constant map shows the amount of electrons deposited, divided by the maximum of the calibration curve for each strip.



(a) BW module, n-side



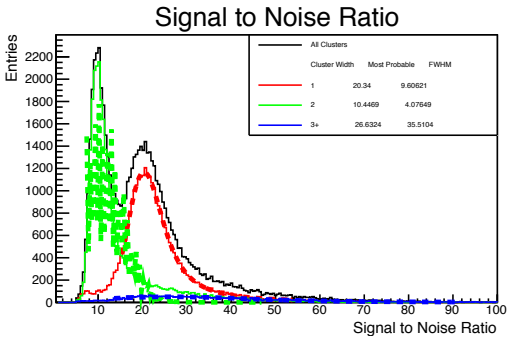
(b) FW module, n-side

Figure 4.4: Two calibration maps.

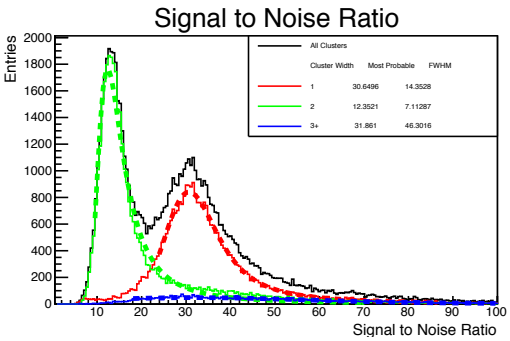
[2]Analog Digital Charge, the amount of charged particles required to increase the digital count by one.

4.1.2 Signal to Noise behavior

As described in section 1.5.5 the noise depends heavily on the capacity of a channel and therefore on the strip length. It is therefore interesting to look at the differences in SNR on the n-side where the strip length varies over the sensor. The expected result would be, that on the wide side the SNR is worse than on the narrow side since the strips are longer (57.59 mm vs 38.42 mm) which is exactly the received result as can be seen in fig. 4.5 and table 4.1.



(a) SNR plot of the wide side. The Landau fit fails for this narrow shape.



(b) SNR plot of the narrow side.

Figure 4.5: Wedge sensor comparison. It is easily visible that the SNR for the narrow side is a lot better due to the shorter strips.

Position	Strip length	SNR (Cluster width 1)	SNR (Cluster width 2)
5	57 mm	19.2	10.1
7	57 mm	19.8	9.8
9	57 mm	19.8	10.1
3	53 mm	19.8	9.8
2	48 mm	22.3	10.5
1	45 mm	24.2	10.7
4	41 mm	29.7	12.8
6	40 mm	29.6	12.6
8	40 mm	29.6	12.7
10	40 mm	29.8	12.8

Table 4.1: Comparison of the signal to noise ratios for different positions (fig. 3.11) on the n-side of the wedge sensor. The strip length is just an approximation as a reference.

4.2 Preprocessing of the data

In order to gain information about the resolution of the sensors the data needs to be converted, analyzed, correlated and analyzed again. To this end multiple steps need to be performed with a lot of parametrization possibilities along the way. In this section I will give an overview of the steps performed, the order they need to be performed in and the tools used.

In short the following steps need to be performed

- Loading of raw APVDAQ-FADC data
- Analysis of APVDAQ-FADC data, gives zero-suppressed hits
- Correlation of strip signals to pixel signals, gives hits on sensor in mm
- Conversion of raw EUDAQ data to telescope data
- Analysis of telescope data
- Combining of telescope data and APVDAQ-FADC (DUT) data
- Combined analysis of all data for resolution plots

4.2.1 TuxOA

Developed by Hao Yin as part of his master's thesis [6] TuxOA is an analysis software designed for online monitoring of the Belle II data but it is also capable of doing offline analysis of raw data files created by either TuxDAQ or APVDAQ-FADC. It is therefore capable of performing the analysis required for the recorded data.

Due to the data format changing between APVDAQ-FADC and TuxDAQ additional actions are required to analyze data originating from APVDAQ-FADC with TuxOA. Fig. 4.6 gives an overview over all the actions required to convert raw APVDAQ-FADC data files to root files containing the analyzed data.

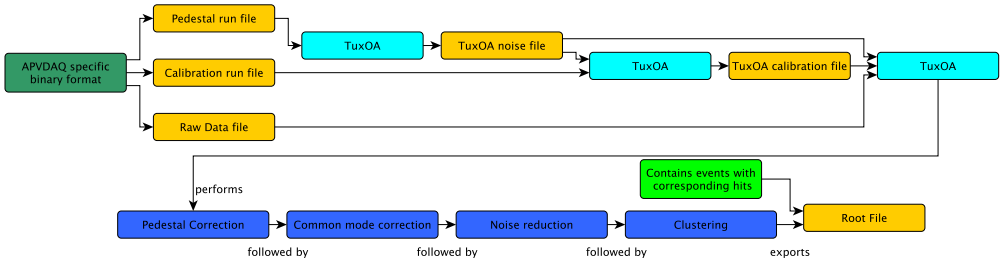


Figure 4.6: Analysis chain performed by TuxOA to get from the raw APVDAQ data to a root file containing analyzed events.

The first step is to generate a noise file for TuxOA. This file contains pedestal and gauss fitted noise values for each strip of each APV25 and also marks each strip as good or bad. During a noise run this file is generated for the run and the strips are automatically graded depending on the parameters set for the noise run. After doing the calibration run it is also possible to do a hardware run using the pedestal run file as input file. Since the data should be statistically evenly spread, faulty strips that could not be determined with the algorithm can then be marked as defective by the user and the noise file updated.

After the creation of a noise file a calibration run needs to be performed. This requires the APVDAQ-FADC calibration file and the TuxOA noise file. In this step a calibration curve is fitted over the values returned by the APV25 after a defined

electron injection (3.1.2) resulting in calibration values that are then stored in the TuxOA calibration file.

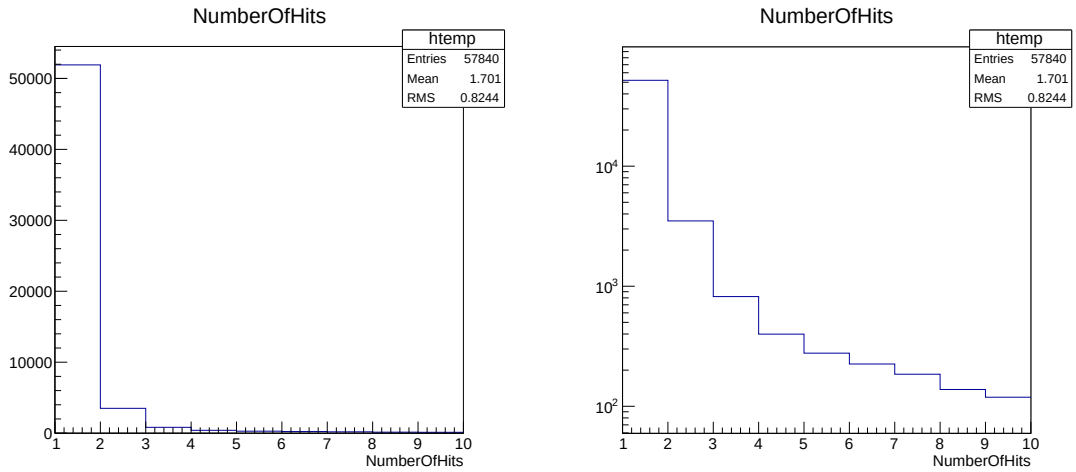
As mentioned before one can now perform a hardware run using the pedestal file as input to mark additional strips as bad that weren't detected by the noise algorithm. After this is done the final hardware run can be performed.

The hardware run uses the raw hardware run data file created by APVDAQ-FADC and the noise and calibration files created in the previous steps to analyze the recorded data and determine hits, their cluster width and η values. This analyzed data can be stored in root files that can then be used by multiple scripts. Another script (TuxRoot) allows to generate all the standard plots from the root file; samples can be seen in section 4.1.

The root file generated by TuxOA was the starting point for my analysis. I implemented an additional parameter to be stored in the root file, the TLUEventNr, that was previously only read in by TuxOA but not stored in the root file. This parameter gives the event number as it is used by the EUDAQ data format and therefore enables matching the DUT events with the telescope events.

4.2.2 Strip-hit correlation

The usage of double sided strip sensors allows for a pixel interpretation of hits by correlating the x and y coordinates gained from each respective side to one set of data. This is mainly possible because the readout time of the APV25 is so low that multiple hits per event don't occur often enough to hinder the analysis(see fig. 4.7). In case of more than one hit being recognized during one event, this event needs to be discarded for this analysis since it is not possible to determine which of the hits on one side correlates with the hit(s) on the other side.



(a) Not logarithmic (b) Logarithmic

Figure 4.7: Histograms of run004 showing the frequency of occurrence of events with more than one hit. In this case, 54340 of the 57840 events recorded contained only one hit which amounts to a loss of about 6% of the events.

TuxOA stores a separate root file for each hybrid board(section 1.5.4) and therefore, the two root files for one sensor need to be opened at the same time and then looped over the events calculating the exact hit position whenever there is exactly one hit per event on both planes.

To calculate a more precise location for each hit, the eta distribution of all values is calculated and then used to correct the position of hits that created a cluster of two or more strips.

To perform all these actions several scripts were developed using Python. Different approaches to the problems were tested with the end result being a sequence of four scripts that could perform all the steps required if one managed to configure them all correctly and called them in the correct order.

Once it was clear that the functionality was given, all scripts were condensed into two: One that now performs all the actions that are specific to the root output file format of TuxOA and the way our sensors work, and the other, that is used as a module by the first script, that can be used standalone to store correlated hits in the *.slcio* files used by EUTelescope.

