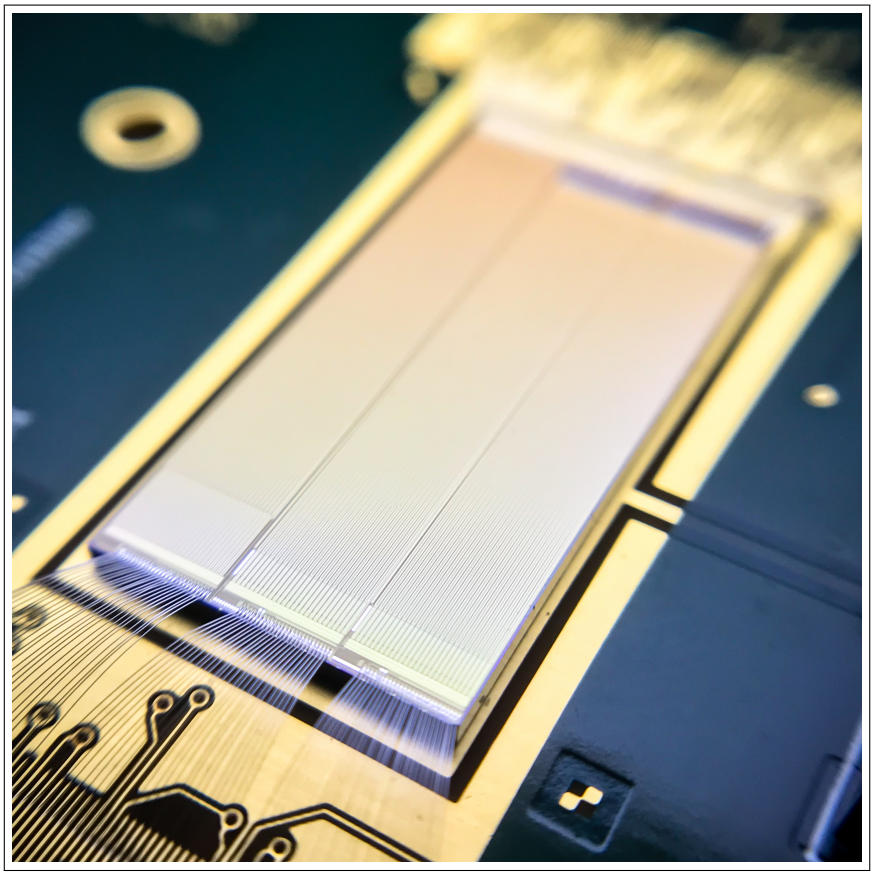
Characterisation of Silicon Pixel Sensors for High-Energy Physics and beyond

Lab Course for ESIPAP 2021



Author: Jens Kröger

Contact: Katharina Dort

katharina.dort@cern.ch

Based on Fortgeschrittenen-Praktikum FP96, Universität Heidelberg

Contents

1 Overview **2**

1.1 Prerequisites . . . 2

1.2 Preparation . . . 2

1.3 Optional: Installation/Setup on Your Personal Computer . . . 4

2 Introduction and Theory **6**

2.1 A Short Introduction to Silicon Pixel Sensors . . . 6

2.2 Sensor Characterisation . . . 9

2.3 Test-Beam Facilities . . . 10

2.4 The Reference Telescope . . . 11

2.5 The Device-Under-Test . . . 11

3 Basics of Test-beam Analysis **13**

3.1 Overview of the Software . . . 13

3.2 The Test-beam Reconstruction Chain . . . 14

4 Instructions **19**

4.1 Getting Familiar with the Tools . . . 19

4.2 Reading in the Raw Data . . . 24

4.3 Clustering . . . 25

4.4 Correlations . . . 26

4.5 Alignment and Tracking . . . 27

4.6 DUT Analysis . . . 32

5 Closing Remarks **34**

5.1 Possible Further Steps . . . 34

1 Overview

This lab course is an introduction to data analysis in the field of silicon pixel detector development for High-Energy Physics (HEP). The objectives are 1) to understand the working principle of silicon pixel detectors and 2) to analyse a set of test-beam data in order to characterise a pixel sensor prototype and investigate its performance.

Of course, actual data taking in a real test-beam environment at CERN or DESY is not possible within this course due to cost and time limitations. Therefore, this lab course is a pure analysis project, but a set of real data recorded at the Super Proton Synchrotron (SPS) at CERN, Switzerland, is provided.

The analysis is carried out with the help of the software packages *ROOT* [**Rootweb**, **root-reference**] and *Corryvreckan* [**Corryweb**, **corryvreckan_manual_v1**], with which you will become familiar throughout the course. Step-by-step, you will build up an entire test-beam data reconstruction and analysis chain, starting with reading in the raw data, performing a quality check on the raw data, which under 'real circumstances' would be done during – or right after – data taking. This is followed by a more in-depth analysis of the performance of a sensor prototype.

1.1 Prerequisites

This lab course requires some basic knowledge and interest in the field of semiconductor physics and electronics. A particular preknowledge in particle physics is not necessary. A list of important keywords you should be able to explain is provided in Section 2.

This course can be carried out without prior knowledge in the area of data analysis or programming.

The analysis and data visualisation is carried out using *ROOT* [**root-reference**] and *Corryvreckan* [**Corryweb**]. These software frameworks are written in C++ but they are ready-to-use and well documented, i.e. the user does not need to go through the source code to understand how it works 'behind the scenes'. A user manual and online help are available and should cover most questions.

1.2 Preparation

In preparation for the lab course, please read Sections 1-3 of this manual. In the introductory discussion with your tutor, expect questions about Sections 1-2, so refresh the relevant theoretical background knowledge.

To run the necessary software packages a virtual machine image has been prepared for you. It contains a compressed version of Ubuntu including *ROOT* and *Corryvreckan* as well as the data to be analysed and templates for the analysis. Note that your PC should have free disk space of $\geq$20 GB and at least 4 GB of RAM. To set it up, follow the instructions below:

- Download and install **VirtualBox 6.1** on your PC: https://www.virtualbox.org/
- Download the virtual machine image:
  https://cern.ch/corryvreckan/VMs/AllpixSquared_Corry_Ubuntu_Minimal.ova.
  Note that this may take a while because of its size of 12–13 GB. Please also consider the video if you have trouble with downloading the machine image.
- Open VirtualBox and click *File* $\rightarrow$ *Import Appliance*.
  Choose the downloaded file `Ubuntu_for_PixelLab.ova` and accept all defaults.

- Once the import is complete, start the virtual machine by double-clicking it. You can now log in as user `student` with the password `vreckan123`.

Troubleshooting:

- **For all Mac users:** If you get the error message "Kernel driver not installed (rc=-1908), please follow the instructions provided here: https://www.howtogeek.com/658047/how-to-fix-virtualboxs-"kernel-driver-not-installed-rc-1908-error/
- If you cannot boot the virtual machine and get an error like "VT-x is disabled in the BIOS for all CPU modes" you may need to adapt your BIOS settings: Reboot your PC, enter the BIOS options and enable "Intel Virtualization Technology" or "AMD Secure Virtual Machine (SVM)" under Advanced CPU Settings or similar.

Some tips:

- Once you are logged in, you may adjust the screen resolution by right-clicking on the desktop and then *Display Settings* → *Resolution*.
- If your personal PC allows you may increase the RAM for the virtual machine by clicking *Settings* → *System* → *Motherboard* and changing the *Base Memory* before booting the virtual machine.

If you like, both of you may install the virtual machine and work together in parallel. Alternatively, one of you can prepare the installation and you can work together by sharing your screen.

With your tutor you will agree on a way to communicate and share your screen, e.g. via Skype [**skype**] or a similar platform.

1.3 Optional: Installation/Setup on Your Personal Computer

The software necessary to perform this lab course (*ROOT* and *Corryvreckan*) can be installed on Linux or MacOS. If you have a personal computer with either of these operating systems, you're welcome to use it for the course as an alternative to the CERN PC.

Please note, that the installation of *ROOT* may be quite time-consuming and you should have it ready before starting the course. It can be downloaded here:

https://root.cern.ch/downloading-root.

We can use the terminal to download all the configuration files and data needed for this lab course. In the terminal, navigate to your home directory (or another directory of your choice) and type:

```
$ git clone https://gitlab.cern.ch/jekroege/fp_pixel_sensor_characterisation.git
```

This downloads all the configuration files and examples from a so-called git repository into the directory `fp_pixel_sensor_characterisation`. In the terminal, navigate into `fp_pixel_sensor_characterisation/scripts/` and execute the following command:

```
$ ./download_data.sh
```

This is a bash script (i.e. a collection of terminal commands), which automatically downloads a compressed archive containing the data and unpacks it. We now have all the files and data needed for this course.

Note: The data needs to be downloaded separately because a git repository (`$ git clone ...`) should only contain text files such as macros, configuration files or code, but no large data files. Git repositories are widely used for versioning of code. The details are not relevant for this course but if you want to know more, talk to your tutor.

When the download has finished, run the other available script to install *Corryvreckan*:

```
$ ./install_corry.sh
```

This might take a few minutes to finish. Please wait until it is complete before proceeding.

2 Introduction and Theory

When students think of data analysis in the context of particle physics, they will first and foremost think of the analysis of "fundamental" physics data such as data acquired at the Large Hadron Collider (LHC) at CERN, in order to examine the fundamental laws of nature or probe the particles of the standard model such as the Z boson – see Heidelberg lab course F91 [**fp91**] – or the Higgs boson [**atlas_higgs**, **cms_higgs**] and beyond. As you can imagine, this data is only available after many years of R&D in detector development and other fields such as accelerator physics.

This course focuses on one very important aspect of detector development: test-beam data analysis, a crucial ingredient to facilitate large-scale experiments as they are performed at the LHC.

Throughout this course, you will become familiar with the analysis and visualisation of test-beam data. You will acquire the following knowledge and skills:

- understanding of the functionality of modern silicon pixel sensors
- basic data analysis and data visualisation skills

2.1 A Short Introduction to Silicon Pixel Sensors

Pixel sensors are solid state particle detectors, which are widely used in particle physics detector systems. Prominent examples from high-energy physics comprise the large experiments at LHC (ATLAS, CMS, ALICE, LHCb). In addition, pixel sensors are developed for smaller experiments like Mu3e [**mu3e**] as well as in medical imaging applications like at the Heidelberg Ion Beam Therapy Centre (HIT) [**hit-gehrke**].

The fundamental working principle of modern pixel sensors is based on the properties of semiconductors, mostly silicon or germanium. Please make yourself familiar with the relevant topics discussed below not only by reading this manual but also in literature.

