DEUTSCHES ELEKTRONEN-SYNCHROTRON
Ein Forschungszentrum der Helmholtz-Gemeinschaft

DESY 20-210
CLICdp-Pub-2020-005
arXiv:2011.12730
November 2020

Corryvreckan: A Modular 4D Track Reconstruction and Analysis Software for Test Beam Data

D. Dannheim, K. Dort, I. Kremastiotis, J. Kröger, M. Munker, T. Vanat, M. Williams

CERN, Geneva, Switzerland

L. Huth, P. Schütze, S. Spannagel

Deutsches Elektronen-Synchrotron DESY, Hamburg

D. Hynds

NIKHEF, Amsterdam, The Netherlands

F. Pitters

ÖAW, Institut für Hochenergiephysik (HEPHY), Wien, Austria

ISSN 0418-9833

NOTKESTRASSE 85 - 22607 HAMBURG

To be sure that your reports and preprints are promptly included in the HEP literature database
send them to (if possible by air mail):

DESY Zentralbibliothek Notkestraße 85 22607 Hamburg Germany	DESY Bibliothek Platanenallee 6 15738 Zeuthen Germany

Prepared for submission to JINST

Corryvreckan: A Modular 4D Track Reconstruction and Analysis Software for Test Beam Data

Dominik Dannheim,[a] Katharina Dort,[a,1] Lennart Huth,[b] Daniel Hynds,[c] Iraklis Kremastiotis,[a] Jens Kröger,[a,2] Magdalena Munker,[a] Florian Pitters,[d] Paul Schütze,[b] Simon Spannagel,[b,*] Tomas Vanat,[a] Morag Williams[a,3]

[a] *CERN, Espl. des Particules 1, 1211 Geneva, Switzerland*
[b] *DESY, Notkestraße 85, 22607 Hamburg, Germany*
[c] *Nikhef, Science Park 105, 1098 Amsterdam, Netherlands*
[d] *HEPHY, Nikolsdorfer G. 18, 1050 Wien, Austria*

E-mail: simon.spannagel@desy.de

Abstract: *Corryvreckan* is a versatile, highly configurable software with a modular structure designed to reconstruct and analyse test beam and laboratory data. It caters to the needs of the test beam community by providing a flexible offline event building facility to combine detectors with different readout schemes, with or without trigger information, and includes the possibility to correlate data from multiple devices based on timestamps.

Hit timing information, available with high precision from an increasing number of detectors, can be used in clustering and tracking to reduce combinatorics. Several algorithms, including an implementation of Millepede-II, are provided for offline alignment. A graphical user interface enables direct monitoring of the reconstruction progress and can be employed for quasi-online monitoring during data taking.

This work introduces the *Corryvreckan* framework architecture and user interface, and provides a detailed overview of the event building algorithm. The reconstruction and analysis capabilities are demonstrated with data recorded at the DESY II Test Beam Facility using the EUDAQ2 data acquisition framework with an EUDET-type beam telescope, a Timepix3 timing reference, a fine-pitch planar silicon sensor with CLICpix2 readout and the AIDA Trigger Logic Unit. The individual steps of the reconstruction chain are presented in detail.

Keywords: Software architectures (event data models, frameworks and databases); Analysis and statistical methods; Data processing methods; Pattern recognition, cluster finding, calibration and fitting methods; Solid state detectors

[1] Also at University of Gießen, Germany.
[2] Also at University of Heidelberg, Germany.
[*] Corresponding author.
[3] Now at ESRF, 71 Avenue des Martyrs, 38000 Grenoble, France.

Contents

1 Introduction

Test beam measurements using beam telescopes for reference track reconstruction have become important tools for detector R&D. The operation of detector prototypes in conditions as close as possible to the final deployment situation enables their testing, qualification, and characterisation in terms of efficiency, spatial resolution, and timing performance. Reconstruction software frameworks specifically written for test beam experiments, such as EUTelescope [1], Judith [2], Proteus [3] or Kepler [4], have served the community as tools for many years. With new generations of silicon detectors emerging and a diversification of the R&D program conducted at test beam facilities, new requirements on flexibility and interoperability pose a challenge to data reconstruction. Trigger-based devices, e.g. those designed to operate at the HL-LHC [5], are operated together with trigger-less detectors developed by the linear collider communities of CLIC [6] and ILC [7], or with fully data-driven multi-purpose detectors such as Timepix3 [8].

The *Corryvreckan* reconstruction and analysis framework has been designed specifically to address the challenges that come with reconstructing data recorded with such heterogeneous setups while at the same time improving the user experience by lowering the entry barrier for new users to analyse their data. It is a lightweight and fast framework written in modern C++, with a modular design. The core components deal with the parsing of configuration files, the central event loop, and coordinate transformations while separate modules implement the individual steps of the reconstruction process. Data are exchanged between individual module instantiations using a central clipboard storage. Its flexible offline event building algorithm allows the reconstruction chain to be adapted to different analysis requirements and to serve particle detectors with a wide variety of readout schemes. The flexibility of the framework also enables the analysis of data recorded in stand-alone laboratory experiments with single detectors.