An overview of the interactions between the programs and the required inputs are given in fig. 4.8.

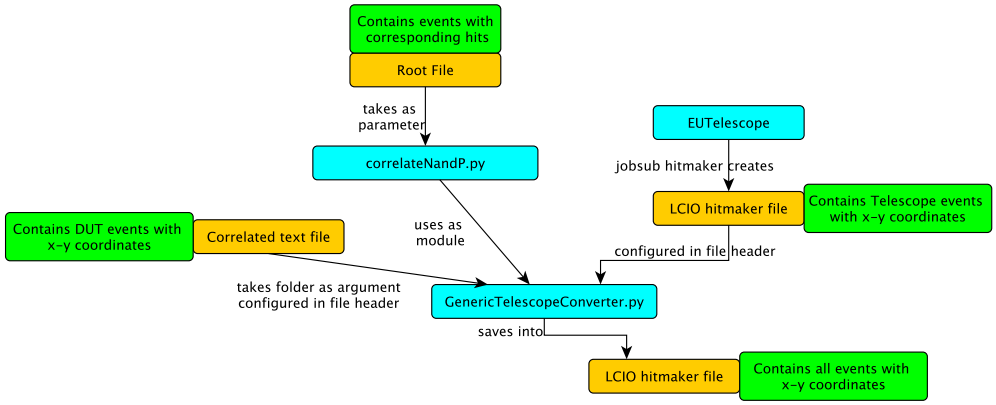


Figure 4.8: Analysis chain correlating the hits and inserting them into the *.slcio* file created by EUTelescope

All required parameters and configuration options need to be set in the header section of the *correlateNandP.py* script.

4.3 EUTelescope

The EUTelescope (http://eutelescope.web.cern.ch) is "A Generic Pixel Telescope Data Analysis Framework" maintained and supported by DESY. It was originally developed for the EUDET telescopes such as the one used during this beam test and is part of the ILCsoft framework that is being developed by the ILC[3] community.

[3]International Linear Collider

> The core elements of the framework are the Linear Collider I/O (LCIO) data model, the Geometry API for Reconstruction (GEAR) markup language and the event processor Modular Analysis Reconstruction for the LINear collider (Marlin).
>
> (EUTelescope 'About' page)

The structure of the framework is modular with a multitude of processors that can be called to act on data and perform various actions. For example, there is a simple processor that just displays the number of the event that is currently being processed, intelligently increasing the step size as the numbers get larger. One processor converts the raw data of each pixel into hit/no hit signals depending on thresholds. Another takes this data and forms clusters of pixels next to each other. Again another identifies clusters that are constantly giving a signal and marks them as noisy.

These processors can be selected and configured using so-called 'steering-files' using XML[4]. Marlin then takes these steering-files as input and calls the required processors in the order specified and provides the data each processor requires from the input files. Each steering-file is usually considered as performing a 'job' e.g. converter, hitmaker, fitter, ... and contains several processors and their configuration.

To perform geometric reconstruction, the layout of the experimental setup needs to be provided for the framework to use. This is done by using GEAR[5] files. Each plane of the setup needs to have a unique SensorID, x,y and z-coordinates as well as a rotation matrix describing the rotation with respect to the z-axis. Additionally, the size of the active sensor area and the (calculated) spacial resolution in x and y direction need to be provided.

To allow for a more flexible usage, jobs are usually not submitted directly to Marlin but via a job submitting program called 'jobsub'. It takes the configuration file and a file containing parameters for each run as parameters as well as the name of the job to be performed. Jobsub then replaces all variables defined in the steering file with the values defined in the config files and generates a finished Marlin XML file. If not prevented by a parameter, it then executes Marlin with the completed steering file and shows the output of the processors.

Event data calculated by the framework is stored in a file format known as *lcio*. It is event based and contains named 'collections' for each event. These collections can be read and written by Marlin processors. One can have a look at the content of a lcio file by performing a *dumpevent* command on it.

Additionally some processors that perform analyses create ROOT files containing their output data so the information can be further processed using the extensible ROOT data analysis framework provided by CERN.

The possibilities offered by the framework are theoretically only limited by the existing processors. Currently there are 90 processors available when installing the framework. Some of them are not state of the art anymore, some have different

[4] eXtensible Markup Language
[5] Geometry API for Reconstruction

approaches for the same goal, but everything that is required for standard track and resolution analysis is provided.

4.3.1 Analyzing with EUTelescope

After installing EUTelescope, following the guidance on the homepage, and setting the environment variables required for analysis, the following steps need to be performed. I will go into detail on some of them in following sections.

Converter Standard EUTelescope tool to convert EUDAQ raw data to lcio files.

Clustering Searches for clusters in the converted data.

Hitmaker Calculates hits from the found clusters considering noisy pixels. Creates a lcio file containing all hits on the telescope planes.

correlateNandP.py My script that takes the DUT data and stores the hits recognized there into the lcio file generated by the hitmaker. See section 4.3.2 for details.

Prealign The second part of the standard hitmaker script now performing prealignment on the telescope and DUT data.

Align Performs a DAF[6] fitting algorithm as well as a Millepede algorithm using both the telescope and the DUT hits. See section 4.3.3 for details

Fitter Uses the GEAR file generated by the alignment processor to again perform a DAF fitter, this time not using the DUT planes, and stores the residual plots in a root file. See section 4.3.4 for details and results.

4.3.2 Merging two separate data sources

Although it is possible to directly store the measurement data of the DUT during a run into a lcio file by writing a producer for EUDAQ, this was not done during the beam test. Therefore it was necessary to add the DUT events to the lcio files during analysis. To this end I wrote the *GenericTelescopeConverter.py* module in Python using a similar program simply called *TelescopeConverter.py* as a point of reference. The final script can now be used standalone by providing a text file containing at least three columns (TLUEventNumber,x-coordinate(in mm),y-coordinate (in mm)) and the path to the lcio hitmaker file to which the data should be added.

The requirements for the GenericTelescopeConverter are all included in the EUTelescope installation and are based on the pyLCIO implementation. It uses the *EventLoop* to loop over the lcio file and the standard *EventMarkerDriver* and *WriteLcioDriver* to display the progress and write the output file. To store the DUT events in the collection during the loop, I wrote an additional driver called *AddHitsDriver*. It requires a dictionary containing all DUTs with each DUT value being a dictionary that contains a tuple of 3 values (x,y,z) for each event number.

[6]Deterministic Annealing Filter

In principle the script loops over the the events, reading in the input collection (default: local_hit), copies the collection, loops over the DUTs and searches for a matching eventNr in each DUT dictionary. If such a set of values exists, it adds them to the new collection and upon completion, the new collection is stored into the event (default: hit_with_DUT_hits). During development some complications arose, some following from the usage of a python interface for a C library, others from the requirements for the implementation of the required classes.

For example, when copying the collection some parameters are not copied and need to be set manually again afterwards. Amongst these parameters are the id encoding string, the properties and of course the sensorID. This looping over the events led to some weird behavior that I attribute to some error in the python interface implementation. This led to me changing the code to just create a new TrackerHit collection from scratch and set the parameters by reading them from the old collection. Then I looped over the existing collection and created a copy of each hit within the event and saved that to the new collection. After this I looped over the events stored in the DUT dictionary and added them as well before storing the collection within the event. This process is less efficient than the intended one and requires more computation time, but under the circumstances I considered it the faster way to get to a result.

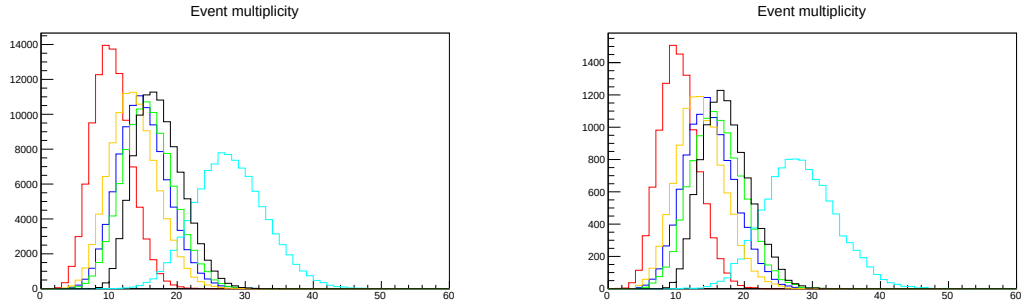
In the case of my analysis I use the GenericTelescopeConverter as a module to my main script, where the coordinates are calculated, and just pass the data to the converter without requiring a text file.

4.3.3 Alignment

This is one of the more complicated steps in the telescope analysis. The aim is to take one hit from each telescope and DUT plane, fit a particle track through it and thereby calculate the offset of each telescope plane from its defined zero point, which is at the center of the active area, and then in the end move the zero point to the actual one to remove the offset for further calculations. What doesn't sound so difficult in theory becomes quite difficult when one considers the following complications: Not every plane always records every event due to several reasons (i.e. particle absorption, scattering, ...), some "pixel" on the DUT could not be calculated because of multiple hits on one side (section 4.2.2) leaving again one plane without a hit. Furthermore the event multiplicity of the telescopes pixel planes is rather high as can be seen in fig. 4.9. This is due to the long readout time of around 150 µs. The high event multiplicity leads to even more complicated calculations.