A very good introduction into the topic of detector physics is the (German) textbook **"Teilchendetektoren - Grundlagen und Anwendungen"** by Kolanoski and Wermes [**Teilchendetektoren**]. An English alternative is **"Pixel Detectors: From Fundamentals to Applications"** by Rossi et al. [**PixelDetectors**].

2.1.1 Semiconductor Physics

Pixel sensors make use of the properties and characteristics of semiconductors. To understand and interpret the results of the analysis during the lab course, it is important to have an understanding of the basic principles of the underlying physics.

Prepare to explain and answer the following concepts and questions:

- What is a semiconductor?
- What is an electron-hole pair?
- What is doping of a semiconductor?
- What happens if a p-doped and an n-doped semiconductor are brought into contact?
- What is a depletion region?
- What happens when a voltage is applied across a diode?
- What is forward and reverse bias?

2.1.2 Interaction of Particles with Matter

As mentioned previously, a basic understanding of the interaction of particles with matter necessary. Please refresh your knowledge on the following concepts and be able to explain them by answering the following questions:

- What is ionisation? What is the difference in gases and semiconductors?
- What does the Bethe-Bloch formula describe?
- What are minimum-ionising particles?

2.1.3 Silicon Pixel Sensors

Silicon pixel sensors (see Figures 1 and 2) are based on pn-junctions operated in reverse bias. When a charged particle traverses through the silicon, electron-hole pairs are created along the trajectory of the particle due to ionisation.

Usually, not the entire silicon is depleted. The electron-hole pairs created outside of the depleted volume of the diode diffuse in a random direction because no electric field is present. They either recombine after a certain time or they diffuse into the depletion region. Inside the depletion zone, the electron-hole pairs are separated by the electric field and drift towards the electrodes of the diode, the so-called collection electrodes. According to the **Shockley Ramo theorem** [**Teilchendetektoren**], these drifting charges induce a signal on the collection electrodes, which can be amplified and thus create a measurable electrical signal.

A pixel sensor consists of a 2-dimensional array of diodes. Each pixel corresponds to one diode with a small electronic signal processing circuit. As illustrated in Figure 3, the induced signal is typically fed into an amplifier to make the signals large enough to be processed further. The amplified signal is then fed into a comparator. If the signal is larger than the (user-configurable) threshold of the comparator, the traversing particle is detected as a hit. In many modern silicon pixel sensors, not only the 2-dimensional pixel address (column and row of the hit) are sent out, but in addition a hit timestamp or time-of-arrival (ToA) and the amount of charge (i.e. the size of the signal) is measured. The concept of the charge measurement as the time-over-threshold (ToT) is illustrated in Figure 18.

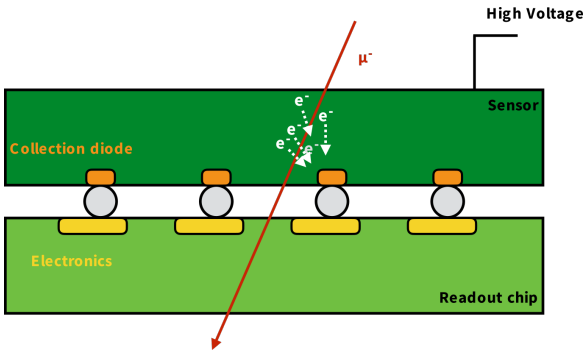


Figure 1: Schematic drawing of a hybrid pixel sensor.

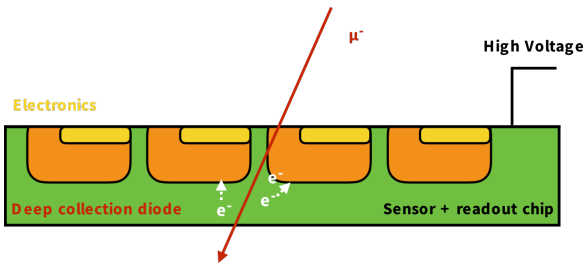


Figure 2: Schematic drawing of a high-voltage monolithic sensor.

Note: The ToT value can be translated into deposited charge by means of an energy calibration. However, this exceeds the scope of this course. Here, it suffices to understand that a large ToT value corresponds to a large induced signal in the pixel and a small ToT value to a smaller signal. In other words, the ToT spectrum corresponds to the spectrum of deposited energy by the ionising particles passing through the sensor.

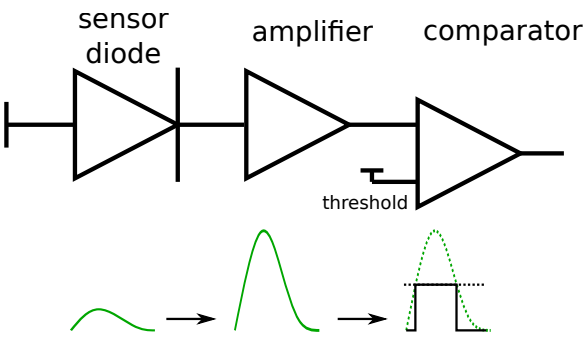


Figure 3: Illustration of the signal amplification and discrimination by the pixel electronics.

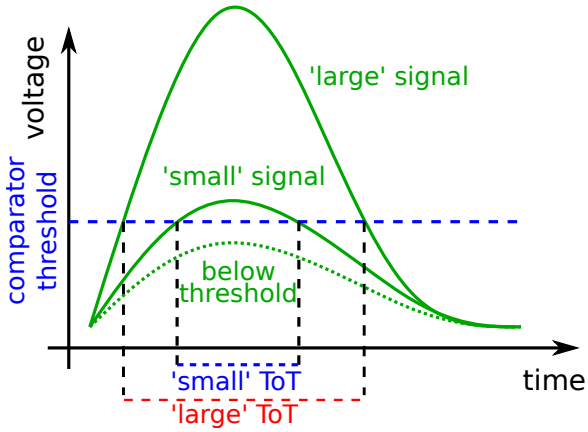


Figure 4: Illustration of the detection threshold and the **time-over-threshold (ToT)** measurement for different pulse heights.

A pixel hit cannot only be caused by a traversing ionising particle. 'Fake hits' can be caused by accidental crossings of the threshold due to (thermal) fluctuations of the amplifier output signal or the threshold or other external factors (e.g. bad solder connections etc.). This effect is called **noise**.

The readout electronics of the sensor contain logic to generate a digital data stream ('ones and zeros'), which is read out by the data acquisition system and stored for offline analysis.

Prepare to answer the following questions:

- How is a signal generated in a pixel sensor?
- What is a ToT measurement and how can it be translated into energy?
- Which spectrum is expected for energy loss of ionising particles in thin layers? What does this have to do with the time-over-threshold?
- What is noise? What can cause it?

2.1.4 Hybrid sensors

The "classical" design of solid state pixel detectors are **hybrid pixel sensors**. As depicted in Figure 1, they consist of two layers. The sensor layer acts as the active detection region. When a traversing ionising particle creates electron-hole pairs, the electrons drift in the electric field created by the applied high voltage towards a collection electrode. Usually, it is connected to a readout chip via small conductive balls, called bump-bonds. The readout chip contains the electronics such as amplifiers, discriminators, etc. and processes the signal induced on the collection electrode to form data packages, which can be sent out to the data acquisition system.

The *Timepix3* chips, which are used in the reference telescope (see section 2.4), are hybrid sensors.

2.1.5 Monolithic pixel sensors

In contrast to hybrid sensors, monolithic pixel sensors consist only of one single layer of silicon, in which the detection volume and the readout electronics are combined (see Figure 2).

In **monolithic active pixel sensors (MAPS)**, a signal amplifier is located in each pixel cell. In **HV-MAPS**, an additional high voltage (HV) in the order of 100 V is applied as a reverse bias. This creates

a strong electric field across the collection diode to allow for a faster charge collection and thus a larger signal and an improved time resolution.

Signal digitisation and data serialisation are usually implemented in a small dedicated area next to the active matrix of the chip, the so-called periphery.

The *ATLASpix_Simple* (see section 2.5), which will be the device-under-test in this lab course, is an example of a HV-MAPS.

Prepare to answer the following questions:

- What is the difference between hybrid and monolithic detectors? Can you think of advantages and disadvantages of the two technologies?

2.2 Sensor Characterisation

The design process of new pixel sensor prototypes is accompanied by an extensive design verification and simulation programme. However, this does not guarantee that they can be taken into operation by plug-and-play as they are often pushing the limits of technology well beyond what has proven to work in the past. Therefore, sensor prototypes need to be characterised and understood in detail once they have been produced.

Usually, the sensor characterisation comprises two complementary parts: laboratory testing and test-beam campaigns. In the laboratory, the sensors are first taken into operation. Typical steps of lab testing and measurements comprise:

- biasing tests (current vs. high-voltage characteristics)
- configuration tests
- optimisation of sensor settings (amplifier, voltages, etc.)
- measurement of the power consumption
- measurement of the noise rate
- energy calibration with radioactive sources or X-ray tubes (photons with well-known energies)

However, not all crucial parameters can be determined in the lab. X-ray tubes or portable radioactive source do not yield the necessary energies to penetrate multiple detector layers. However, this is necessary to achieve a high spatial and time resolution on the reference measurement of the incident particle. For this reason, high-energy particle beams and reference telescopes are used at large-scale test-beam facilities.

Measurements in test-beam campaigns comprise:

- spatial and time resolution
- hit detection efficiency
- detailed studies, such as hit detection efficiency across pixel matrix or within pixels
- comparison and optimisation of different operation parameters
- system integration tests, such as simultaneous operation and readout of many sensors in parallel or multiple subsystems

2.3 Test-Beam Facilities

Test-beam facilities are large-scale scientific user facilities, which allow relativistic particle beams to be used for the testing and characterisation of electronics, biological samples or new materials.

Due to the size and cost of such infrastructure, there are only a few places in Europe, where test-beam campaigns can be conducted. Examples of institutes operating test-beam facilities in Europe are [**tbdb**]:

- DESY in Hamburg, Germany
- CERN near Geneva, Switzerland
- Paul-Scherrer Institute near Zurich, Switzerland

The Super-Proton Synchrotron at CERN

The data provided for this lab course has been recorded at the Super-Proton Synchrotron (SPS) at CERN. As shown in Figure 5, the SPS is not only used as a pre-accelerator to fill the Large Hadron Collider, but it also provides particles for a range of smaller experiments and it serves as a test-beam facility. Figure 6 shows a photograph of the experimental hall at CERN, in which a number of experiments and test-beam setups are located.

The SPS has a circumference of around 7 km and accelerates protons to energies of up 450 GeV [**cern-sps-web**]. Some of these protons are directed onto a target in order to produce secondary particles. The particle beam provided to the test-beam setup shown in the red circle in Figure 6 normally consists of charged pions with a momentum of 120 GeV.