Corryvreckan is published as free and open source software under the MIT license, the code can be obtained from the project software repository [9]. A comprehensive user manual has been published [10] and is continuously updated as the framework is extended with new features, the most recent version is available online [11]. The framework has greatly profited from the development of the Allpix^2 Generic Pixel Detector Simulation Framework [12], as both frameworks follow similar philosophies in terms of modularity and configurability, and share parts of their code base. Furthermore, thanks to a dedicated module in Allpix^2, *Corryvreckan* is able to directly process simulated detector responses with the same reconstruction chain configured to analyse experimental data.

The framework has already been used in a number of different scenarios, such as the reconstruction of data recorded using the high-rate beam telescope of the CLIC detector & physics (CLICdp) collaboration at the CERN SPS [13, 14], the analysis of test beam data from EUDET-type beam telescopes at the DESY II Test Beam Facility [15–19], the qualification of calorimeter modules with minimum ionising particles [20], and the reconstruction of silicon pixel detector simulations and their comparison to data [12].

This paper describes the architecture and functionality of the *Corryvreckan* framework as released in version 2.0. Section 2 summarises the software structure as well as its user interface. A detailed description of the reconstruction and analysis chain is presented in Section 3, highlighting features unique to this framework. Section 4 introduces the offline event building algorithm and provides detailed examples for different setups and configurations. The full reconstruction and analysis chain is demonstrated in Section 5 using test beam data recorded with a combination of triggered, frame-based and data-driven devices. Finally, a summary and an outlook to future developments are given in Section 6.

2 Architecture of the Framework

The *Corryvreckan* framework[1] is designed as a modular software where the algorithms for tasks such as pattern recognition, track reconstruction, and data analysis are implemented as independent modules, loaded as needed at run time. Key components such as user interaction, the event loop, and parsing of configuration or geometry description are handled centrally by the framework core. This section provides an overview of this functionality.

2.1 Core Components of the Framework

Corryvreckan provides a command line interface for interaction with the user. The main configuration file constitutes the only mandatory argument, containing the reconstruction chain definition. Individual framework parameters can be supplied or overwritten by passing them to the framework directly on the command line, e.g. for parameter scans, or to submit runs to a batch processing system. The output verbosity of the program as a whole as well as of individual modules can be adjusted via configuration file or command line parameters.

An example for the command line interface of *Corryvreckan* is:

```
corry -c main.config -o parameter=5 -v WARNING
```

where the main configuration file is specified via the `-c` option, the `-o` flag allows the definition of an additional framework parameter, and `-v` option is used to configure verbosity. A more detailed description of the interface, including all options, can be found in the user manual.

Three different types of modules are known to *Corryvreckan*. Global modules are meant to operate on data from all of the detectors available, e.g. implementing a track-finding algorithm, and are only instantiated once per run. Detector modules only operate on the data of an individual detector of the setup, e.g. for clustering, and can be limited to processing data only for a subset of the available detectors. Finally, *Device Under Test (DUT)* modules are only created for detectors that are marked as such. These are typically analysis modules that calculate observables for the detector under investigation. It is also possible for developers to restrict the use of individual modules to specific detector types at compile-time, which then omits the creation of the corresponding instances by the module manager. The separation of modules into these categories simplifies the development of new algorithms as their scope is always well-defined and the processing of data from multiple detectors is inherently abstracted from the algorithm by the framework core.

In contrast to other reconstruction frameworks, *Corryvreckan* does not rely on a specific definition of an event, such as all data related to a single trigger decision, but leaves it to the user to configure, which data constitute one event. The exact definition of what belongs to this chunk is configured by the user via the event building algorithm described in detail in Section 4. Data are passed between the individual modules by means of a clipboard onto which each module can place new data or read and alter existing data. The central event loop takes care of clearing the clipboard content after processing of an event has been finalised, and before the subsequent event is executed. A persistent storage space is available on the clipboard to cache data required beyond the scope of a single event. This storage can be used to preserve data beyond the lifetime of a single event, e.g. to accumulate tracks for performing alignment, and is only cleared at the end of the run.

2.2 Coordinate Systems and Transformations

Within *Corryvreckan*, the local coordinate system for a detector plane is defined as a right-handed Cartesian coordinate system, with the x and y axes defining the sensor plane, the z axis pointing towards the readout side of the sensor. Its origin is placed at the centre of the active pixel or strip matrix of the sensor. The global coordinate system for the full detector setup is defined as a right-handed Cartesian coordinate system as well,

[1] The framework is named after a whirlpool between the islands of Jura and Scarba in the Inner Hebrides.

```
[Corryvreckan]
log_level = "WARNING"
detectors_file = "geometry.conf"
number_of_tracks = 900000

[EventLoaderEUDAQ2]
file_name = "data/run0000456.raw"
inclusive = false
buffer_depth = 1000
shift_triggers = -1
```

(a) Example for the main configuration with a global and a module section defining an *EventLoaderEUDAQ2* module and four associated parameters.

```
[W0013_D04]
number_of_pixels = 256, 256
orientation = 10.9deg, 17.2deg, -1.3deg
orientation_mode = "xyz"
pixel_pitch = 55um, 55um
position = 886.5um, 270um, 0
spatial_resolution = 4.8um,4.8um
time_resolution = 1.56ns
type = "Timepix3"
role = "reference"
```

(b) Example for a detector geometry configuration. The section name *W0013_D04* corresponds to the detector name, the parameters determine its properties.