The alignment step performs two separate alignment methods that are independent of each other. The first is called DAF which is a Kalman Filter running iteratively over a set of weighted measurements, reweighing the measurements after each fit based on provided residual values. The configuration values I used for my analysis were:

```
FitDuts             = true
NDutHits            = 2
RequireNTelPlanes   = 5
```



(a) event multiplicity of run 1008 (b) event multiplicity of run 1010

Figure 4.9: Comparison of event multiplicity between runs. The plots look almost exactly the same in spite of run 1008 having 10x as many events. Each color represents one plane. The plane with the higher multiplicity is the quite noisy 5th plane.

```
TelPlanes          = 0 1 2 3 4 5
DUTPlanes          = 10 11 12
```

For a converging fit it is important that there are two fixed planes defined, usually these are the first and last plane. I allowed a maximum of two missing hits per fitted track which leads to at least one DUT hit being a requirement for a fit.

The second alignment method is called Millepede and uses the second iteration of the Millepede algorithm that was initially developed by V. Blobel[10]. It uses linear least square fits and can deal with a large number of parameters. One of the applications of this algorithm is the alignment of all 24 000 sensors used in the CMS detector[11].

The configuration values I used for my analysis were:

```
UseResidualCuts      = 1
ResidualXMax         =   200.0  200.0  200.0  500.0  500.0  500.0  700.0  800.0  900.0
ResidualXMin         =  -200.0 -200.0 -200.0 -500.0 -500.0 -500.0 -700.0 -800.0 -900.0
ResidualYMax         =   200.0  200.0  200.0  500.0  500.0  500.0  700.0  800.0  900.0
ResidualYMin         =  -200.0 -200.0 -200.0 -500.0 -500.0 -500.0 -700.0 -800.0 -900.0
DistanceMax          =  500

AllowedMissingHits   = 2
ExcludePlanes        =
FixedPlanes          = 0 5
```

All residual parameters are in µm.

In the configuration I allowed two missing hits, thereby allowing for one DUTs alignment being improved even if the other two didn't record a hit for this event and are therefore excluded from the fit.

The parametrization of the alignment step was a lot of experimentation with different parameters. During my analysis, the EUTelescope framework was also released as version 1.0 with a changed way of handling alignment which actually solved a couple of problems that happened to me in the beginning, but it also introduced a new one.

The new alignment now works with updated GEAR files and calculating the new coordinates from the GEAR file instead of an extra database for shifted coordinates. This reduces the amount of sources that can cause errors.

4.3.4 Estimation of resolution

Digital resolution of a sensor

The simplest method of strip readout is to check if a value exceeds a threshold and, if yes, consider the signal a hit, otherwise discard the signal. If two neighboring strips exceed the limit they are considered a cluster. If only one strip signal is above the limit, the information about the position that can be gained from this is rather limited because the particle could have passed through an area ranging from $-\frac{\text{pitch}}{2}$ to $+\frac{\text{pitch}}{2}$ on either side of the strip center while only triggering this strip. This is known as the **digital resolution** of a sensor. The maximum resolution of such a sensor is given by the uniform distribution for this area (with p as the *pitch*)

$$\sigma^2 = \frac{1}{p}\int_{-\frac{p}{2}}^{+\frac{p}{2}} x^2 dx = \frac{p^2}{12}$$

Following from this, the digital resolution is defined as $\frac{\text{pitch}}{\sqrt{12}}$. This formula is widely used in high energy physics to quickly calculate the theoretical resolution of any given sensor.

Applying this formula to the pitches of the rectangular sensor (75 µm and 240 µm) one receives $\frac{75\,\mu\text{m}}{\sqrt{12}} = 21.65\,\mu\text{m}$ and $\frac{240\,\mu\text{m}}{\sqrt{12}} = 69.28\,\mu\text{m}$ as results.

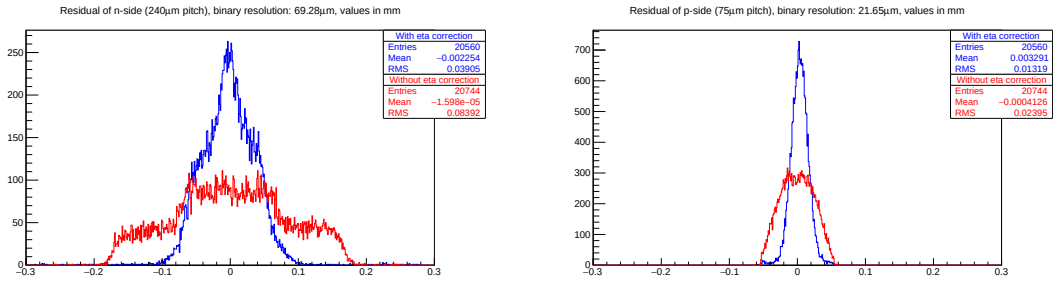
Improving the resolution

Using the amount of charge deposited at each strip within a cluster in relation to each other allows for a more precise calculation of the particle position. This factor, known as the η-value, calculated for each respective cluster creates a distribution function that can then be used to calculate a particle's position (see section 4.1.1).

In this case the eta distribution is used to calculate the position of each hit during the *correlateNandP.py* script where first the distribution is created by looping over many eta values and creating a histogram, using the *numpy* library which has an optimized function for that purpose. The generated histogram is then used by numerically integrating the histogram values up to the point of the eta value of the strip. The bin size for this histogram was set to 100 after seeing that the influence of a smaller bin size is non existent.

To check if my calculations are correct I changed the code to return the values without applying the eta correction and therefore not improving the resolution above the digital. The results (fig. 4.10) are a little bit higher than the theoretical binary resolution which is to be expected since the binary resolution would only consider clusters containing one strip and once the cluster size is greater than that, the resolution actually gets worse because the seed strip, which is used for this analysis, doesn't have to be the strip with the highest signal, it is just the first strip above the threshold.

To further investigate this I went ahead and tried to plot the distributions separately once for cluster width one, which should give me the central, flat distribution, and



(a) n-side (b) p-side

Figure 4.10: Improvement of resolution given by using the eta distribution.

once for cluster widths two and greater. The result was unexpected (fig. 4.11) and made me question my analysis since the two green curves together should add up to the red curve but don't even remotely do so. Additionally the calculated RMS value for the cluster width one residual is lower than the digital resolution and even lower than the calculated resolution of the whole system if all cluster sizes are considered.

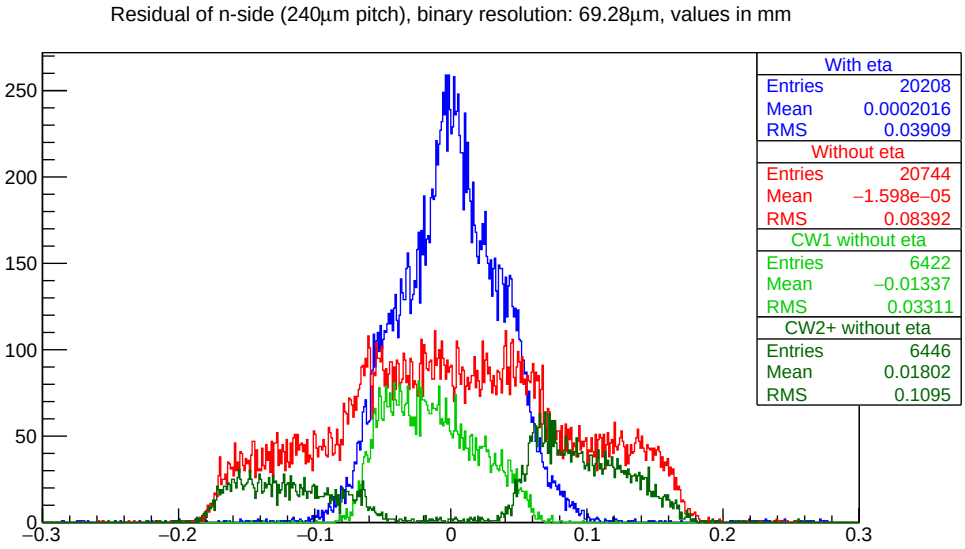


Figure 4.11: This figure shows which cluster width makes up which part of the total distribution. The two green curves should add together to the red curve but don't.

I tried multiple approaches to understand this phenomenon, especially the part of the two curves not adding up, but encountered only similarly weird results until at some point I understood why this was happening. The first step in the right direction was only deactivating the eta distribution for one sensor direction (i.e. x or y) which led to better plots and made me realize that I was preventing the track finding algorithm from finding a lot of tracks by not applying the eta correction to any plane. This was of course reinforced by the fact that two of the three DUT planes are not rectangular but wedge shaped. I was already only looking at the plots of the rectangular backward sensor to exclude other influences.

The moment I activated the eta correction for all planes but the n-side of the backward sensor and again split the plots between cluster widths the resulting plot was

(fig. 4.12)

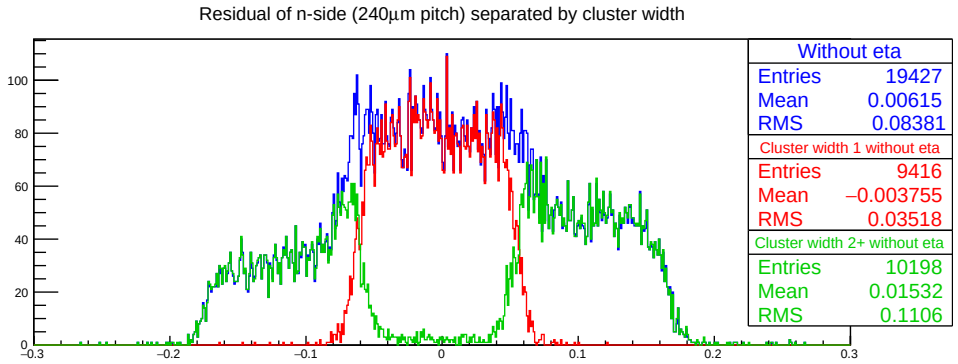


Figure 4.12: This figure shows how the residual without eta distribution is built up by the different cluster sizes.