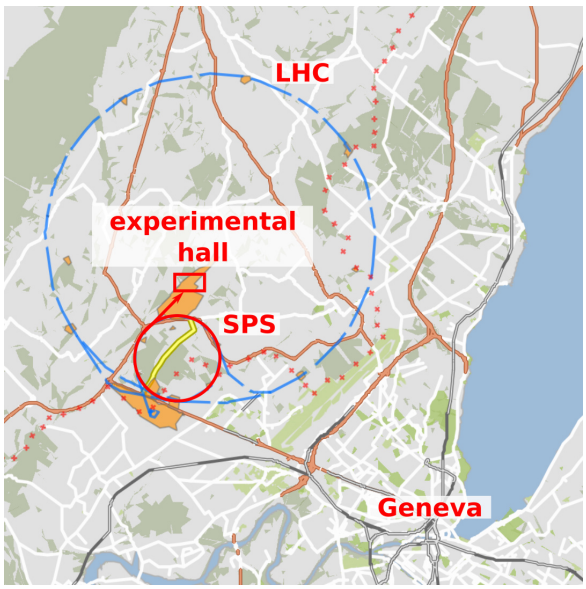


Figure 5: Illustration of the CERN accelerator complex. The red arrow indicates the direction of the particle beam towards the experimental hall (red box) in the North Area at CERN. Screenshot from [**cern-map**] (edited).

Figure 6: Photograph of the experimental hall in the North Area of CERN. The red line indicates the direction of the beam and the red circle marks the position of the reference telescope.

2.4 The Reference Telescope

The data provided for the lab course was taken with the *Timepix3* telescope of the CLICdp collaboration [**yellowreport_detectortech**]. It is located in the experimental hall in the North Area at CERN as indicated in Figures 5 and 6. The telescope consists of seven planes of *Timepix3* readout chips [**timepix3_paper**] bump-bonded to silicon sensors with a thickness of 300 µm. Each Timepix3 consists of a matrix of 256×256 pixels with a pitch of $55 \times 55\,\mu m^2$. In addition to the pixel address (column and row), the *Timepix3* provides a pixel timestamp (time-of-arrival) with a binning of 1.5625 ns (640 MHz) and a time-over-threshold (ToT) measurement with a range of 10-bit (0 - 1023) for each hit.

In short, the information we store for each *Timepix3* hit is:

- pixel address (column, row)
- pixel timestamp: time-of-arrival (ToA)
- pixel charge: time-over-threshold (ToT)

As can be seen in Figures 7 and 8, the telescope planes are rotated by $\pm 9°$ around the *x* and *y* axes (the beam defines the *z* axis) so that particles pass through more than one pixel. This improves the achievable spatial resolution by increasing the amount of charge sharing. You will learn more about charge sharing in Section 3.2.2.

The hits on the planes of the reference telescope will be connected into tracks fitted by a straight line in the analysis. These tracks describe the trajectory of the ionising particles through all sensor planes. The telescope reaches a tracking resolution of $\sim$1–2 µm at the position of the **device-under-test (DUT)** with a timing precision of $\sim$1 ns [**yellowreport_detectortech**].

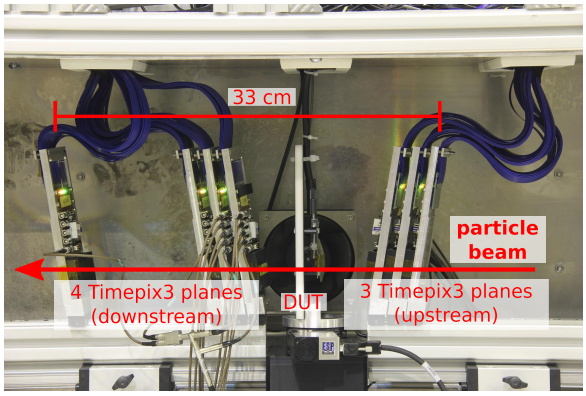


Figure 7: Photograph of the Timepix3 telescope at the SPS, CERN.

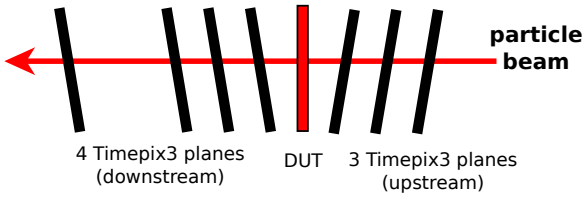


Figure 8: Schematic drawing of the Timepix3 telescope with a device-under-test (DUT) at the SPS, CERN.

2.5 The Device-Under-Test

The device-under-test (DUT) for this lab course is an *ATLASpix_Simple*. It is a High-Voltage Monolithic Active Pixel Sensor, which was designed as a test chip for the ATLAS ITk upgrade [**Prathapan:2019eej**] as well as the CLIC tracking detector [**yellowreport_detectortech**]. After production, the sensors can be thinned down to 50 µm by removing undepleted bulk material from the backside. A high bias voltage of up to $\mathcal{O}(100\,V)$ leads to a large depleted volume with a high electric field resulting in a fast charge collection via drift.

A photograph is shown in Figure 9 and a drawing is shown in Figure 10. The sensor has an active matrix consisting of 25 columns and 400 rows of pixels with a pitch of $130 \times 40\,\mu m^2$. As depicted in

Figures 2 and 3, each pixel consists of a collection diode, which is connected to an amplifier and a comparator. After the comparator, the signal is sent to the digital periphery, where the pixel timestamp (ToA) is added. In addition, the digital periphery houses the readout electronics, which creates the data stream of 'ones and zeros' that is sent out from the chip.

For each hit, the time-of-arrival (ToA) is recorded with a binning of 16 ns. In addition, the signal charge is determined with a time-over-threshold (ToT) measurement with a 6-bit precision (0-63).

In short, the information we get from the *ATLASpix_Simple* is:

- pixel address (column, row)
- pixel timestamp: time-of-arrival (ToA)
- pixel charge: time-over-threshold (ToT)

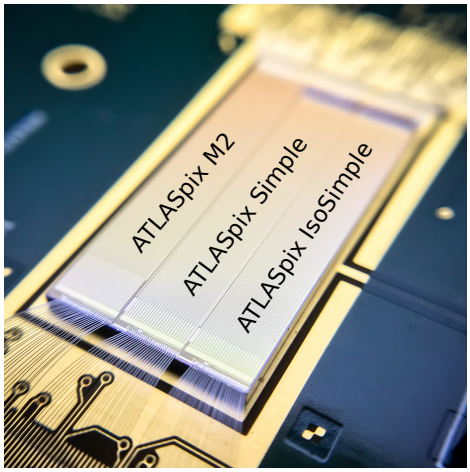


Figure 9: Photograph of an *ATLASpix_Simple* next to similar sensors glued and wire-bonded to a printed circuit board.

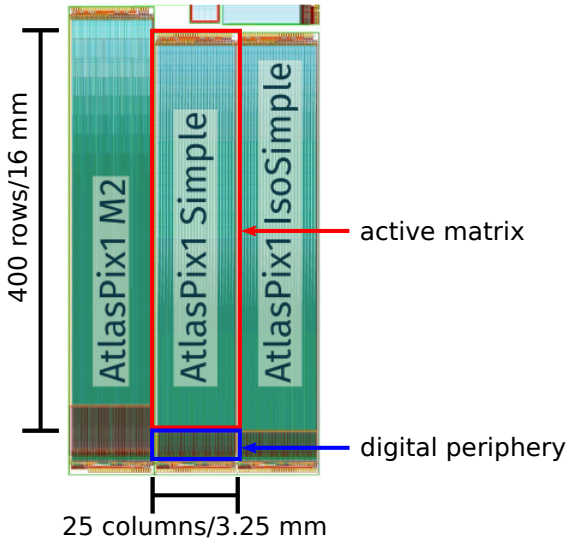


Figure 10: Micrograph of the *ATLASpix* sensors showing the geometric dimensions and the division into active matrix and digital periphery.

3 Basics of Test-beam Analysis

3.1 Overview of the Software

This section provides a brief overview of the software packages *ROOT* and *Corryvreckan*, which will be used during this course.

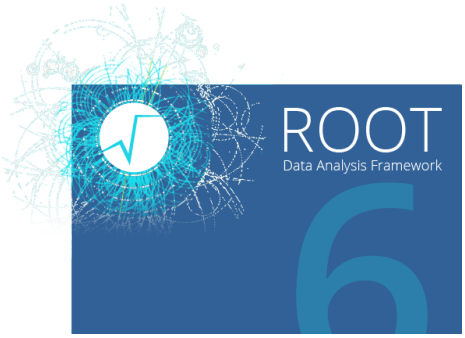


Figure 11: *ROOT* Data Analysis Framework. From [**wikipedia-root**].

Figure 12: *Corryvreckan* – The Maelstrom for Your Test Beam Data. From [**Corryweb**].

3.1.1 *ROOT*

ROOT is a data analysis software package developed at CERN for usage in high-energy particle physics [**root-reference**]. It provides all the functionalities needed for 'big data' and statistical analysis, as well as visualisation ('plotting') and data storage. Consequently, it is not only used in particle physics but also other fields of science and even industry, and many plots that you may have seen in lectures or publications, such as the famous Higgs discovery plots, have been made with *ROOT* (see Figures 13 and 14). More examples can be found here:

https://root.cern.ch/gallery

During this lab course, you will get familiar with *ROOT* and use it for the analysis and visualisation of your results.

3.1.2 *Corryvreckan*

Corryvreckan is a flexible and highly configurable software for the reconstruction and analysis of test-beam (and laboratory) data [**Corryweb**]. It was initially developed in the *CLIC Detector & Physics (CLICdp)* collaboration at CERN [**clic-web**]. The software is capable of performing all steps of a test-beam analysis as described in the following (section 3.2). Each step of the reconstruction is performed by a dedicated module, which can be configured by the user. An illustration of the reconstruction chain for this lab course with the involved modules is shown in Figure 15.

The individuals steps of the analysis are described in the following. You will learn how to use *Corryvreckan* and how to configure the necessary modules in Section 4.

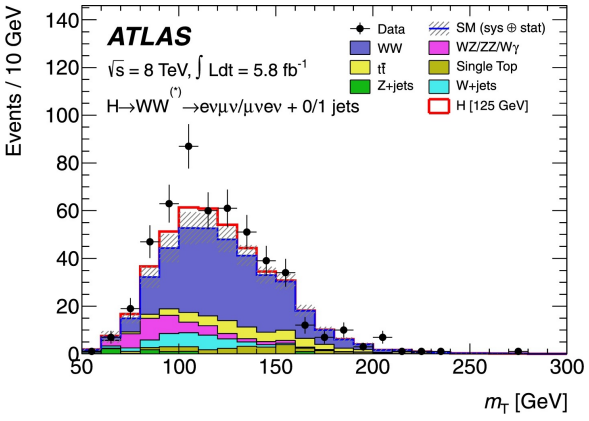


Figure 13: Example plot from the 'ATLAS Higgs discovery paper' [**atlas_higgs**] created with *ROOT*.