Figure 1: Excerpts from the two *Corryvreckan* configuration files, the main configuration file (a) and the geometry description file (b).

where the z axis points in the direction of the particle beam. The orientation of a detector is described by extrinsic active rotations around the geometrical centre of the sensor. Coordinate transformations between local and global coordinate systems for each detector are provided by the detector class of the framework core.

2.3 Configuration Files

The reconstruction and analysis chain of *Corryvreckan* as well as the detector geometry are configured through text files with an intuitive syntax: a main configuration file and a geometry description file. These files contain section headers to identify modules, and key/value pairs for the individual configuration parameters. *Corryvreckan* is capable of interpreting the units provided along with configuration parameters and performs all calculations in a set of internal base units to minimise the need for CPU-intensive conversions and the permanent carrying of units within the code. If a parameter is provided with units, the value is converted directly into these internal units or otherwise interpreted directly in the base units of the framework.

2.3.1 Main Configuration File

The main configuration file contains the structure of the reconstruction chain as well as all relevant module parameters. Modules are identified by a section header in square brackets, followed by key/value pairs for this specific module as shown in Figure 1a. Here, the *Corryvreckan* section holds global configuration parameters, while the section *EventLoaderEUDAQ2* instantiates a module of the given type. The four parameters for this particular module are set to a string, a Boolean flag, and integer values.

For each event, modules are executed in the order they are listed in the configuration file. This is of particular importance for the event building algorithm, which is described in detail in Section 4. The configuration parameters available for each module as well as global parameters for the whole framework are listed in the user manual.

2.3.2 Geometry Configuration File

The properties of all detectors as well as their position and orientation in the global reference frame are specified in the geometry configuration file. *Corryvreckan* will only process detectors that are present in this geometry, data from other devices are ignored. Each section of the configuration describes one detector and the section header serves as identifier for the device throughout the reconstruction. An example for an individual detector

is shown in Figure 1b. The section contains parameters to describe detector properties such as pixel or strip pitch and matrix size, but also its position and orientation with respect to the global reference frame.

Of special relevance is the *role* parameter that defines how this specific detector behaves in the reconstruction. It can be configured as DUT so that all DUT-specific modules will be instantiated for this detector; as reference detector against which the alignment is calculated; or as auxiliary device, which is allowed to contribute additional information to the event building process but does not participate in the reconstruction directly. An example of auxiliary devices are scintillator triggers that can provide additional time stamps. All other detectors will serve as members of the beam telescope and are used to form reference tracks.

2.4 Data Model & Object History

Corryvreckan provides classes to store different quantities relevant to the reconstruction chain, such as pixel hits, clusters or particle tracks. These objects inherit from a common C++ base class that enables storage on the central clipboard of the framework and seamless exchange with other modules and components of the framework as well as writing to and reading from files.

Furthermore, it allows for the storage of inter-object relations via persistent pointers. Using this, the history of individual objects can be resolved and e.g. the original set of pixel hits that lead to the formation of a track can be traced. This is both possible during processing of the data within *Corryvreckan* and after writing intermediate results to disk.

The separation of the data objects from the algorithmic parts of the framework facilitates the central implementation of track models, which can be used by different, independent track finding modules without reimplementation of the track fitting algorithm.

2.5 Software Development

The development of *Corryvreckan* follows best practices for software development by adopting an agile development model, requiring strict format compliance, enforcing a rigorous testing scheme, and by using the C++14 language standard [21]. The code base is well documented; a full class reference is automatically generated from the source code using `doxygen` [22] and provided on the website [23]. New code contributions to the framework are always reviewed by at least one other developer before merging into the central repository.

Releases of the project follow the semantic versioning scheme, where changes breaking backwards compatibility are only introduced in every new major version, while minor versions add new features and patch versions only provide bug fixes to existing functionality.

Proper functioning of modules and the framework core components as well as compatibility with previous versions of the *Corryvreckan* framework are ensured by a continuous integration, which builds, checks and tests the software on all supported platforms. This process is integrated with the project software repository and is executed for every new code submission suggested for inclusion. Here, *checking* refers to directly code-related tasks such as ensuring proper formatting and coding style as well as spell checking, while *testing* implies running complete analyses using reference detector data. The reference is a centrally hosted, publicly accessible set of data recorded during different test beam campaigns. They are downloaded automatically by the testing system on demand, and particle tracks are reconstructed and the result compared to a predefined value. With this data-driven system test, the framework code base can be protected against undesired changes that deteriorate or break existing functionality. The test suite is extended whenever needed to cover new features or supported detectors.