Theoretically it should not be possible to get a better resolution than the digital ($\frac{\text{pitch}}{\sqrt{12}}$) for this kind of readout, therefore it came as a surprise that the RMS[7] for this analysis was far below the expected value (35 µm instead of 69.28 µm).

After some investigations and talks with people I understood that the way intermediate strips increase the resolution is based on the fact that if an intermediate strip absorbs charges it couples them to equal parts to the two neighboring strips which then couple the charge to the readout strip.

This reduces the pitch by 50% since the area where only one strip sends a signal is reduced to half of the distance between readout strip and intermediate strip on both sides of a strip ($\frac{\frac{\text{pitch}}{2}}{2} = \frac{\text{pitch}}{4}$ leading to the distribution ranging from $-\frac{\text{pitch}}{4}$ to $+\frac{\text{pitch}}{4} = \frac{\text{pitch}}{2}$). Therefore if one now considers the pitch to be not 240 µm but 120 µm the expected digital resolution becomes 34.64 µm which is not too far from the 35 µm in the plot. Further investigations into this topic on the other side of the sensor showed that this is actually the best case scenario. With a smaller pitch of 75 µm the effect is much smaller with a measured resolution of 15 µm instead of the calculated 21.65 µm an improvement of 30% over the digital resolution. For the narrow side of the wedge sensor, where the pitch is only 50 µm the improvement is actually lower than a micrometer.

Now for the last step it is also interesting to know what the distribution with η correction looks like when separated into the different contributions of the cluster widths(fig. 4.13).

Looking at the RMS values also shows, that the resolution of clusters with a size of one is actually better than for cluster width two and higher. Presumably this is all due to the higher amount of 'virtual' cluster width one hits using the intermediate strips.

Taking a look at the same plot for the y-axis it becomes apparent that this effect is also visible with a smaller pitch, although a lot smaller (fig. 4.13).

[7]Root Mean Square

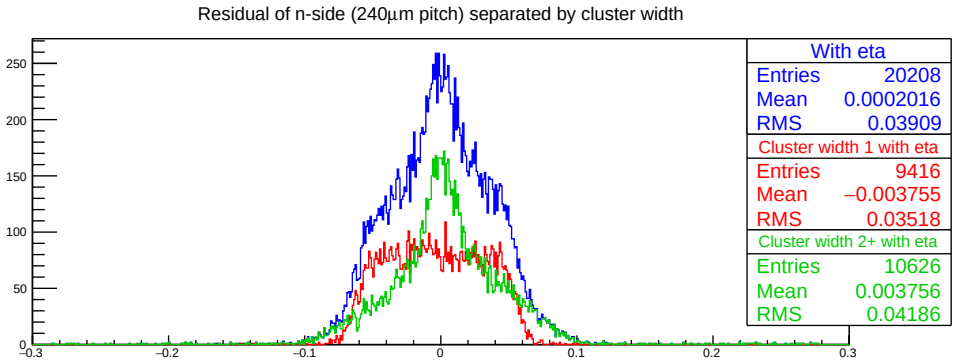


Figure 4.13: This figure shows how the residual with eta distribution is built up by the different cluster sizes. What is especially visible is that the application of the eta distribution onto the cluster width 1 events changed nothing in comparison to fig. 4.12 and the distribution is still uniform while the application onto the events with a higher cluster size forms the peak that is visible in the combined distribution.

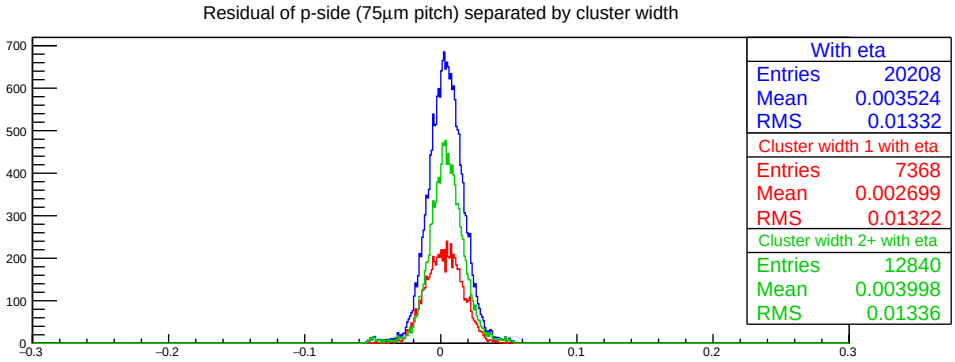


Figure 4.14: Residual with eta distribution built up by the different cluster sizes for the p-side

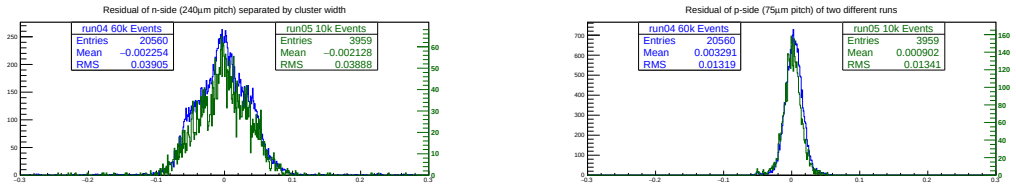
The resolution of a sensor is not only depending on the pitch of the strips and the existence of an intermediate strip but also on the signal to noise ratio of each sensor. This can be attributed to the fact that a low SNR leads to a worse recognition of clusters and can also shift the eta value away from the actual center of charge by influencing the signal height.

4.3.5 Resolution of each separate module and comparisons

As mentioned before the data taken during the first beam test was rather limited both in quantity as well as in knowledge that could be gained. Still some analysis was possible using these datasets.

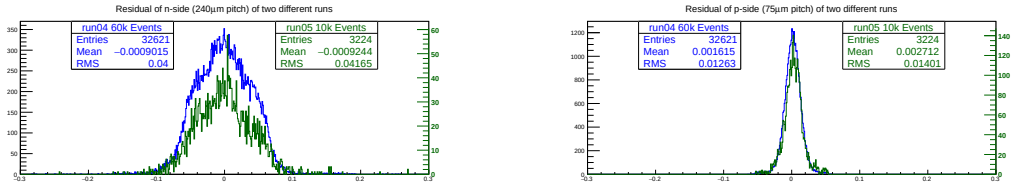
The plots in figure 4.15 and 4.16 show histograms of the resolution residual calculation as generated by EUTelescope. The resolution in given by the RMS value as it gives the mean offset in regard to the zero point. What is mainly visible in the comparison is that the higher amount of events in run004(60 000) provides a smoother histogram than the lower amount in run005(10 000). The effective influence this has

on the resolution is negligible, because the variation between the values is always in the sub-micrometer range.



(a) n-side (b) p-side

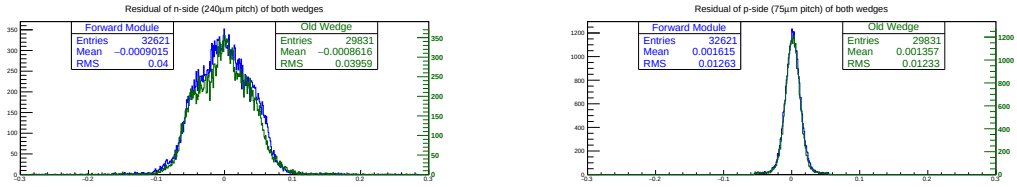
Figure 4.15: Compare resolution of the Backward Module between run004 and run005



(a) n-side (b) p-side

Figure 4.16: Compare resolution of the Forward Module between run004 and run005

The plots in figure 4.17 are comparisons between the old wedge and the forward module which contain the same sensor with the difference, that the forward module is one generation newer. Interestingly enough, the resolution for both the n and the p-side is slightly worse for the forward module, although again the absolute values are only in the range of sub micrometers.



(a) n-side (b) p-side

Figure 4.17: Compare resolution of Forward Module with Old Wedge in run004

The results, especially the comparison between the resolution of cluster width one events and cluster width two and greater events (with eta), show that analog readout at these pitches allows for wider clusters to deliver a comparable resolution to the theoretical cluster width one digital resolution (with consideration of the smaller pitch for these due to the intermediate strip).

The main objective for the second beam test was to check how the resolution of the wedge sensor behaves with changing pitch. The resolution of the wedge sensor should improve when moving from the wide to the narrow side since the pitch is reduced by 33% from 75 µm to 50 µm. To quantify this change multiple points on the sensors were measured (fig. 3.11) and then compared.

In figure 4.18 one can see the improvements between the wide and narrow side by comparing the resolutions of the sensor at the positions 05 and 10. The improvement from 15.24 µm to 12.72 µm results in a relative improvement of 16.53 % so nowhere near the theoretical improvement of 33 %.