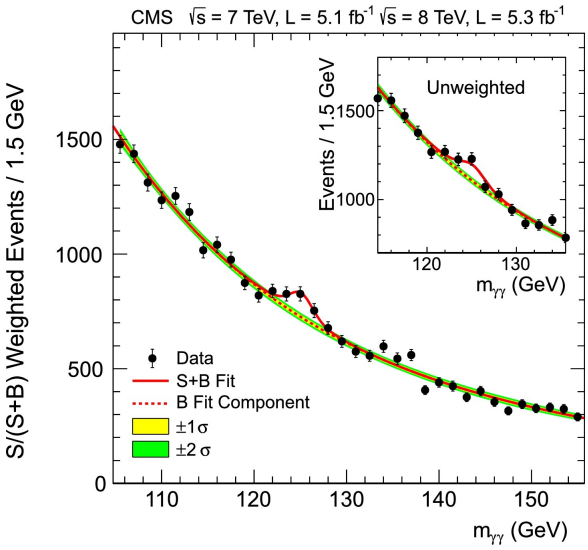


Figure 14: Example plot from the 'CMS Higgs discovery paper' [**cms_higgs**] created with *ROOT*.

3.2 The Test-beam Reconstruction Chain

This section is a short introduction to the fundamentals of test-beam data reconstruction and pixel sensor characterisation. It covers the steps illustrated in the flow chart in Figure 15. As you follow the instructions in Section 4, you will recognize all the steps discussed here and understand them in more detail.

3.2.1 Reading in of Raw Data

The data acquired during a test-beam campaign is usually not stored in tables but in binary files, i.e. in a format which is not human-readable. Therefore, the first step of the reconstructions is reading the raw data from file and interpreting chunks of the data stream as the pixel hit information. In *Corryvreckan*, this is done in the modules called `EventLoader<SomeDetector>`.

The pixel hit information for the *Timepix3* and the *ATLASpix_Simple* comprises:

- pixel address (column and row)

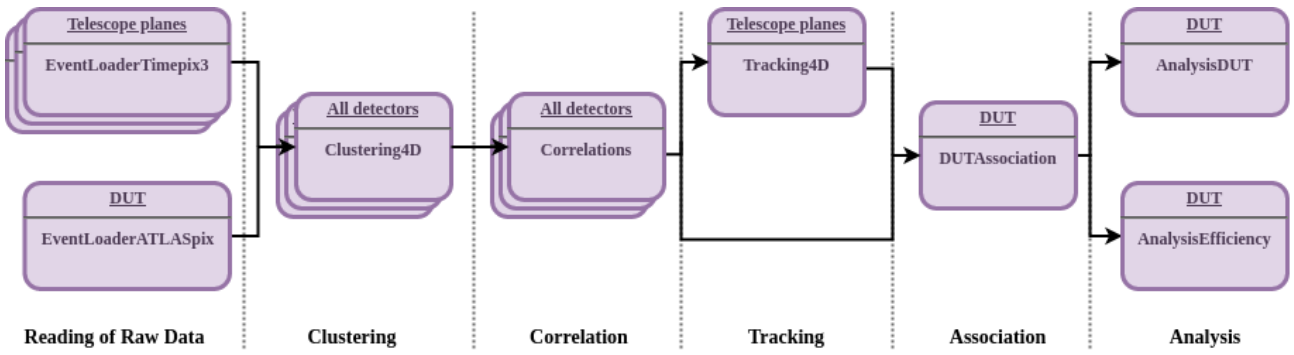


Figure 15: Flow chart of the *Corryvreckan* reconstruction chain for this lab course.

- pixel timestamp: time-of-arrival (ToA)
- pixel charge: time-over-threshold (ToT)

3.2.2 Clustering

Depending on the incident point and angle of a traversing particle, it may create electron-hole pairs in two or more adjacent pixels. In addition, charge can "leak" from one pixel into neighbouring cells by lateral diffusion. This effect is called **charge sharing**.

As a consequence, a clustering algorithm is needed to group such pixels into one cluster. Since both the *ATLASpix_Simple* and the *Timepix3* measure a pixel-by-pixel timestamp (ToA), the clustering can be performed using both spatial and time information. The corresponding module in *Corryvreckan* is called `Clustering4D`.

Charge sharing can be beneficial for the spatial resolution of a detector: If multiple pixels fire and the energy deposition is measured by means of the time-over-threshold (ToT), a ToT-weighted mean of the cluster positions can be calculated.

3.2.3 Tracking

The beam at SPS consists of pions with an energy of around $120\,\mathrm{GeV}$. This energy is sufficiently high to assume that the particles traverse all planes of the telescope in a straight line and scattering effects in the material of the detector planes are negligible.

For the tracking, clusters within a certain spatial and time interval on the planes of the reference telescope are combined into a straight-line track. The DUT plane should be excluded from the tracking to avoid a bias of the analysis.

In order to make use of the spatial and time information provided by the *Timepix3*, the *Corryvreckan* module `Tracking4D` is used.

3.2.4 Alignment

The raw data only contain basic pixel hit information (column, row, timestamp, ToT) for all sensors. Consequently, the reconstruction software is not aware of where exactly the sensor planes are positioned. Therefore, a detector geometry file is needed to describe the position and rotation of all detectors.

In addition, the positions and rotations of each sensor can only be measured by hand to a precision of $\mathcal{O}(1\,\mathrm{mm})$ and they differ from the ideal position with perfect overlap of all sensors as illustrated in Figure 16. However, for a precise tracking, the geometry needs to be known with a higher precison because even a misalignment well below $1\,\mathrm{mm}$ corresponds to an offset of multiple pixels. Therefore, a software alignment needs to be performed to find the exact position of each plane. This way, the intercept of the track with all planes can be determined accurately.

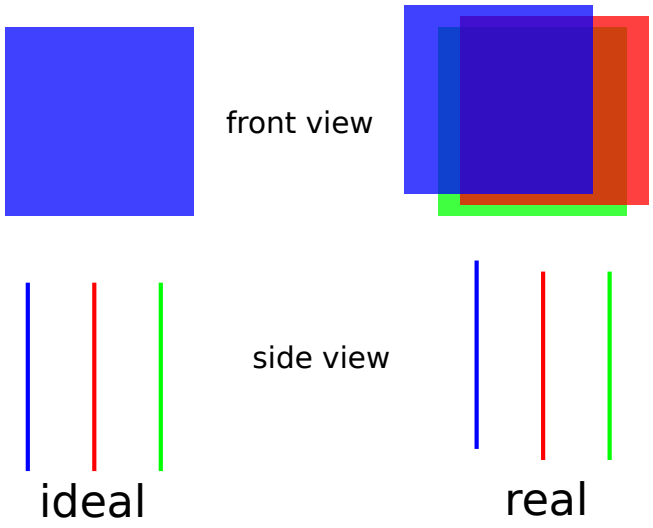


Figure 16: Illustration of the ideal and real (mis-)alignment of the sensor planes (exaggerated). Different telescope planes are indicated in different colours.

3.2.5 Correlations

Spatial correlations in the context of test-beam reconstruction can be defined on pixel or cluster level. In the latter case, the spatial correlations and histogrammed are calculated as:

$$x_{\text{correlation}} = x_{\text{cluster on } \textbf{reference} \text{ detector}} - x_{\text{cluster on } \textbf{this} \text{ detector}} \quad (1)$$

$$y_{\text{correlation}} = y_{\text{cluster on } \textbf{reference} \text{ detector}} - y_{\text{cluster on } \textbf{this} \text{ detector}} \quad (2)$$

for all clusters of a particular detector vs. all pixels on the reference detector. Any detector can be chosen as the reference.

Calculating correlations is a very powerful tool in particular in the early stage of the analysis or even during data taking as a first quality check of the data and a confirmation of its coherence. This is because correlations can be calculated from the raw data **without a working track reconstruction** – which is a bit more work as you will see below.

3.2.6 DUT Association

Since the DUT was excluding from the tracking (see section 3.2.3), its clusters need to be related ('associated') to the reference tracks.

To this end the *Corryvreckan* module `DUTAssociation` is used. It loops over all tracks and associates matching clusters on the DUT within a defined spatial and time interval to the track for further analysis as shown in Figure 17.

3.2.7 DUT Analysis

In the DUT analysis, the clusters on the DUT can be compared with the tracks of the reference telescope.

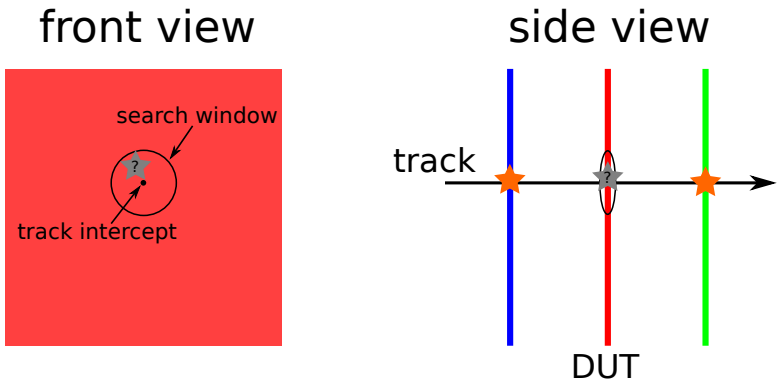


Figure 17: Illustration of the DUT association.

Spatial and Time Resolution

In order to determine the spatial resolution of the DUT, the positional difference between the intercept of a track with the DUT and the associated cluster are calculated and histogrammed:

$$x_{\text{residual}} = x_{\text{track intercept}} - x_{\text{associated cluster}} \tag{3}$$

$$y_{\text{residual}} = y_{\text{track intercept}} - y_{\text{associated cluster}} \tag{4}$$

The root mean square (RMS) of this distribution gives an estimate on the spatial resolution of the DUT.