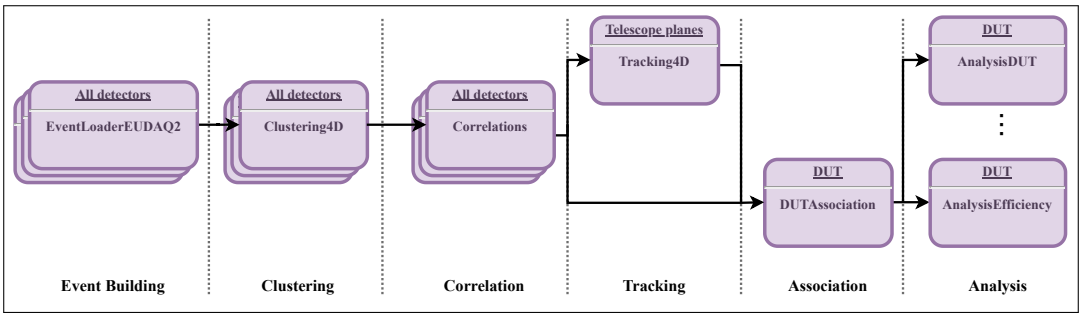


Figure 2: Flow chart of a typical *Corryvreckan* reconstruction chain with a beam telescope setup and a single DUT. Multiple instantiations of a module for different detectors are represented by stacked layers, offsets indicate modules operating on a subset of detectors, and the dots in the *Analysis* stage indicate the possible inclusion of further analysis modules.

3 Reconstruction & Analysis of Test Beam Data

The reconstruction chain of *Corryvreckan* is built via the main configuration file by adding modules for each individual task to be performed for the analysis. This section provides a general overview of a typical configuration of such a reconstruction chain.

Modules are separated from the core framework and can therefore be flexibly inserted or removed as required to create the desired analysis chain. A typical example of a reconstruction chain is represented in Figure 2. As discussed in Section 2, the order of execution of the modules is defined by their order in the configuration file. This means they have to be placed at a point in the reconstruction chain where all relevant data are expected to be available. Each of the different stages of the reconstruction will be briefly discussed in the following.

3.1 Event Building

The preparation of pixel or strip hit information from all detectors, such as the decoding of raw data or the application of charge or hit-time calibrations, is performed by a group of modules referred to as *EventLoaders*. *Corryvreckan* comes with a set of detector-specific *EventLoader* modules, which directly decode and process the relevant raw data, but also with the *EventLoaderEUDAQ* and *EventLoaderEUDAQ2* modules, which allow data to be read and used from any detector supported by the EUDAQ1 [24] or EUDAQ2 [25] frameworks, respectively.

The first module of the reconstruction chain defines the extent of the currently processed data chunk, defined by means of start and end timestamps or by one or multiple associated trigger numbers. All subsequent modules have to adhere to this event definition and only provide matching data.

A special role is played by the *FileReader* module that allows intermediate results of a previous reconstruction run or a simulation to be read from a file. All *Corryvreckan* objects associated with the current event are placed on the clipboard for further treatment. This allows data to be preprocessed by building events and performing the clustering, while deferring the tracking to a later stage of the data processing.

A detailed description of the event building process with different device combinations is provided in Section 4.

3.2 Clustering and Hit Position Interpolation

Corryvreckan offers two different detector modules for clustering, one that performs a closest-neighbour search for all pixel or strip hits on a detector plane, and one that uses pixel hit timestamps as additional criterion

for associating pixels with a cluster. Using timing information drastically reduces combinatorics and allows individual clusters to be recovered accurately in high-rate environments. Here, strip detectors are treated as detectors with a single row of elongated pixels.

The hit position is calculated from the cluster by means of a charge-weighted centre-of-gravity algorithm if charge information is available. For binary hit information, the hit position is calculated as the arithmetic mean.

Two modules are provided to perform an η-correction for non-linear charge sharing [26] of the cluster centre position in a two-pass approach. In the first pass, the *EtaCalculation* module is used to record the η-distribution of the given detector, and to perform a fit of the cumulative η-function. The fitting function used can be defined by the user via the configuration file. The function and the parameters from this fit can then be used in the second pass by the *EtaCorrection* module, which corrects the hit position accordingly. Only clusters with a width or height of two pixels are considered and corrected by this algorithm. Since this implementation makes use of the reference tracking information, an independent data set should be used for deriving the correction parameters in order not to bias the results.

3.3 Track Finding and Fitting

Corryvreckan separates the procedure of finding track candidates from the actual fitting routine of particle tracks. The track finding algorithms are realised as *Tracking* modules, which have access to the different track models implementing the fit. In the tracking module, a track model is selected via the configuration and a track candidate is constructed with hits from the relevant detectors. Then, the fit is performed by the track object and the result reported back to the tracking module.