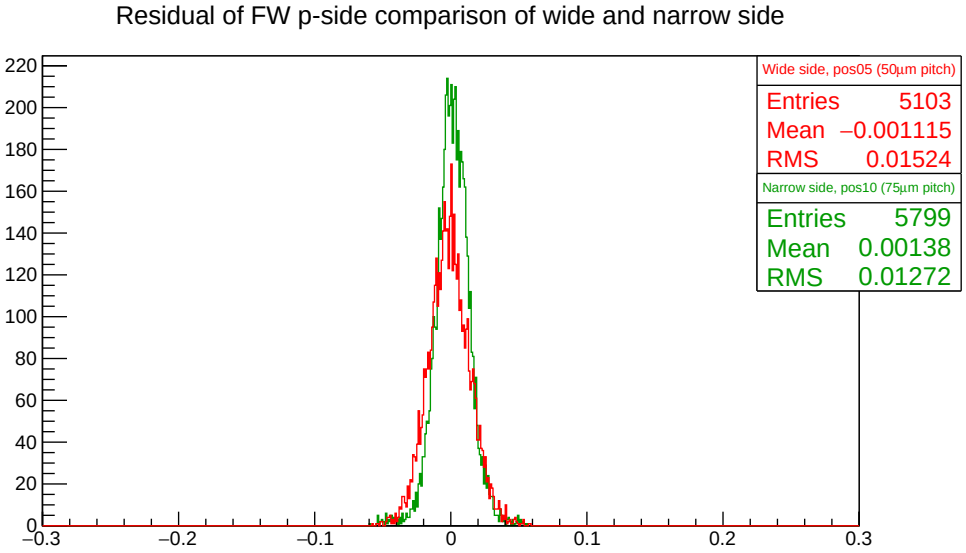


Figure 4.18: Comparison of the resolutions measured at the wide and narrow side of the wedge sensor.

The SNR also has a non-negligible influence on the resolution as can be seen in fig. 4.19 where the changing SNR also changes the resolution because the center of gravity calculation is more precise for a higher SNR.

In table 4.2 the results of the analysis are given with the digital resolutions and the resulting improvements from using the eta correction. All the values are taken with the FE-I4 plane included and forcing EUTel to only use tracks where each DUT plane had a hit. What can be seen from the results is, that the improvements to the resolution diminish the smaller the pitch is. Furthermore it is visible that the effective resolution between the wide and narrow side of the sensor differs only by 2.52 µm

Sensor	side	pitch	digital res.	measured res.	improvement
Wedge n-side	wide	240	69.28 µm	40.84 µm	41.05 %
Wedge n-side	narrow	240	69.28 µm	37.77 µm	45.48 %
Rectangular n-side	wide	240	69.28 µm	41.89 µm	39.53 %
Rectangular n-side	narrow	240	69.28 µm	39.94 µm	42.34 %
Wedge p-side	wide	75	21.65 µm	15.24 µm	29.40 %
Wedge p-side	narrow	50	14.43 µm	12.72 µm	11.80 %
Rectangular p-side	wide	75	21.65 µm	14.42 µm	33.30 %
Rectangular p-side	narrow	75	21.65 µm	14.10 µm	34.80 %

Table 4.2: Results of the residual plots.

Improvement of resolution visible between with and without FE-I4 plane, can be attributed to the fact that the fitter then has to use all 3 DUT planes which leads

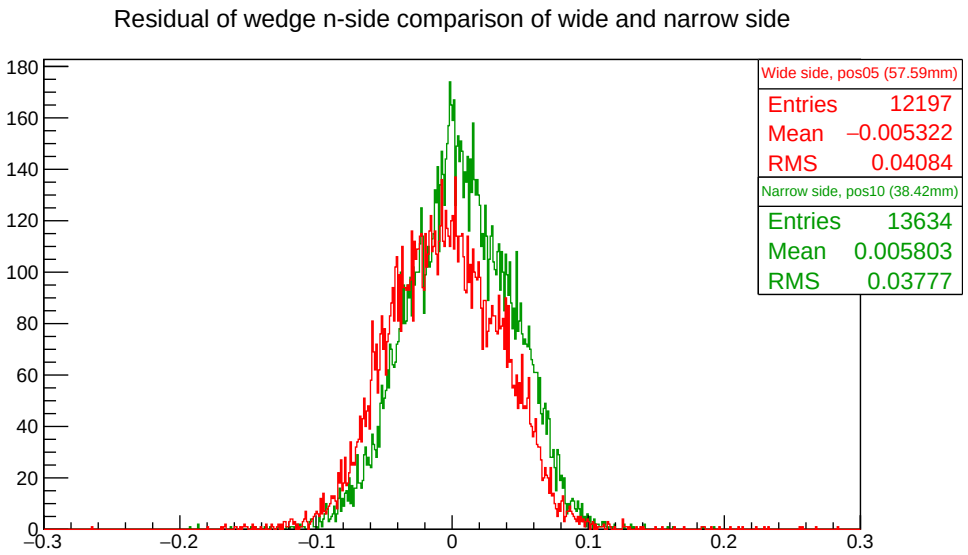
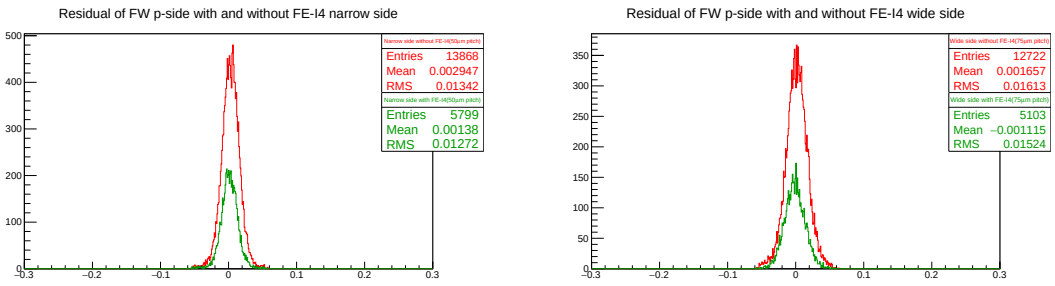


Figure 4.19: Comparison of the residuals on the narrow and wide side of the wedge sensor on the n-side. The change is the influence of the SNR on the resolution since the strip-length is the only thing changing in this comparison.

to fewer entries but a narrower result range altogether since an event has to contain a hit in all three DUT planes. An example of this can be seen in fig. 4.20. The limited amount of entries is based on the fact of how many exclusion criteria exist. To show up in this histogram an event must contain exactly one hit in the FE-I4 plane and each of the four DUT planes since otherwise correlation is not possible. This immensely limits the amount of useable events down from the 50k taken to the used ∼5k. As can be seen in fig. 4.20 the number of entries becomes a lot larger if only one DUT is required to have a hit.



(a) narrow side (b) wide side

Figure 4.20: Comparison of the FW n and p-side resolutions with and without the usage of the FE-I4 sensor.

4.4 Conclusion

Two beam tests were performed to study the resolution of the sensors to be used for the Belle II experiment. Due to the limited time that was available to take measurement data during the first beam test, no big insights could be gained from

these results which resulted in the scheduling of the second beam test. The second beam test was aimed at investigating the change in resolution of the sensor for the varying pitch on the p-side of the FW sensors as well as comparing the change in SNR for the n-side of these sensors, since the strip length decreases over them which should be visible. The fact that data from the first beam test already existed made preparations for the second beam test a lot easier and also allowed for immediate analysis of the data during the beam test since most of the work had already been done for the data of the first beam test.

The data taken during the second beam test provides the information lacking from the first beam test and allowed for a detailed analysis of the sensors and their resolution as well as further improvements of the measurements by means of the FE-I4 sensor.

The insights gained are valuable as the results show that the resolution of the modules is similar to the expected one for a not-cooled sensor. The specifications as defined in the Belle II TDR[2] are not fulfilled most probably because the sensors were not cooled and therefore the SNR was not good enough. Tracking the particle with a resolution of 13 µm on the p-side(r-ϕ) and 40 µm on the n-side(z-axis) for each plane could still be enough for the precise vertex reconstruction required for the experiment since the fit needs to be performed through all six planes of the detector which further improves the resolution.

Furthermore the investigations of the influence of the intermediate strip on binary sensor readout are also quite significant as they prove that binary readout can be improved by a significant margin (up to 50% for large pitches) by using intermediate strips due to the reduced max size of cluster width one events.

As a point of comparison one can look at [12] which investigated single-sided strip detectors with 64 mm strip length and 50 µm pitch. Comparability is given for one of the three investigated zones in this work, since it also had one intermediate strip. The best resolution achieved there was around 6 µm which is a lot better than the results achieved with these wedge sensors. Multiple differences between the works performed make it difficult to pinpoint one exact reason for the results. Aside from the fact that the strip length for the wedge sensors is about twice as much (122 mm) the wedge sensors are also double-sided and the analysis software used was a completely different one. One additional work performed by another student in parallel using the same analysis software but a different telescope setup lead to almost identical results regarding the sensor resolution lending additional credibility to the results presented in this thesis.

Acknowledgements

I would like to express my gratitude to all the people that helped me along the way of performing my work for this master's thesis.

First and foremost I want to thank *Thomas Bergauer* for his help with all the questions and requests I bombarded him with as well as all the information he provided during long conversations where he did his best to help me understand complex connections between topics. Additionally I want to thank him for always taking the time to proofread all the presentations I held during my time and all the time he invested into this thesis becoming the document it is now.

Special thanks also goes out to my supervisor *Christoph Schwanda* who aside from thesis relevant topics also took the time to talk about some less thesis relevant but to me very interesting topics.

Moreover I would like to express my gratitude to the electronics group especially to *Markus Friedl* who was always willing to explain sensor electronics to me as well as *Christian Irmler* who gave me a lot of insight regarding the database and the construction of the detector and was a valuable source of knowledge in many regards.

I also want to thank *Hao Yin* for all the talks we had about programming and all the things I learned about low level programming that I never even thought about before.

Special thanks also go to *Richard Thalmeier aka. Fachi* whose vast knowledge of electronics and computer hardware kept astounding me and who was always willing to talk to me when I required a break from programming and just walked over to his desk.

Furthermore I want to thank *Florian Buchsteiner* and *Lukas Bulla* for making the two beam tests a lot more enjoyable than ten full working days could have been.