A simple calculation (see e.g. Appendix E.1 of [**Teilchendetektoren**]) shows, that for pixel sensors with little or no charge sharing (i.e. mostly clusters of size 1) the so-called 'binary resolution' is determined by

$$RMS_{\text{binary, x/y}} = \frac{\text{pixel pitch in x/y}}{\sqrt{12}} \tag{5}$$

Important: The measured resolution deduced from a histogram contains the combined resolutions of both the reference telescope and the DUT. Only if the telescope resolution is much smaller than the DUT resolution, it can be neglected in the calculation:

$$\sigma_{\text{measured}} = \sqrt{\sigma^2_{\text{telescope}} + \sigma^2_{\text{DUT}}} \tag{6}$$

$$\sigma_{\text{measured}} \approx \sigma_{\text{DUT}} \tag{7}$$

In analogy to the spatial resolution, the time resolution can be determined from the width of the time residual between the reference track and the assocated cluster on the DUT:

$$t_{\text{residual}} = t_{\text{track intercept}} - t_{\text{associated cluster}} \tag{8}$$

Unlike the spatial residuals, which are symmetric, the time residual often shows a non-gaussian tail.

As illustrated in Figure 18, a 'small' signal will cross the detection threshold later than a 'large' signal. This signal-height dependent delay is called **timewalk**. If a sensor does not only measure the arrival time of a pixel hit, but also the signal size (time-over-threshold), then the timewalk effect can be corrected for in the analysis. However, this exceeds the complexity of this course.

Both the spatial and the time resolution are determined using the *Corryvreckan* module `AnalysisDUT`.

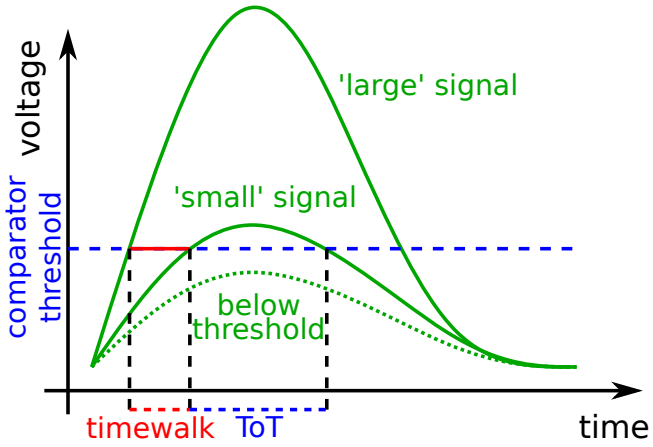


Figure 18: Illustration of the **time-over-threshold (ToT)** and the effect of **timewalk** for pulses of different heights.

3.2.8 Hit Detection Efficiency

The hit detection efficiency ϵ corresponds to the probability of detecting a particle passing through the sensor. It can be determined as follows:

$$\epsilon = \frac{\text{\# tracks } \textbf{with} \text{ an associated cluster}}{\text{\# tracks } \textbf{with + without} \text{ an associated cluster}} \tag{9}$$

If the spatial resolution of the reference telescope is smaller than the size of pixels on the DUT, not only the global efficiency of the sensor can be measured. In addition, the hit detection efficiency can be determined in dependence on where the track intercepts with the sensor within a pixel, i.e. close to the center, edge, or corner of the pixel. This is called the in-pixel efficiency.

The efficiency is determined with the dedicated *Corryvreckan* module `AnalysisEfficiency`.

Table 1: Overview of the most important unix terminal commands. For more, search the internet for *'unix terminal command cheat sheet'* or something more specific.

Command	Meaning	Action
`ls`	list	List the content of the current directory.
`pwd`	print working directory	Show the path to the current directory.
`cd <some_directory>`	change directory	Enter a directory.
`cd ..`		Go to parent directory (one directory up).
`mkdir <some_directory>`	make directory	Create a new directory.
`rm <some_file>`	remove	Remove a file.
`rm -r <some_directory>`	remove recursively	Remove a directory and all its content. Please be very careful! The file or dir will be deleted completely. There is **no trash bin!**

4 Instructions

This section contains the instructions, which should be followed step-by-step during the lab course.

Important: All the steps you will perform here have been discussed in the previous section. You are encouraged to scroll back and read Section 3.2 again before getting started.

4.1 Getting Familiar with the Tools

Before starting with the actual analysis, let's get familiar with the tools we need. Both *ROOT* and *Corryvreckan* are no "Windows-like" software packages with a graphical user interface with clickable buttons (except for the `TBrowser`, see below). Instead of clicking on buttons, all commands need to by typed in via the keyboard.

While this is surely less intuitive and a bit less beginner-friendly – especially if you have only used Windows or MacOS before – it actually turns out to be very powerful as the workflow can be scripted using macros. This is particulary valuable when repeating certain steps many times. Imagine that every command in a macro corresponds to a certain mouse click.

4.1.1 Using the Unix Terminal

Let's start by opening up a Unix terminal. You find it in the list of applications or type `Ctrl+Alt+t`.

All interaction with the terminal happens by using the keyboard. The most important commands are listed in Table 1. Please take your time to play around with them. For example, try to complete the following tasks:

- Check in which directory you currently are.
- Navigate into your `Documents/` folder
- Create a few new directories
- Delete the new directories.

Tip: It is very useful to use the 'arrow-up' key to go back in the history of commands. You can also open multiple terminal windows (`Ctrl+Shift+N`) or tabs (`Ctrl+Shift+T`) in parallel.

The macros and analysis configuration files in this lab course are text files which can be modified with any text editor of your choice. However, also here it's very useful to get familiar with a terminal in-built editor. The two most common editors are **vi** and **emacs**. Again these may not be very intuitive in the beginning, but if you learn how to use them now, it will be very handy in your further career in physics for sure. Search for *'vi cheat sheet'* or *'emacs cheat sheet'* on the internet to get a list of the most important keyboard shortcuts. Alternatively, **gedit** is available as well and its use is self-explanatory. You can start either of the editors from the command line:

```
$ vi <name_of_text_file>
$ emacs <name_of_text_file>
$ gedit <name_of_text_file>
```

During the lab course, you can change between text editors as much as you want. Try them all and use the one you're most comfortable with.

Once more: If you don't know how something works, simply search for it on the internet. There are countless helpful pages out there, which answer all questions you might have. In addition, that's exactly what you will do many times when working on your Bachelor thesis and beyond.

4.1.2 *ROOT* and the TBrowser

As described in section 3.1.1, *ROOT* is a widely used software framework using in particle physics and beyond. In the CIP-Pool, it is already installed and ready to be used. It is controlled via the command line interface and can be started by typing

```
$ root -l
```

You are now in a different terminal-like environment indicated by a leading `root [0]`. Before, it was the `$` symbol. The commands you've learned above are not valid here. Exit from the *ROOT* environment by typing

```
root [0] .q
```

Note: The `-l` flag only avoids a splash screen with the software version to be displayed on start-up. Have a look at the difference by comparing `$ root` and `$ root -l`.

You can append one or multiple *ROOT* files. In your terminal, navigate into the `fp_pixel_sensor_characterisation/output` directory. Then type:

```
$ root -l 01_example.root
```

We can now inspect the histograms in this file in an interactive way using the `TBrowser`. To open it, type:

```
root [0] new TBrowser
```

A new windows will pop up and you can inspect the content of the *ROOT* file by clicking through the navigation panel on the left. To plot a histogram, double-click on its name. A right-click on the different elements of the histogram (axis, data points, etc.) will show many options. Take some time to play around and get familiar with the functionality. For instance, try the following:

- Draw a 1D histogram.
- Zoom into different parts of the x-axis.
  Tip: Mark an axis range with the mouse (hold down the mouse button while selecting). Alternatively, right-click on the axis and click "SetRangeUser" to type in numbers. "Unzoom" to revert.
- Change the line color and thickness of the histogram.
  Tip: Right-click on the histogram and select "DrawPanel"
- Save the histogram as a PDF.
- Draw a 2D histogram.
  Tip: Use the draw option "colz".

Note: The statistics box of a histogram shows the total number of entries in the histogram. In additon, it displays the arithmetic mean ("mean") of the current zoom range and its RMS ("Std Dev"). When you vary the zoom range, you will see how they change.

We will use the `TBrowser` and extensively throughout this lab course because the analysis software *Corryvreckan* saves its results into *ROOT* files. We'll have a look at it next.

Using *ROOT* Macros

The `TBrowser` is very useful to inspect the output of the analysis in a fast way and look into different histograms in detail. However, it is not ideal to create plots in a certain style or perform a particular task repeatedly such as plotting multiple histograms with a defined zoom range on the axes, which would involve a lot of time-consuming and error-prone clicking. For this reason, we can make use of *ROOT* **macros**. These are text files, in which any click we do with the mouse in the `TBrowser` is represented by a written command.

ROOT macros can be written in C/C++ or Python. Since C/C++ is more widely used and more help is available online, we will go ahead using C/C++. However, if you have a strong personal preference, you can also choose to write your macros in Python.

Note: You do not require a deep understanding of C/C++ or Python. The macros in this lab course are rather basic and can also be understood and modified without previous programming experience.

Now let's have a look at our first macro. Open `example_macro.C` with a text editor and try to understand its functionality line-by-line. To execute the macro, simply type in the terminal:

```
$ root -l example_macro.C
```

Take some time to modify parts of the macro or comment out some of the lines by adding two slashes "`//`" at the beginning of a line. Then re-run it and see what changed.

Later you can use this example as a basis for your own *ROOT* macros. Copy the file, rename it and modify it to your needs.

4.1.3 Corryvreckan

As described in section 3.1.2, *Corryvreckan* is a flexible and highly configurable software for the reconstruction and analysis of test-beam data. In the following, you will learn how to configure and use it step-by-step.

Similar to *ROOT*, *Corryvreckan* is controlled via the command line. You can run it by typing

```
$ corry
```

This will print the software version and some help on the usage in the terminal:

```
Corryvreckan v1.0.3
The Maelstrom for Your Test Beam Data

Usage: corry -c <config> [OPTIONS]

Options:
  -c <file>    configuration file to be used
  -l <file>    file to log to besides standard output
  -o <option>  extra configuration option(s) to pass
  -g <option>  extra detector configuration options(s) to pass
  -v <level>   verbosity level, overwriting the global level
  --version    print version information and quit
```

Corryvreckan needs to be provided with a configuration file as an input parameter to know what it is supposed to do. This is referred to as the **main configuration file**.

An example can be found in `fp_pixel_sensor_characterisation`. Open `01_example.conf` with a text editor and inspect it.

As described in section 3.1.2, *Corryvreckan* is built in a modular way, such that each module performs a separate task such as reading in raw data, clustering or tracking. The main configuration file defines the reconstruction chain by listing all modules to be used and contains all relevant parameters. If a parameter is not specified, its internal default value is used. Each module is identified by a section header in square brackets "`[]`", followed by parameters and values, separated by an equal sign "`=`".