Different tracking modules are available. The *Tracking4D* module uses two planes of the telescope to build a first track candidate and subsequently adds more clusters from additional telescope planes if they are within a configurable spatial and time window. Depending on the detectors and the beam environment, the selection of clusters based on proximity in time as well as space can reduce the number of track candidates to process significantly compared to tracking based only on the spatial correlation of clusters. The *TrackingMultiplet* module is based on an independent search for particle trajectories, called *tracklets*, in different arms of a beam telescope, usually separated by the DUT. This approach facilitates an unbiased estimation of the material budget at the kink of the resulting track and is also more efficient at finding particle tracks when a considerable amount of multiple Coulomb scattering is expected from the DUT.

Currently, three different track models are supported and implemented in the respective track objects described in Section 2.4. A simple *straight-line track*, described by a reference position and a direction; the *multiplet track* model, described by two tracklets and a single kink between them; and the *General Broken Line (GBL)* [27, 28] track model, which allows for kinks at every detector and takes into account multiple Coulomb scattering in all detector planes as well as the surrounding air.

3.4 Association of DUT Clusters with Reference Tracks

The association between clusters of a DUT and reference tracks built from the telescope planes is implemented as a separate module in *Corryvreckan*. This facilitates the independent treatment of multiple detectors as DUT at the same time, possibly with different association criteria and configurations. Multiple clusters from the same detector can be associated with the track, and either all of them or only the associated cluster closest in space can be retrieved from the track for analysis.

The *DUTAssociation* module provides two different ways of associating DUT clusters to the track. The first compares the cluster centre position to the track position at the DUT and assigns it based on a distance criterion, while the second option makes the association decision based on the closest distance between the track and any of the pixel hits in the cluster. The latter allows the recovery of large clusters, e.g. with contributions from delta rays, where the cluster centre is pulled far away from the track incidence position.

At this step of the reconstruction, it is possible to employ timing information to reduce mismatches between the particle track and the DUT cluster.

3.5 Prealignment & Alignment of the Setup

Corryvreckan provides the tools necessary for a two-step alignment procedure. After each of these steps, an updated geometry file in the same format as described in Section 2.3.2 is produced, which can be used as input for subsequent alignment steps as well as the final analysis. Detailed instructions on how to perform alignment, including examples, can be found in the user manual [11].

In the first step, a prealignment of all detectors is performed by calculating correlations between all detectors of the setup and the reference plane. The resulting residuals are then centred around zero by applying shifts to the individual detector positions in the global x and y coordinates. With this initial prealignment and relatively loose matching criteria it is possible to perform tracking.

These preliminary tracks form the input to the second step of alignment. Here, different modules are available, which either iteratively refit the individual tracks and minimise the sum of the track χ^2 values using Minuit2 [29], or which employ the Millepede-II algorithm [30, 31] to perform a simultaneous fit of all track candidates to determine the alignment corrections. A dedicated guide to alignment is available as part of the *Corryvreckan* user manual.

3.6 Analysis of the DUT Performance

The concept of modularity is also applied to the final analysis of detector performance figures of merit. Individual modules, such as *AnalysisEfficiency*, *AnalysisSensorEdge* or *AnalysisDUT*, are available to assess for example the efficiency or spatial resolution of each DUT in the reconstruction chain. These modules can also serve as the basis for a more customised solution for individual analyses. In addition, an *AnalysisTelescope* module is provided, which facilitates the evaluation of the reference beam telescope performance in terms of tracking efficiency and resolution at the position of the DUT.

Additional modules for specific analysis targets or dedicated to a specific detector prototype can be added to the reconstruction chain as required and users are encouraged to contribute their analysis modules to the code base to make them available to others within the community.

3.7 Import & Storage of Reconstruction Data & Results

The reconstructed detector clusters and particle trajectories can be stored in the form of *Corryvreckan* data objects as a ROOT TTree [32] at any point during the reconstruction chain using the *FileWriter* module. This data format also allows the information to be read back into the framework at a later stage to continue data processing via the *FileReader* module as shown in Section 3.1. Other output modules such as the *TextWriter*, *TreeWriterDUT* or *JSONWriter* only store a selection of the available information, or implement the conversion to different data formats.

In addition, most modules create ROOT histograms from the reconstruction process, which are stored in a central file managed by the framework. These histograms aid in gauging the quality of the reconstruction process and identifying possible problems, or serve directly as analysis plots for the detector of interest. Lists of all plots produced by each module can be found in the user manual module descriptions.

In order to facilitate the analysis of detector simulations, the Allpix^2 framework provides a dedicated *CorryvreckanWriter* module, which directly writes files in the format interpreted by the *FileReader* module of *Corryvreckan*. With this seamless integration, simulation results can be analysed with the same reconstruction chain applied to data.