My thanks also goes out to *Karen Pommer* for proofreading my thesis and thoroughly correcting my english down to the last comma.

My biggest gratitude goes out to my parents for their moral, social and financial support during my studies without which I would surely never have managed it this far.

Last but not least I want to thank my girlfriend *Lisa Krainz* for all her support especially for dealing with me after long working days and for always listening to me when I was rambling about what occupied my mind.

Appendix A - Source code

AddHitsDriver.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''
Created on Feb 17, 2015

@author: <a href="mailto:b.wuerkner@gmail.com">Benedikt Würkner</a>
'''

from pyLCIO.drivers.Driver import Driver
from pyLCIO import IMPL, EVENT, UTIL
from time import time
import ctypes

class AddHitsDriver ( Driver ):
    ''' Driver to add Hits to an Event (if applicable)
        Expects a dictionary containing a dictionary with a touple of x,y and z coordinates
        Example: allEvents = {"DUT1":{"1":(1.2,1.1,1.3),"7":(1.1,1.2,1.3)},"DUT2":{"1":(1.2,1.1,1.3)
            ,"4":(1.1,1.2,1.3)}}
    '''

    def __init__( self , allEvents,configDict):
        ''' Constructor '''
        Driver.__init__( self )
        self.allEvents = allEvents
        self.currentCollectionName = "local_hit"
        self.targetCollectionName = "hit_with_DUT_hits"
        self.configDict = configDict["DUTs"]

    def processEvent( self, event ):
        ''' Method called by the event loop for every event '''
        eventNumber = event.getEventNumber()
        trackerHits = IMPL.LCCollectionVec( EVENT.LCIO.TRACKERHIT)
        encodingString = event.getCollection(self.currentCollectionName).getParameters().getStringVal(EVENT.
            LCIO.CellIDEncoding)
        trackerHits.getParameters().setValue(EVENT.LCIO.CellIDEncoding,encodingString)
        idEncoder_DUT = UTIL.CellIDEncoder( IMPL.TrackerHitImpl )( encodingString, trackerHits) #<----
            requires the most amount of time
        idEncoder_DUT.reset()
        idEncoder_DUT['properties'] = 0
        for iHit in range(0,event.getCollection(self.currentCollectionName).getNumberOfElements()):
            trackerHits.addElement(IMPL.TrackerHitImpl(event.getCollection(self.currentCollectionName).
                getElementAt(iHit)))
        for DUT in self.allEvents.keys():
            if eventNumber in self.allEvents[DUT]:
                idEncoder_DUT['sensorID'] = self.configDict[DUT]["sensorID"]
#               startTime = time()
#               print "adding additional Hits:", eventNumber, time()-startTime
                newHit = IMPL.TrackerHitImpl()
                idEncoder_DUT.setCellID(newHit)
                newHit.setType(EVENT.LCIO.THBIT_BARREL)
                hitPos = [self.allEvents[DUT][eventNumber][0],self.allEvents[DUT][eventNumber][1],self.
                    allEvents[DUT][eventNumber][2]]
                pos = (ctypes.c_double * len(hitPos))(*hitPos)
                newHit.setPosition(pos)
                newHit.setQuality(1)
                trackerHits.addElement(newHit)
        event.addCollection(trackerHits,self.targetCollectionName)
        startTime = time()
```

GenericTelescopeConverter.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''
Created on January 14th 2015

Tool to convert a file with tab separated values containing the event number and the x,y coordinates of a hit
      in mm into LCIO EUTelescope format.

@author: <a href="mailto:b.wuerkner@gmail.com">Benedikt Würkner</a>
'''

#from pyLCIO import  UTIL
#from pyLCIO import EVENT,IMPL,IOIMPL, IO
from pyLCIO.io.EventLoop import EventLoop
from pyLCIO.drivers.EventMarkerDriver import EventMarkerDriver
from pyLCIO.drivers.AddHitsDriver import AddHitsDriver
from pyLCIO.drivers.WriteLcioDriver import WriteLcioDriver
from pyLCIO import IMPL
#from pyLCIO import IMPL, IOIMPL, EVENT, UTIL, IO
import sys
import ctypes

inputFileFormatConfig = {"xCol":3,"yCol":4,"zCol":5}

def addEventsToFile(allEvents,configDict):

    #Create filename from run number

    dataFileName = "run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker_backup.slcio"
    outputFileName = "run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker.slcio"



    # Create the event loop
    eventLoop = EventLoop()

    # Set the input file. The actual reader is determined from the file ending (stdhep or slcio)
    eventLoop.addFile( configDict["inputDataDir"]+dataFileName )

    # Add a driver to print the progress
    markerDriver = EventMarkerDriver()
    markerDriver.setInterval( 1000 )
    markerDriver.setShowRunNumber( False )
    eventLoop.add( markerDriver )

    # Add the driver that adds my hits to the events
    addHitsDriver = AddHitsDriver(allEvents,configDict)
    eventLoop.add( addHitsDriver )

    #Create Run Header
    run = IMPL.LCRunHeaderImpl()
    run.setRunNumber( configDict["runNumber"] )
    run.setDetectorName( "EUTelescope" )
    # Add the driver that saves the new collection to a file
    writeLcioDriver = WriteLcioDriver()
    writeLcioDriver.setOutputFileName(outputFileName)
    writeLcioDriver.setRunHeader(run)
    eventLoop.add( writeLcioDriver )

    # Skip some events if desired
    eventLoop.skipEvents( 0 )

    # Execute the event loop
    eventLoop.loop( 59000 )
    eventLoop.printStatistics()

    return

################################################################################################
def usage():
    print 'Tool to convert a file with tab separated values containing the event number and the x,y and z
         coordinates of a hit in mm into LCIO EUTelescope format.'
    print 'Usage:\n python %s <inputTextfileDir> <inputEUTelDatafileDir> <runNumber>' % ( sys.argv[0] )

################################################################################################
if __name__ == '__main__':
    if len( sys.argv ) < 4:
        usage()
        sys.exit( 1 )

    allEvents   = dict()
    inputDir     = sys.argv[1]
    inputDataDir = sys.argv[2]
    runNumber    = int(sys.argv[3])
    offset     = int(sys.argv[4])

    configDict = {"DUTs":{}}
        #Example: Just add rows for each of the correlation files that can be found in the folder passed as
             an argument.
    #configDict["DUTs"]["fileName"] = {"runNumber":1000,"sensorID":10}
    configDict["DUTs"]["fwbw_run005_WedgeOld_correlated.txt"] = {"sensorID":10}
    configDict["DUTs"]["fwbw_run005_FW_correlated.txt"] = {"sensorID":11}
    configDict["DUTs"]["fwbw_run005_BW_correlated.txt"] = {"sensorID":12}
    configDict["inputDataDir"] = inputDataDir
    configDict["runNumber"] = runNumber
    # Number of the col in the input file corresponding to the respective value
```

```
    for fileName in configDict["DUTs"].keys():
        eventsDUT = dict()
        with open(inputDir+fileName) as inputTextFile:
            for line in inputTextFile:
                temp = line.split()
                #only store required values based on the file format defined at the top
                #x,y,z
                eventsDUT[int(temp[0])] = (float(temp[inputFileFormatConfig["xCol"]]),float(temp[
                    inputFileFormatConfig["yCol"]]),0) #New version with z-coordinate set to 0 (local frame
                    now)
            allEvents[fileName] = eventsDUT
            print fileName,"lowest event Number is:",sorted(eventsDUT.keys())[0]
        print "loaded",fileName,"with",len(allEvents[fileName]),"events"

    addEventsToFile(allEvents,configDict)
```

correlateNandP.py

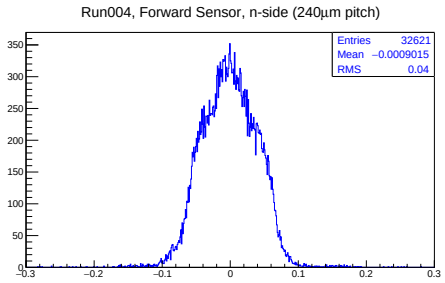
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''
Created on March 24th 2015

Program to create x and y coordinates in millimiters out of double sided strip sensors (DSSDs) which have
     strips angled 90deg to each other.

@author: <a href="mailto:b.wuerkner@gmail.com">Benedikt Würkner</a>
'''
import sys, shutil
import csv
import os.path
import numpy as np
import ROOT as R
import resource
import time

'''
It requires to define the amount of strips on the n and p-side as well as the overall geometry of the sensor
     while allowing trapezoidal shapes and automatically calculating the pitch from the given data.
It is only capable of calculating hits when there was only one hit per event number because there is no
     possibility to get rid of the phantom hits.

To define the geometry some parameters are required as well.
'''
longHeight  = 57.590    #height of the sensor on the long (left) side in mm
shortHeight = 38.420    #height of the sensor on the short (right) side in mm
width       = 122.76    #width of the sensor in mm
nStrips     = 512       #Number of strips on the n-side
pStrips     = 768       #Number of strips on the p-side
cnStrips    = nStrips-1 #count of strips on n-side
cpStrips    = pStrips-1 #count of strips on p-side

'''

We need to define some global parameters that are used throughout the program and depend on the input file
     format or on the data.
'''
originalOffset = -1 #Offset to be subtracted from the event number given by the TLU in relation to the event
     number stored in the lcio event
offsetCorrectionFactor = 1 #Second offset for after 32767 events
inputDataDir = "../NewReconstruction/output/lcio/"
useEtaCorrection = True
#the Following are only avaliable if 'useEtaCorrection' is set to False
useOnlyCW2Plus = True
useOnlyCW1 = False
'''
Furthermore we need to define the files that shall be used including if x or y coordinate need to be flipped
     because of the way the sensors were geometrically positioned.