If you want to know more about a particular module and its functionality, all available modules and their parameters are described in detail in the user manual [**corryvreckan_manual_v1**]

Each configuration file needs to start with a **global** section labelled `Corryvreckan` in which the **detector geometry file** and the name of the **output *ROOT* file** are defined (we'll look at both in a moment).

The global section of the configuration file is followed by a module called `Metronome`, which defines time slices of 20 µs (0–20 µs, 20–40 µs, ...). These are called **events**. The subsequent analysis will be performed event-by-event (time-slice by time-slice), i.e. the entire analysis chain (all listed modules in the main configuration file) will be executed sequentially for one event and then start again at the top. The reason for using the `Metronome` is that not the entire data set, which can easily be multiple gigabytes, can be loaded into the RAM simultaneously. In addition, it is not necessary to have pixel data with a timestamp of 15 min in the RAM, when processing the first few seconds of the data set or vice versa.

The following module `EventLoaderTimepix3` loads all data (pixel column/row, timestamp, time-over-threshold) of the detector named `Timepix3_0` matching the current event, i.e. suiting into the time-slice defined by the `Metronome` as depicted in Figure 19.

The next module `Clustering4D` performs the clustering as described in Section 3.2.2 for all pixel hits of the current event.

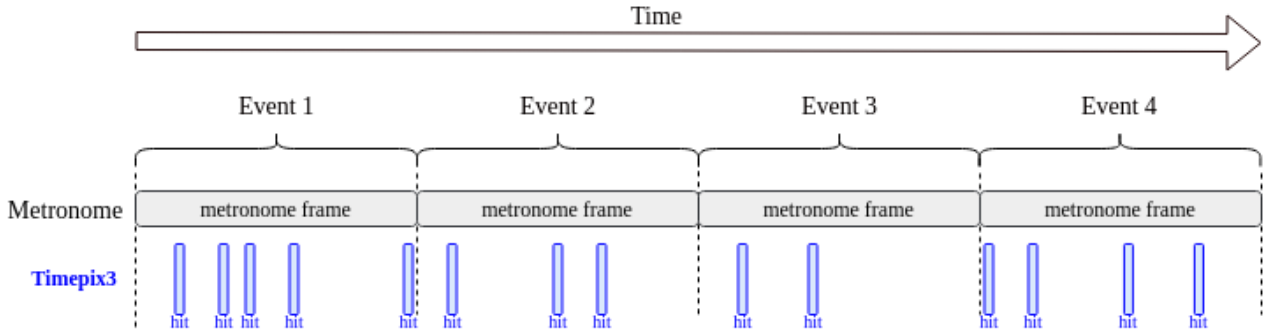


Figure 19: Pictoral representation of the event building mechanism. The `Metronome` defines the extend of the event and the `EventLoaderTimepix3` adds all pixels with a suitable timestamp to it.

After this, *Corryvreckan* moves on to the next event and repeats the procedure.

When the end-of-file of the data set is reached or the user interrupts the analysis, the **output *ROOT* file** is created and saved as specified in the global section of the configuration file. It can be found in the `output/` directory.

The input raw data files only contain pixel hit addresses (column/row), timestamps and ToT values. Hence, *Corryvreckan* does not know how the sensors are located relative to each other in the test-beam setup. Consequently, the above-mentioned **detector geometry file** is needed to specify the properties of all detectors as well as their position and orientation in space. It can be found in the `geometries/` directory. Open `01_example.geo` with a text editor and inspect it. In this file, each section describes one detector. The section header corresponds to the name of the detector, for which the number of pixels and the pixel pitch are provided. In addition, geometrical information such as the position and orientation of the sensor in space are specified.

Now let's run the example by typing:

```
$ corry -c 01_example.conf
```

Note: A warning will be shown that no material budget was specified for the detectors. The material budget can be deduced from the type of material and thickness of the sensor (e.g. 100 µm of silicon). This is not needed for the analysis of the data set at hand due to the high beam momentum at SPS. It is, however, relevant for a particle beam with a lower momentum such as at DESY (5.4 GeV at DESY vs. 120 GeV at SPS), where a more complex track model needs to be used to take the scattering of the particles in the material of the sensors into account. It is also displayed that no calibration file was provided. Calibration files are needed for a conversion of the ToT values into electrons for each detector. Also this is not relevevant for the analysis within this lab course. So both messages can be safely ignored.

In addition, you will see updated information about the ongoing analysis:

```
(STATUS) ========================| Event loop |========================
(STATUS) Ev: 599.9k Px: 2.68M Tr: 0.0k (0/ev) t = 11.998s
```

It displays the number of processed events ("Ev"), the sum of pixel hits in all detectors ("Px"), the number of found tracks ("Tr"), the ratio of tracks per event, and the run time of the data taking. Since we haven't used the tracking module yet, the number of tracks remains zero for now.

Once *Corryvreckan* is finished, go to the `output/` directory and inspect the output file:

```
$ root -l 01_example.root
```

In the *ROOT* file, you will find the following histograms for `Timepix3_0` in `EventLoaderTimepix3`:

- `pixelToT`: This is a 1D histogram of the ToT values of all pixels.
- `hitMap`: This is a 2D histogram with the size of the pixel matrix. It is filled with the pixel address of each hit.

The `Metronome` does not produce any histograms as it performs only a trivial task.

Tip: The entire data set contains ~ 24 events. You don't have to reconstruct the entire data set each time you change one parameter, especially while developing the analysis chain and testing your configuration files. To save time, you can run

```
$ corry -c 01_example.conf -o number_of_events=600000
```

to restrict the reconstruction to a defined number of events. Alternatively, you can press `Ctrl+C` while *Corryvreckan* is running. This way, *Corryvreckan* will finish analysing the current event, write out the *ROOT* file and then stop.

In the following, we'll build up the entire data analysis chain as described previously. For an overview of the steps, go back to Section 3.2 and have a look at Figure 15 and read the corresponding paragraphs again.

4.2 Reading in the Raw Data

Now that we know how to use *Corryvreckan*, we can start to read in the data of multiple detectors. To do this, we need to update both the main configuration file and the detector geometry file.

Open `geometries/02_read_data.geo` with a text editor. Using the geometrical information about the setup shown in Figure 20, complete the detectors geometry file. Since we do not yet know the precise (mis-)alignment in x- and y-direction, our "best bet" is to put all zeros.

Note: One telescope plane of your choice (e.g. the `Timepix3_3`) needs to be marked as `role = "reference"`. The *ATLASpix_Simple* needs to be marked as `role = "DUT"`. Since we don't know

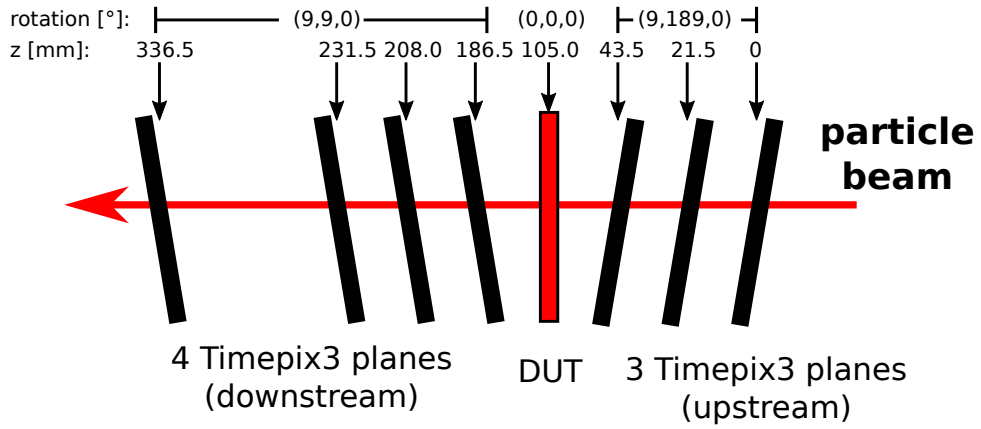


Figure 20: Schematic drawing of the detector setup with geometrical information on the plane distancing and rotations.

the spatial resolution of the *ATLASpix_Simple* yet, we can assume a binary resolution both in x and y (see Equation 5). For its time resolution assume 16 ns.

Next, let's open the main configuration file `02_read_data.conf`. Update the path to the detectors geometry file and choose a reasonable name for the output *ROOT* file. The `Metronome` module is needed for the same reasons as before and the module `EventLoaderTimepix3` will now handle all detectors of type `Timepix3` specified in the geometry file. To load the data of the device-under-test, the *ATLASpix_Simple*, we need to add a new dedicated event loader for this detector type. This is the `EventLoaderATLASpix`, which requires the following configuration:

```
[EventLoaderATLASpix]
type = "ATLASpix"
clock_cycle = 8ns # This is a chip setting, which was used during data taking.
clkdivend2 = 15   # This is a chip setting, which was used during data taking.
input_directory = <path/to/data>
```

This way, all *ATLASpix_Simple* pixel hits with the right timestamps will be added to the event as illustrated in Figure 21.

Once you've updated both the detectors geometry file and the main configuration file, re-run *Corryvreckan* and inspect the resulting output *ROOT* file. There should now be additional histograms for all sensors. If some histograms are empty, you've probably missed something in geometry or main configuration file.

Reminder: Use the option `-o number_of_events=600000` to save time for all further steps unless otherwise stated.

Inspect the following histograms:

- `hitMap` for all sensors: 2D histogram with the size of the pixel matrix, filled with all pixel hits per detector

Write a macro to draw the following plot:

- Divided canvas with the `hitMap` of all *Timepix3* planes and the *ATLASpix_Simple*.

Answer the following questions:

- Do you see any pattern? Can you explain it?
- Does any of the telescope planes look significantly different from the others? What might cause it? **Tip:** Compare the z-ranges of the different hitmaps and use the zoom on the z-axis.

4.3 Clustering

Until now, we have only read in individual pixel hits. To combine these into clusters, we use the module `Clustering4D`, which takes both spatial and time information into account. Add it to your main geometry file `03_clustering.conf` and again, find a reasonable name for the output *ROOT* file. By default, the `Clustering4D` will only search for touching neighbours. Split cluster, i.e. clusters with a gap, as depicted in Figure 22 are not allowed. The ToT-weighted centre-of-gravity is calculated as the cluster position. Note that the ToT values are used instead of the charge because no calibration was performed. The cluster timestamp corresponds to the earliest pixel timestamp within a cluster. For the time cut, it is recommended to start with a large cut of `time_cut_abs = 200ns`.