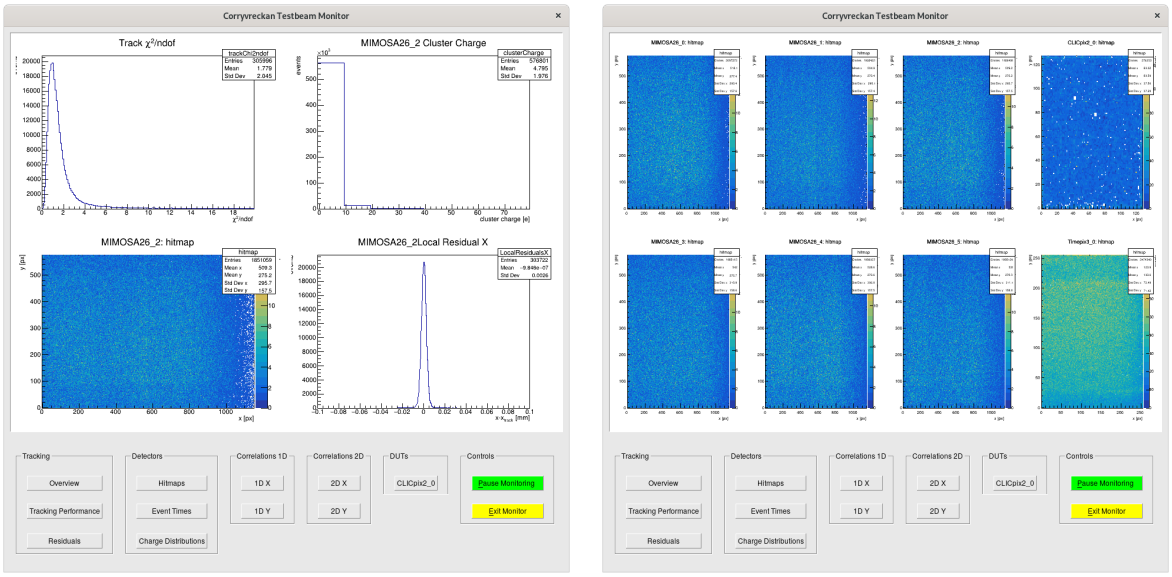


Figure 3: The *Corryvreckan OnlineMonitor* window showing the quality monitoring overview page (left) and the detector pixel hit maps (right). The page selector and control buttons are visible in the lower part of the windows.

3.8 Online Data Quality Monitoring

The *OnlineMonitor* module turns *Corryvreckan* into a quasi-online data quality monitoring tool. It provides a graphical user interface, implemented using the ROOT user interface toolkit [32], displaying a configurable set of histograms from modules of the current reconstruction run as shown exemplarily in Figure 3. Multiple pages are available for different stages of the reconstruction such as initial pixel hit maps for all detectors, or correlations and residuals from tracking. The displayed plots can be selected by the user via the regular *Corryvreckan* configuration files.

The histograms are continually updated as events are processed. This allows the *OnlineMonitor* to be used for identifying and solving issues in the reconstruction already during data-taking, but also can be used to quickly assess the quality of the data being taken in a test beam environment.

4 Offline Event Building

One of the key features of *Corryvreckan* is its flexible event building algorithm. While the reconstruction of data from a heterogeneous set of detectors with similar data and time structure is usually easy to accomplish, test beam experiments often involve the joint operation of prototypes aiming at different experiments and thus require very different readout modes. This requires an algorithm that enables data to be flexibly collected from all devices in order to successfully reconstruct particle trajectories and measure performance criteria, such as efficiencies.

This section demonstrates the capabilities of the *Corryvreckan* event building algorithm by constructing different events from the same data set for various analysis purposes. In the following, four types of detectors are distinguished:

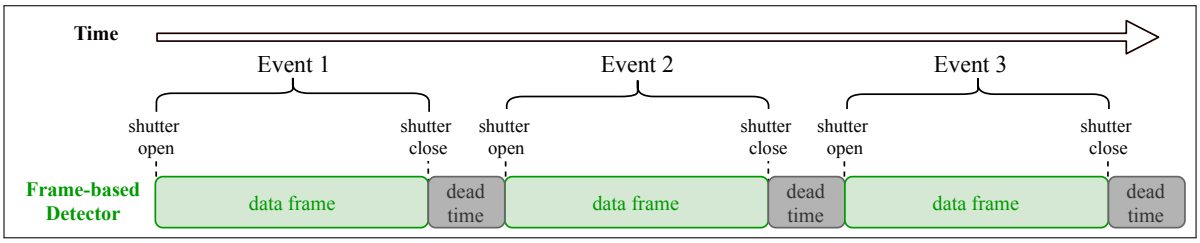


Figure 4: Schematic of the event definition provided by a device with frame-based readout scheme.

Trigger Logic Units (TLUs) provide time information and a corresponding trigger ID for the passage of particles, e.g. using the coincidence of different scintillator signals, as well as a common time reference for other detectors in the same setup. An example for such a device is the AIDA-TLU [33].

Trigger-based detectors without individual hit timestamps return blocks of data associated to an externally generated trigger received by the device. Other data, for which no trigger has been received, are discarded. In many cases, the corresponding trigger ID is the only additional information provided alongside the hit data. Examples are detectors developed for the LHC experiments or the Mimosa26 sensors used in the EUDET-type beam telescopes [34].