This new version directly accepts root files from TuxOA as input thereby eliminating the intermediate text
     files used previously.
This leads to the requirement of knowing the name of the Module number corresponding to the module e.g. 000 =
      WedgeOld_p-side

parameters reqired:
    x-coordinate-file,
    y-coordinate-file,
    DUTName,
    flipX,
    flipY,
    longHeight,
    shortHeight
Make educated guess on the ResultTree name by listing all the names and then taking the one containing "
     Result"?
Maybe enable general prefix parameter for files.
TuxOA standard output file name: DataFileName_ModuleNumber e.g. fwbw_run003_000.root if the datafile is
     called fwbw_run003.dat

Coordinate system definition of the telescope is standard right handed with positive z along the beam axis in
      beam direction.
Following from this the positive y-axis points upwards and the positive x-axis points to the left (looking in
      beam direction)
Furthermore following from the fact that the n-strips go from top to bottom they are capable of giving a
     position varying left to right and therefore x-coordinate
```

```
The flip parameter needs to be considered of how the apv numbering is equivalent to the positive axis.
If the highest strip number is at the same position as the highest positive value then it does not need to be
      flipped. (TODO: Check that i didn't mess that up!)

'''
runNumber = 1008   #EuDAQ run number, used to determine the file name of the hit file. NEEDS to be adapted
     along with the runName
runNumber = int(sys.argv[2])
runName   = "run003" #Makes it easier to change the run number
runName   = sys.argv[3]
etaSteps  = 100 #amount of bins used in the eta histogram. Increases calculation time by a lot for high
     values. Recommended: 100

files = [
        {
            'fileX':'fwbw_'+runName+'_006.root',
            'fileY':'fwbw_'+runName+'_002.root',
            'oFileName':'fwbw_'+runName+'_BW_correlated.txt', #unused parameter
            'DUTName':'BW',
            'flipX':False,
            'flipY':False,
            'longHeight':longHeight,
            'shortHeight':longHeight
        },
        {
            'fileX':'fwbw_'+runName+'_005.root',
            'fileY':'fwbw_'+runName+'_001.root',
            'oFileName':'fwbw_'+runName+'_FW_correlated.txt', #unused parameter
            'DUTName':'FW',
            'flipX':True,
            'flipY':True,
            'longHeight':longHeight,
            'shortHeight':shortHeight
        },
        {
            'fileX':'fwbw_'+runName+'_004.root',
            'fileY':'fwbw_'+runName+'_000.root',
            'oFileName':'fwbw_'+runName+'_WedgeOld_correlated.txt', #unused parameter
            'DUTName':'WedgeOld',
            'flipX':False,
            'flipY':True,
            'longHeight':longHeight,
            'shortHeight':shortHeight
        },
        ]



#Code starts here, only mess with it if you know what you're doing!


#Checking parameters
if len( sys.argv ) < 2:
    print "requires at least one path as parameter (for input and output file location). If two parameters
         are given the first is assumed to be the input folder and the second the output folder"
    sys.exit( 1 )

if len(sys.argv) == 2:
    inFileDir = sys.argv[1]
    outFileDir = sys.argv[1]
else:
    inFileDir = sys.argv[1]
    outFileDir = sys.argv[1]

#To make sure the program doesn't abort later on check if all input files and the output directory exist
for settings in files:
    if not os.path.isfile(inFileDir+settings['fileX']):
        print inFileDir+settings['fileX'],"doesn't exist, check your settings"
        sys.exit()
    if not os.path.isfile(inFileDir+settings['fileY']):
        print inFileDir+settings['fileY'],"doesn't exist, check your settings"
        sys.exit()
    if not os.path.isdir(outFileDir):
        print "outputdir",outFileDir,"doesn't exist, check your parameters"
        sys.exit()


startTime = time.time()
def runtime(description="Something"):
    global startTime
    print description,"{:1.2f}".format(time.time()-startTime)+"s"
    startTime = time.time()

allEvents = dict()

print "Starting to process run",runName,"with EuTel run number",runNumber

for settings in files:
    print "Processing DUT:",settings["DUTName"]
    xFile = R.TFile(inFileDir+settings['fileX'])
    xTree = xFile.Get("ResultTree_"+settings['fileX'].split(".")[0].split("_")[-1])
    xDict = dict()
    xEtaValues = list()

    yFile = R.TFile(inFileDir+settings['fileY'])
    yTree = yFile.Get("ResultTree_"+settings['fileY'].split(".")[0].split("_")[-1])
    yDict = dict()
    yEtaValues = list()
```

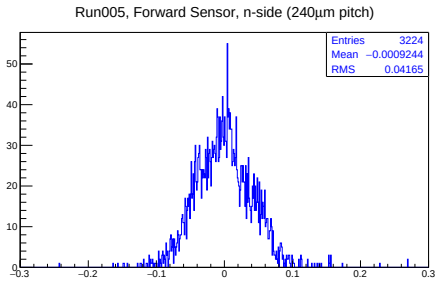
```
    for event in xTree:
        #if any of the values contains more than one value then the event contains multiple hits and cannot
             be used because of ghost hits
        if len(event.Eta)>1:
            continue
        xDict[event.TLUEventNumber] = {"eta":event.Eta[0],"clusterWidth":event.ClusterWidth[0],"firstStrip":
             event.FirstStrip[0]}
        xEtaValues.append(event.Eta[0])
    runtime("Calculating eta values x:")

    for event in yTree:
        #if any of the values contains more than one value then the event contains multiple hits and cannot
             be used because of ghost hits
        if len(event.Eta)>1:
            continue
        yDict[event.TLUEventNumber] = {"eta":event.Eta[0],"clusterWidth":event.ClusterWidth[0],"firstStrip":
             event.FirstStrip[0]}
        yEtaValues.append(event.Eta[0])
    runtime("Calculating eta values y:")

    xEtaDistr = np.histogram(np.array(xEtaValues),bins=etaSteps,range=[0,1])
    yEtaDistr = np.histogram(np.array(yEtaValues),bins=etaSteps,range=[0,1])
    #Generate histogram of files and store them in the corresponding np.array
    F1 = 1./np.sum(xEtaDistr[0])
    F2 = 1./np.sum(yEtaDistr[0])

    def x(nStrip):
        return  width*(0.5-float(nStrip)/(nStrips-1))

    a = (settings['longHeight']-settings['shortHeight'])/(2.0*(cnStrips))
    b = -(settings['shortHeight'])/(cpStrips)
    c = (settings['shortHeight']-settings['longHeight'])/((cnStrips)*(cpStrips))
    d = settings['shortHeight']*0.5
    def y(nStrip,pStrip):
        return nStrip*a + pStrip*b + nStrip*pStrip*c + d

    eventNos = set(yDict.keys()).intersection(set(xDict.keys()))#merges two lists only keeping elements in
         both thereby creating a set that only contains only events where both sides had one hit
    allEvents[settings["DUTName"]] = {}
    print "starting to loop events, this may take a while"
    sumOfCW1 = 0
    sumOfCW2 = 0
    for eventNo in eventNos:
#       Trial to change offset. Leads to weird side effects and not many tracks being recognized anymore. No
     Idea why....
        offset = originalOffset+int(eventNo/32768)*offsetCorrectionFactor
#       print offset
#       print eventNo, offset, originalOffset
        #If no add it to the dictionary and increase one of the files (irrelevant which)
        #cog = (leftStripPos-RightStripPos)*eta+RightStripPos
        sum1 = 0.
        for i in range(0,etaSteps):
            if xEtaDistr[1][i] > float(xDict[eventNo]["eta"]): #Abort loop if current eta value is larger
                 than loop position eta value
                break
            sum1 += xEtaDistr[0][i]
        FetaX = F1*sum1

        sum2 = 0.
        for i in range(0,etaSteps):
            if yEtaDistr[1][i] > float(yDict[eventNo]["eta"]): #Abort loop if current eta value is larger
                 than loop position eta value
                break
            sum2 += yEtaDistr[0][i]
        FetaY = F2*sum2
        #in this case (firstStrip-firstStrip+ClusterWidth)*eta+firstStrip+ClusterWidth = firstStrip+
             ClusterWidth+ClusterWidth*eta
#                   cog1 = int(file1[clusterWidth])*(1-float(file1[eta]))+int(file1[firstStrip])
#                   cog2 = int(file2[clusterWidth])*(1-float(file2[eta]))+int(file2[firstStrip])

        cog1 = int(xDict[eventNo]["firstStrip"])+FetaX*xDict[eventNo]["clusterWidth"]
        cog2 = int(yDict[eventNo]["firstStrip"])+FetaY*yDict[eventNo]["clusterWidth"]
#       print xDict[eventNo]["clusterWidth"],x(cog1)-x(xDict[eventNo]["seedStrip"]), xDict[eventNo]["eta"]
#       if useEtaCorrection and yDict[eventNo]["clusterWidth"] != 0 and settings["DUTName"] == "BW":
#           continue
        if not useEtaCorrection: #strip numbers are then rounded leading to the strip with the highest charge
             being the strip used
            if useOnlyCW1:
                if int(xDict[eventNo]["clusterWidth"]) == 0:
                    cog1 = round(cog1)
                    cog2 = round(cog2)
                else:
                    continue
                sumOfCW1 += 1
            elif useOnlyCW2Plus:
                if int(xDict[eventNo]["clusterWidth"]) != 0:# or int(yDict[eventNo]["clusterWidth"]) != 0:
                            cog1 = round(cog1)
                                    cog2 = round(cog2)
                else:
                    continue
                sumOfCW2 += 1
            else:
                                cog1 = round(cog1)
                                cog2 = round(cog2)
                sumOfCW1 += 1
                sumOfCW2 += 1
```