Re-run *Corryvreckan* and inspect the histograms in the output file. The relevant histograms are:

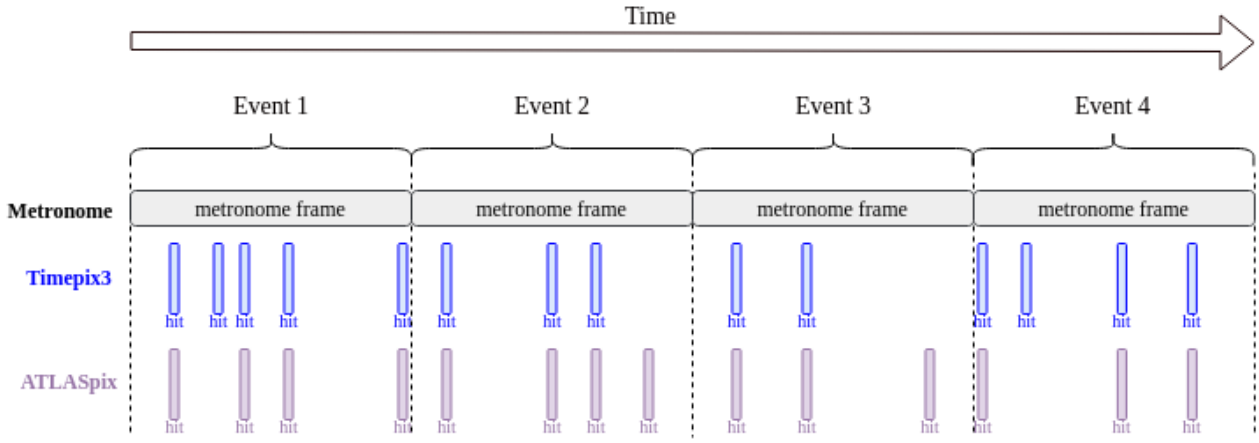


Figure 21: Pictoral representation of the event building mechanism. The `Metronome` defines the extend of the event and the `EventLoaderTimepix3` and `EventLoaderATLASpix` add all pixels with a suitable timestamp to it.

- `clusterSize`: Number of pixels in one cluster.
- `clusterTot`: ToT of the cluster (sum of all pixel ToTs).
- `clusterSeedTot`: ToT of the seed pixel of each cluster.
  Reminder: The seed pixel is the pixel with the largest charge within the cluster.
- `clusterWidthRow` and `clusterWidthColumn`: Extend of the cluster in row or column direction.

Write macros to draw the following plots:

- Overlay the `clusterSize` distribution for the *ATLASpix_Simple* and the *Timepix3* (a plane of your choice). **Tip:** Use the draw option "same" to draw them on top of each other for a better comparison.
- Overlay the `clusterWidthColumn` and `clusterWidthRow` distributions for the *ATLASpix_Simple*.
- Overlay the `clusterSeedTot` and `clusterTot` for both the *ATLASpix_Simple* and the *Timepix3*.

Don't forget to label the axes properly and add a legend.

Answer the following questions:

- Compare the cluster sizes for the *Timepix3* and the *ATLASpix_Simple*. How do they differ? Why?
- Compare the cluster width in column and row for the *ATLASpix_Simple*. How do they differ? Why?
- Compare the seed pixel ToT and the cluster ToT for the *ATLASpix_Simple* (for the *Timepix3*). Which difference can you observe for the *ATLASpix_Simple* and the *Timepix3*? Why?

4.4 Correlations

So far, the alignment specified in the detector geometry file was only measured "by hand". It is (most likely) not good enough to find tracks because even a misalignment below 1 mm already corresponds to an offset of many pixels. To have an initial estimation on how good the alignment is, we can investigate the spatial correlations. For this, add the module `Correlations` to the main configuration file `04_correlations_telescope.conf`. Again, use `time_cut_abs = 200ns`.

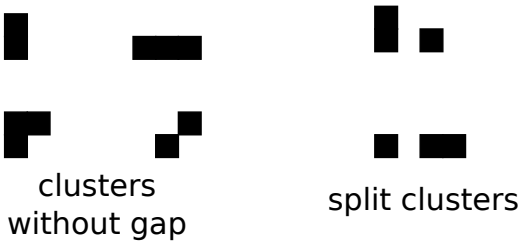


Figure 22: Examples of different clusters with and without a gap.

Re-run *Corryvreckan* and inspect the output file. The following histograms are the most important:

- `correlationsX` and `correlationY`: These show the spatial correlations between each sensor and the reference sensor (`type = reference` in the geometry file) on cluster-level.

Peaks should be visible in the spatial correlation plots in both X and Y for all detectors. The peaks are superimposed with a "triangular" background, which comes from non-correlated clusters, either from different tracks or from noise. It is there because we correlate all clusters will all clusters within the entire event (20 µs).

Answer the following questions:

- How can translational misalignments be observed in the correlations? How large are they approximately in your case? How many pixel pitches do they correspond to?
- How can a bad rotational alignment be observed in the correlations?

You can test your answers by modifying the position and rotation in the detector geometry file by hand and checking the result. Make sure to rename the output file to avoid overwriting the one from the previous analysis.

4.5 Alignment and Tracking

As discussed above, the detector geometry file describes all sensor planes present in the setup as well as their position and rotation. The z-positions of all planes **must** be measured by hand in the existing setup and entered into the geometry file. They cannot be reconstructed a posteriori from the raw data. The x- and y-positions as well as the rotations can be measured only rough by hand. However, they have a strong influence on the tracking because even an offset well below 1 mm easily corresponds to a shift by multiple pixel pitches.

Consequently, an alignment procedure is needed, in which the detector planes are shifted and rotated iteratively relative to the detector marked as `role = reference` to increase the tracking quality.

At this point, it is important to recall the difference between **correlations** and **residuals**.

- A spatial **correlation** plot is filled with the spatial distance of any cluster on a particular detector and any cluster on the reference plane. No tracking is involved.
- On the contrary, a **spatial residual** plot shows the difference of the interpolated track intercept onto a particular plane and the position of the cluster on this plane. Here, we need to distinguish between **biased** residual, in which case the cluster on the plane is part of the track and **unbiased** residuals, in which case the cluster has been associated to the track after the track fitting and is not part of the track itself.

$$x_{\text{correlation}} = x_{\text{cluster on reference detector}} - x_{\text{cluster on this detector}} \tag{10}$$

$$\text{biased: } x_{\text{residual}} = x_{\text{track intercept on this plane}} - x_{\text{track cluster on this plane}} \tag{11}$$

$$\text{unbiased: } x_{\text{residual}} = x_{\text{track intercept on this plane}} - x_{\text{associated cluster on this plane}} \tag{12}$$

Ultimately, the goal of the alignment is to force the **residuals** to be centered around zero. The **correlations** do not necessarily need to centered at zero as a possible offset reflects the *physical displacement* of a detector plane in x and y with respect to the reference plane. This is illustrated in Figure 23. For the assumed ideal alignment, which is what we start with, the distance between track intercept and hit on the blue plane is large. If initial alignment is too far away from the 'real' alignment, the track finding and fitting will not work at all. For the fine-tuned alignment, which is what we want to achieve with the alignment procedure, the residuals (i.e. the distances between track intercept and associated hits on each plane) are close to zero. On the contrary, the correlations between the blue and the red plane are 'off' by two rows representing the physical misalignment between the two planes.

However, it is useful to start by inspecting the **correlation** plots in the beginning when the assumed ideal alignment is too different from the real alignment and no or only few tracks can be found such that the residual plots cannot be filled reasonably.

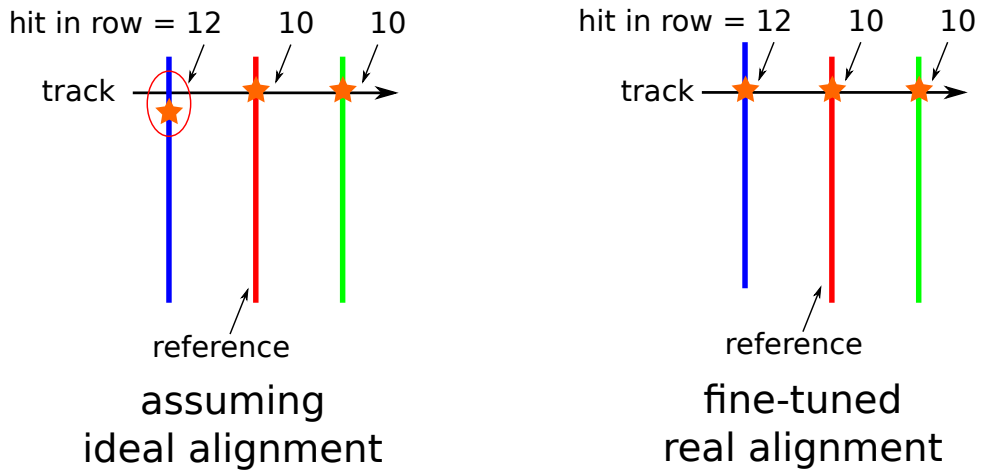


Figure 23: Illustration of a track fit through an assumed ideal and the fine-tuned real alignment of the sensor planes (exaggerated). Different telescope planes are indicated in different colours.

4.5.1 Prealignment

If the misalignment is too large, a reasonable track reconstruction is not yet possible. In this case, the "best bet" is to shift all spatial correlations to zero despite of what was discussed above.

In principle, this can be done by hand by inspecting all correlation plots and editing the detector geometry file. Alternatively, the `Prealignment` in *Corryvreckan* can automatically determine the peak position of the spatial correlations and update the detectors geometry file accordingly.

To use it, edit `05_prealignment.conf` and add

```
[Prealignment]
time_cut_abs = 200ns
```

```
method = gauss_fit
fit_range_rel = 10
```

With these settings a Gaussian is fitted to the peak to determine its position with a fit range of $10\times$ the spatial resolution of each detector as specified in the detector geometry file.

You should also add `detectors_file_updated = ...` in the global section of the main configuration file to avoid that the initial geometry file is overwritten.

The output *ROOT* file from this analysis does not show the effect of the updated geometry because the corrections are calculated at the very end of the analysis when the histograms are already filled. Nevertheless, you should have a look at correlation plots in the prealignment module to verify that the peaks were detected correctly and the Gaussian fits look reasonable, i.e. represent the peak position well. If the peaks have not been found, you may need to adjust the fit range or re-run the analysis with a larger number of events.

After running the prealignment, re-use the previous configuration `04_correlations.conf` with the updated detector geometry file (and changed output file name). Compare the correlation plots of all detectors you get with the updated geometry to the initial geometry and verify that the peaks have been shifted towards zero.

Write a macro to draw the following plots:

- Divided canvas with the correlations in x for all *Timepix3* planes before and after the prealignment to show that they are centered at zero now.
- Divided canvas with the correlations in y for all *Timepix3* planes before and after the prealignment to show that they are centered at zero now.