Frame-based detectors acquire data in a defined time interval, usually referred to as shutter. The timestamps of the beginning and end of this shutter are often recorded and stored together with the hit data.

Data-driven detectors are devices that send their hit data to the DAQ directly after registering the information without requiring an external stimulus for the readout. Therefore, these data usually contain information on hit times using a common clock signal, which is often provided by the TLU for offline synchronisation.

Depending on what information is available, different event building strategies need to be applied as described in the following sections.

4.1 The Event Building Algorithm

In *Corryvreckan*, events are formed by first defining the extent of the event in time and by subsequently adding matching data from all detectors that fit into this event. The *event definition* is performed by the first module in the reconstruction chain, and subsequent modules have to adhere to this definition. Choosing one or the other detector of the setup to take the role of defining the event enables the selection of different event boundaries, e.g. defined by the DUT to only include reference tracks within the active time of this detector.

Usually this task is performed by the *EventLoader* modules introduced in Section 3.1. The module first checks if an event has already been defined and then either adds its data within the event definition, or defines the event before loading data. Alternatively, special modules can be placed before the first *EventLoader*, allowing the event to be defined independently of a detector. Examples are the *Metronome* module, which defines successive events of equal length, or the *EventDefinitionM26* module, which correlates information from different devices.

After the event has been defined and all initial data loaded, the full reconstruction chain is executed for the current event, the resulting data is stored and the next event is processed – defined via the same algorithm.

4.2 Event Building based on Individual Detectors

The information available for the event definition depends on the detector used. For a frame-based readout detector, the time of frame start and end can be used for this definition as shown in Figure 4. All subsequent

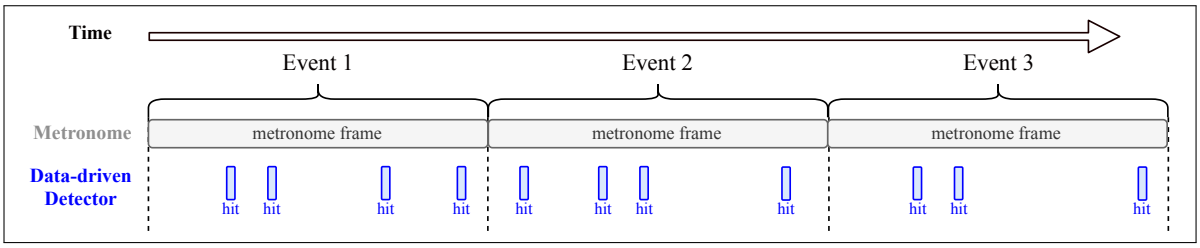


Figure 5: Schematic of the event definition as provided by the *Metronome* module, generating events of equal length in time. Data from the detector are added within the given time ranges.

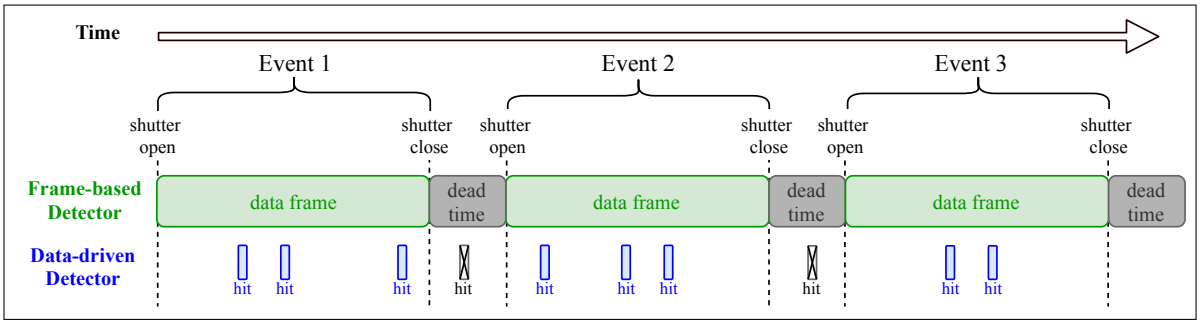


Figure 6: Schematic of the event definition using the shutter information from a frame-based device and applying the definition on the data stream of the subsequently placed detector.

data would have to lie within this defined time frame and would therefore have been recorded during the active time of this detector. By using this event building definition the efficiency of the frame-based device can be measured correctly despite possibly large dead times outside of the acquisition frame.

In contrast, for a data-driven detector the data stream does not have an inherent structure that would allow a clear separation into events. Therefore, an arbitrary time structure can be generated using the *Metronome* module as indicated in Figure 5. The module defines consecutive events with a configurable length in time such that the data stream is split into time slices of equal length. If multiple data-driven devices are present, the same definition of an event is applied to all of those devices. Ambiguities in the attribution of hits near the borders of the event-time windows can be resolved in the event building by applying additional selection criteria taking into account the hit-time resolution of the individual detectors to preserve an unbiased efficiency measurement.