```
        if not settings['flipX'] and not settings['flipY']:
            xStrip = cog1
            yStrip = cog2
        elif settings['flipX'] and settings['flipY']:
            xStrip = cnStrips-cog1
            yStrip = cpStrips-cog2
        elif not settings['flipX'] and settings['flipY']:
            xStrip = cog1
            yStrip = cpStrips-cog2
        elif settings['flipX'] and not settings['flipY']:
            xStrip = cnStrips-cog1
            yStrip = cog2
        allEvents[settings["DUTName"]][(eventNo+offset)] = (float(x(xStrip)),float(y(xStrip,yStrip)),0)

#       print eventNo,int(xDict[eventNo]["firstStrip"]),cog1,int(yDict[eventNo]["firstStrip"]),cog2,x(xStrip)
    ,y(xStrip,yStrip),xDict[eventNo]["clusterWidth"]
#       coordinates[eventNo] = (cog1,cog2)
    print "Cluster Width 1 events:",sumOfCW1
    print "Cluster Width 2 events:",sumOfCW2
    runtime("Looping events took: ")
    print "Memory usage: ","{:1.2f}".format(resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.),"MB"
    print ""
#for DUT in allEvents:
#   for event in allEvents[DUT]:
#       print event,allEvents[DUT][event]

    #output data
    # with open(outFileDir+settings['oFileName'], 'wb') as csvfile:
    #   w = csv.writer(csvfile, delimiter='\t', quoting=csv.QUOTE_NONE)
    #   for eventNo in coordinates.keys():
    #       nStrip, pStrip = coordinates[eventNo]
    #       if not settings['flipX'] and not settings['flipY']:
    #           w.writerow([(eventNo-offset), nStrip, pStrip, x(nStrip), y(nStrip, pStrip)])
    #       elif settings['flipX'] and settings['flipY']:
    #           w.writerow([(eventNo-offset), nStrip, pStrip, x(cnStrips-nStrip), y(cnStrips-nStrip, cpStrips
        -pStrip)])
    #       elif not settings['flipX'] and settings['flipY']:
    #           w.writerow([ (eventNo-offset), nStrip, pStrip, x(nStrip), y(nStrip, cpStrips-pStrip)])
    #       elif settings['flipX'] and not settings['flipY']:
    #           w.writerow([ (eventNo-offset), nStrip, pStrip, x(cnStrips-nStrip), y(cnStrips-nStrip, pStrip)
        ])
    # #print nStrip,pStrip,nStrips,pStrips,cnStrips-nStrip,cpStrips-pStrip
    # runtime("writing "+settings['oFileName']+" to disk")


import GenericTelescopeConverter as GTC

configDict = {"DUTs":{}}
configDict["DUTs"]["WedgeOld"] = {"sensorID":10}
configDict["DUTs"]["FW"] = {"sensorID":11}
configDict["DUTs"]["BW"] = {"sensorID":12}
configDict["inputDataDir"] = inputDataDir
configDict["runNumber"] = runNumber

#Check if backup hitfile exists and if not create one because the GenericTelescopeConverter expects there to
     be one
if not os.path.isfile(inputDataDir+"run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker_backup.slcio"):
    if os.path.isfile(inputDataDir+"run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker.slcio"):
        print "Creating backup of hitfile since it doesn't exist already"
        shutil.copyfile(inputDataDir+"run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker.slcio",
            inputDataDir+"run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker_backup.slcio")
    else:
        print "No hitfile found, have you not run the hitmaker yet?"
        print "Or is the path",inputDataDir,"wrong?"
        sys.exit()

GTC.addEventsToFile(allEvents,configDict)
#Move file to directory where it is expected, i.e. where the source hitfile was taken from.
print "Moving created file to correct location at ",inputDataDir+"run"+str(configDict["runNumber"]).zfill(6)+
    "-hitmaker"".slcio"
shutil.move("run"+str(configDict["runNumber"]).zfill(6)+"-hitmaker"".slcio",inputDataDir+"run"+str(configDict
    ["runNumber"]).zfill(6)+"-hitmaker.slcio")
print ""
print "Finished without errors"
print "Memory usage: ","{:1.2f}".format(resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.),"MB"
```

Appendix B - Additional plots

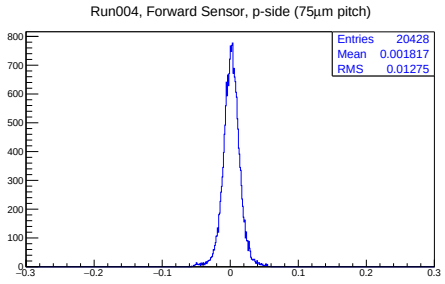


(a) run004

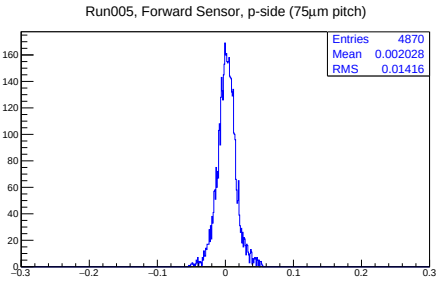


(b) run005

Figure 4.21: Compare resolution of Forward Module between run004 and run005, n-side

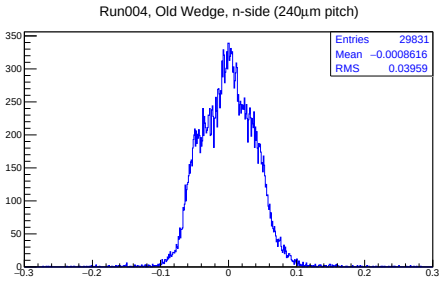


(a) run004

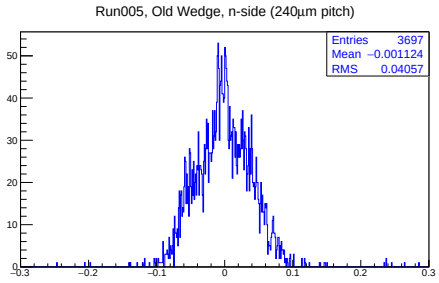


(b) run005

Figure 4.22: Compare resolution of Forward Module between run004 and run005, p-side

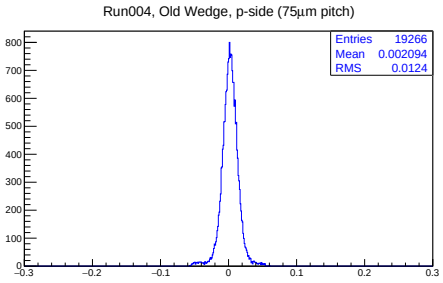


(a) run004

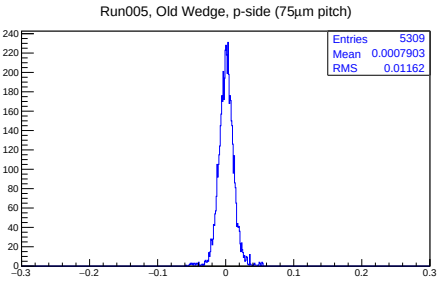


(b) run005

Figure 4.23: Compare resolution of Old Wedge Module between run004 and run005, n-side



(a) run004



(b) run005

Figure 4.24: Compare resolution of Old Wedge Module between run004 and run005, p-side

Bibliography

[1] T. Aushev, W. Bartel, A. Bondar et al. *Physics at Super B Factory*. Technical report, KEK [2009].

[2] Z. Doležal and S. Uno. *Belle II Technical Design Report*. Technical report, High Energy Accelerator Research Organization [2010].

[3] T.Bergauer, P.Doljeschi, A.Frankenberger et al. *Recent Progress in Sensor- and Mechanics-RD for the Belle II Silicon Vertex Detector*. Technical report, Institute of High Energy Physics, Nikolsdorfer Gasse 18, A-1050 Vienna, Austria [2012].

[4] M. Friedl, T. Bergauer, I. Gfall et al. *The Silicon Vertex Detector of the Belle II Experiment*. Technical report, HEPHY [2010].

[5] M. Friedl. *The CMS Silicon Strip Tracker and its Electronic Readout*. Ph.D. thesis, Vienna University of Technology [2001].

[6] H. Yin. *Readout software for the Belle silicon vertex detector and test beam data analysis*. Master's thesis, Vienna University of Technology [2015].

[7] B. Leitl. *Measurement and Simulation of the Interstrip Capacitance of Double Sided Silicon Sensors*. Master's thesis, University of Vienna [2014].

[8] J. Behr. *Test Beam Measurements with the EUDET Pixel Telescope*. report, DESY, `http://www.eudet.org/e26/e26/e27/e107291/eudet-report-2010-01.pdf` [2010].

[9] F.-I. Collaboration. *The FE-I4A Integrated Circuit Guide*. Technical report, CERN, `https://espace.cern.ch/atlas-pixel-upgrade-elec/Final%20Design/Reference/FE-I4A_V11.1.pdf` [2011].

[10] V. Blobel. *Millepede II - Linear Least Squares Fits with a Large Number of Parameters*. `http://www.desy.de/~blobel/Mptwo.pdf` [2007].

[11] M. Stoye. *Calibration and Alignment of the CMS Silicon Tracking Detector*. Ph.D. thesis, Universität Hamburg, `http://www-library.desy.de/preparch/desy/thesis/desy-thesis-07-026.pdf` [2007].

[12] W. Kiesenhofer. *Performance studies on Silicon Strip Sensors with 50 mu pitch*. Master's thesis, Vienna University of Technology [2010].