4.5.2 First Tracking

Now that the correlation peaks are close to zero, we can have a first go on the tracking. For this, edit `06_tracking.conf` and add the module `Tracking4D`. As for the previous modules, use a time cut of 200 ns. In addition, we should specify a spatial cut, which defines a "search ellipsoid" around the projected track on the next plane. Try to use

```
time_cut_abs = 200ns
spatial_cut_abs = 200um,200um
```

In addition, we should set

```
exclude_dut = true
```

to make sure that tracks are built with clusters from the reference telescope only and the device-under-test is excluded. The track timestamp is set to be the average timestamp of all clusters along the track.

Run *Corryvreckan* and inspect the output file. The most important histograms for the tracking module are:

- `residualsX` and `residualsY` for each detector: These are the biased residuals discussed above.

- `trackChi2ndof`: This is the track χ^2/ndof (number of degrees of freedom) for each track fit. It is calculated from the residuals of the track and the clusters on each plane used for the track, as well as the resolutions specified in the detector geometry file.

Answer the following questions:

- Compare the residuals with the correlations from the previous analysis. What is the difference?
- As you remember, a reasonable fit should have a χ^2/ndof ~ 1. What does the track χ^2/ndof look like for you? Why? **Tip:** Right-click on the statistics box and choose `SetOptStat`. Then add two additional 'ones' to obtain the overflow bin.

4.5.3 Telescope Alignment

With the coarse prealignment and a working track reconstruction (even though the track χ^2/ndof might be bad), we can proceed with the precise alignment – still excluding the DUT. For this, add the module `AlignmentTrackChi2` to the configuration file `07_telescope_alignment.conf`. This module iteratively varies the position and rotation of each sensor, which is used in the tracking, refits the tracks and tries to minimize the track χ^2. Initially, you can run it with all parameters on default.

Note: For a good results more statistics is needed than in the previous steps. However, it is still not necessary to analyse the entire data set. For instance, you can use the option `-o number_of_tracks=250000`.

As for the prealignment, the histograms in the output file show the initial alignment because the updated geometry file is only created at the very end of the analysis, when all histograms are already filled. To test the new alignment, update the previous configuration `06_tracking.conf` with the new geometry file and re-run the analysis.

If the result is not satisfactory, you can re-run the alignment a second time (don't forget to update `detectors_file` and `detectors_file_updated`). In addition, you can try to allow only shifts or only rotations by using the following parameters:

```
[AlignmentTrackChi2]
align_position = true    # set to false to disable
align_orientation = true # set to false to disable
```

The alignment is reasonable when the residuals on all telescope planes are well centered at zero (the mean of the histogram should be smaller than $1\,\mu\text{m}$) and symmetric. In addition, the track χ^2/ndof distribution should be peaking around 1.

Write macros to draw the following plots:

- Overlay the χ^2/ndof distributions before and after the alignment.
- Divided canvas with the residuals in x for all *Timepix3* planes before and after the alignment.
- Divided canvas with the residuals in y for all *Timepix3* planes before and after the alignment.

Tip: Instead of `hist->Draw()` use `hist->DrawNormalized()` to renormalize the histograms. Otherwise the histograms are hard to compare because the numbers of entries are very different.

Explain what you see.

4.5.4 First DUT Residuals

Now that a good alignment of the reference telescope was achieved, we can go ahead with the alignment of the device-under-test.

Since the DUT is excluded from the tracking, the clusters on the DUT are not part of a track. Instead, we need to associate the clusters on the DUT to tracks in a different way. To do so, we use the module `DUTAssociation`. For each track, it calculates the track intercept with the DUT and searches for clusters within a defined spatial and time window, which can be associated with this track. Since the DUT may not yet be well aligned, use generous cuts like

```
time_cut_abs = 200ns
spatial_cut_abs = 350um, 350um
```

In addition, the module `AnalysisDUT` can be used to calculate the DUT residuals.

Add both modules to `08_dut_residuals.conf` and run *Corryvreckan*. In the output file, look at the following histograms:

- `residualsX` and `residualsY`: These are the unbiased residuals as discussed above.

Answer the following question:

- Why should the DUT be excluded from tracking? For instance, think about the effect on the hit detection efficiency of the DUT.

4.5.5 DUT Alignment

In the prealignment step, the correlations have already been shifted toward zero for the DUT. Consequently, we can proceed with the precise alignment of the DUT. Since it is not part of the tracking, we need to apply a different method than the track χ^2 minimization here. The module `AlignmentDUTResidual` is used, which iteratively varies the position and rotation of the DUT to center the unbiased residuals in x and y around zero and minimize their widths.

To use it, add this module to `09_alignment_dut.conf`. As before, you should also add

```
detectors_file_updated = ...
```

in the global section of the main configuration file to avoid that the initial geometry file is overwritten.

For a good alignment result, it is recommended to use a large number of tracks. As for the telescope alignment described above, the new alignment will be calculated and written out at the very end of the analysis, so it will not yet be seen in the histograms of the output file. To see how the alignment improved, update the previous configuration `08_dut_residuals.conf` with the new geometry file, rename the output *ROOT* file and re-run *Corryvreckan*. The mean of the residuals should be below $1\,\mu\text{m}$ and the residuals should be symmetric.

If the result is not yet satisfactory, try to run the alignment a second time. Again, don't forget to update the configuration file to use the correct geometry.

Write a macro to draw the following plots:

- Overlay the residuals in x/y for the *ATLASpix_Simple* comparing the initial and the final alignment.

Explain what you see.

Note: If you're running out of time or have trouble at this step, ask your supervisor for help. He/she might provide you with the finished alignment such that you can continue with the other tasks.

4.6 DUT Analysis

With a fully aligned setup, we can proceed to the actual DUT analysis, in which we can characterize the sensor and investigate its performance. Update `10_dut_analysis.conf` with the most recent geometry file. We now have the full data analysis chain in place as described in Secion 3 and illustrated in Figure 15 (apart from the efficiency analysis, but we'll get to that later).

For reliable analysis results, it is important to have a high number of tracks for good statistics. Again, you can test your configuration file with a short analysis, but for the final results, the entire data set should be analysed for the best possible statistics.

Tip: To save time, have a quick look at Section 4.6.3 and add the corresponding module already now.

Note: If you are working on the CIP-Pool, limit your analysis to 18 million events by using the option `-o number_of_events=18000000`. Otherwise, the PC may run out of resources and crash.

After running *Corryvreckan*, inspect the output file and compare the following histograms:

- `Correlations/ATLASpix_0/hitmap_clusters`: 2D histograms of all clusters on the DUT
- `AnalysisDUT/ATLASpix_0/clusterMapAssoc`: 2D histogram of all clusters on the DUT that have been associated to a track

Write a macro to compare the above-mentioned histograms. Answer the following question:

- In one case, not the entire matrix is filled. Can you explain why? Think about the geometrical dimensions of the *Timepix3* sensors and the *ATLASpix_Simple*.

4.6.1 Spatial Resolution

Next, we can have a closer look at the spatial resolution. As described in Section 3, the width (RMS) of the spatial residuals can be used to estimate the spatial resolution. Look at the following histograms:

- `residualsX` and `residualsY` in `AnalysisDUT`

Write a macro to draw the following plot:

- Overlay spatial residuals in x and y for the *ATLASpix_Simple*.

Answer the following questions:

- Are these biased or unbiased residuals? Why?
- Can you quantify the spatial resolutions?
- Are the spatial resolutions what you expected? Why?
  Tip: Look at the cluster size distribution for the *ATLASpix_Simple*.
- Why are the spatial residuals non-gaussian?
  Tip: Compare the pixel size of the *ATLASpix_Simple* with the precision of the telescope.

4.6.2 Time Resolution

Similarly, the time resolution can be inferred from the track-cluster time residual. Find the following histograms:

- `residualsTime`: This is the time residual between the reference tracks and the associated clusters on the DUT.
- `residualsTimeVsTot`: This is the same time residual between the reference tracks and the associated clusters on the DUT as in the previous histogram plotted against the seed pixel ToT of the associated cluster.

Write a macro to draw the following plots:

- Draw the time residual for the *ATLASpix_Simple* and fit a Gaussian to the distribution. Make sure to choose an appropriate axis range.
- Draw the `residualsTimeVsTot`.

Answer the following questions:

- Is the peak position (offset from zero) relevant for the time resolution? What may cause it?
- Describe the shape of the distribution and compare it to the 2D histogram. Why is there a strong non-gaussian tail? Does it make sense on which side it is?
- Can you quantify the time resolution? Compare the Gaussian fit with the RMS of the distribution. Which one is more appropriate?

4.6.3 Hit Detection Efficiency

For efficiency measurements, a dedicated module called `AnalysisEfficiency` exists in *Corryvreckan*. It can be placed behind the other analysis module `AnalysisDUT` in the main configuration file. Add it to `10_dut_analysis.conf` and re-run *Corryvreckan*.

The hit detection efficiency can be read off from the terminal or the *ROOT* file. In addition, look at the following histograms:

- `chipEfficiencyMap_trackPos`: 2D map of hit detection efficiency across the whole matrix of the sensor.
- `pixelEfficiencyMap_trackPos`: 2D map of the efficiency distribution inside a pixel cell. All pixels are superimposed.

Write macros to draw the two histograms mentioned above.

Answer the following questions:

- How large is the hit detection efficiency?
- Why is the chip efficiency map not filled entirely?
- Do you recognize any pattern in the pixel efficiency map? Can you explain why?

Optional: Repeat the efficiency analysis multiple times while systematically scanning the spatial cuts in `DUTAssociation`. Plot the measured efficiency versus the association cut. What do you observe? Can you explain why?

5 Closing Remarks

To conclude the lab course and prepare for the written report, go through Section 4 again and make sure to consider all questions. In additon, plot all requested histograms and be able to explain them.

Important:

- Not all histograms need to be part of the "main PDF". Clearly label them, i.e. combine the number of the exercise with a reasonable name for the histogram.

If you have any feedback, suggestions for modifications or you found any errors in the manual elsewhere, please forward it to your tutor and help improving this lab course.

5.1 Possible Further Steps

The scope of this lab course is fairly limited in time. Nevertheless, you have built up an entire test-beam reconstruction chain and determined some of the crucial performance parameters of a pixel sensor prototype using real data.

If we had more time and wanted to dive into some more details, the following things could be done:

- Investigate the effect of noise on the analysis. Noisy pixels could be identified and excluded from the analysis.
- You've clearly seen the effect of timewalk on the time resolution of the sensor. This effect could be corrected for to achieve an improved time resolution.
- In addition, further data sets could be analysed with different chip operating conditions. Typically, the applied bias voltage is scanned systematically to investigate its effect on the sensor performance. In addition, the efficiency could be studied for varying hit detection thresholds.