Following this logic it is possible to combine detectors with different readout schemes by ordering their *EventLoader* modules sensibly in the reconstruction chain. For example, combining a frame-based with a data-driven detector could be performed by defining the event using the shutter information of the frame-based detector to partition the data of the other device as shown in Figure 6. By employing this event building scheme, hits from the data-driven device that have been recorded outside of the shutter are automatically discarded when loading data.

When building events solely from trigger-based devices, events with successive IDs can be defined using the *Metronome* module. With only trigger-based devices, no time information is necessary and data belonging to the event are identified for each detector by the event ID stored alongside the data as indicated in Figure 7. In this case, however, no additional assumption can be made about overlap between data segments and additional measures have to be used to e.g. allow for correct efficiency measurements.

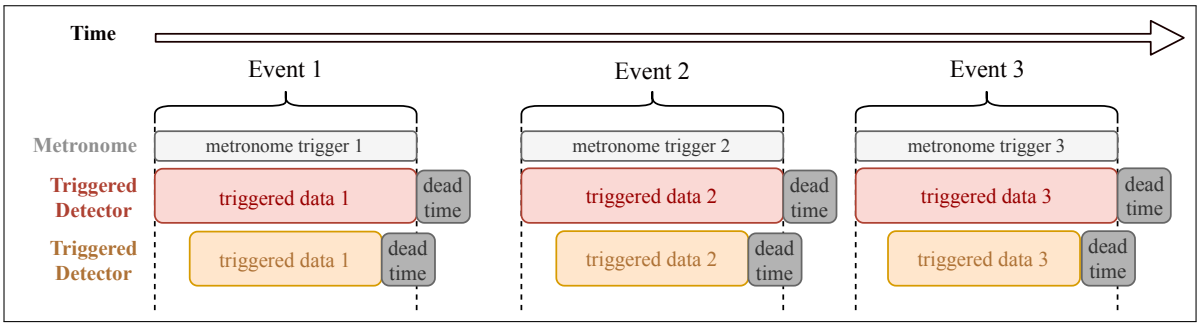


Figure 7: Schematic of the event definition via event IDs. The *Metronome* defines the event using successive event IDs, and subsequent detectors provide data for matching event IDs.

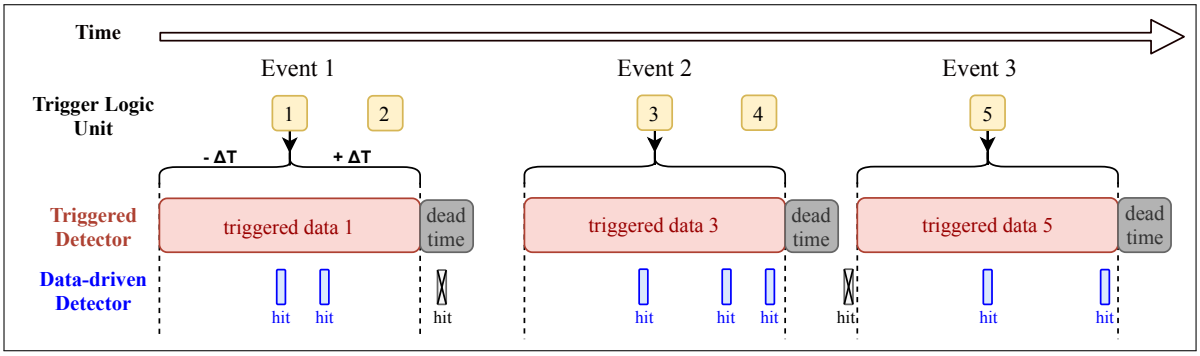


Figure 8: Schematic of the event definition by a TLU, combining trigger ID and timing information. The event is defined in time by the trigger timestamp and an optional time window indicated by $\pm\Delta T$. Additional triggers during the defined event are added to the definition with their corresponding timestamps.

4.3 Event Building Based on a Trigger Logic Unit

A coherent definition of the event becomes more difficult when detectors with and without time information have to be combined. In this case, at least one device is required that records both time and trigger information and therefore provides a relation between the other data streams. This can either be a detector that also provides hit-time information, or a TLU. The event is then defined both by a trigger ID and the corresponding timestamps. Optionally, the beginning and end of the event can be defined as a time window around the trigger decision in order to accommodate for the full integration time of the trigger-based device. An event building scheme taking this into account is illustrated in Figure 8 where the TLU defines the event and subsequent data streams are matched to the event definition either by comparing the trigger number (trigger-based detector) or the beginning and end of the event (data-driven detector).

Similar to the event building around a frame-based device shown in Figure 6, hits outside the event are discarded. The IDs of further triggers (such as trigger ID 2 in Figure 8) arriving during the active window are added to the same event (Event 1 ibid.) instead of defining a new event. This ensures that all related data from subsequent detectors are collected and avoids re-usage of the same hits for a second event.

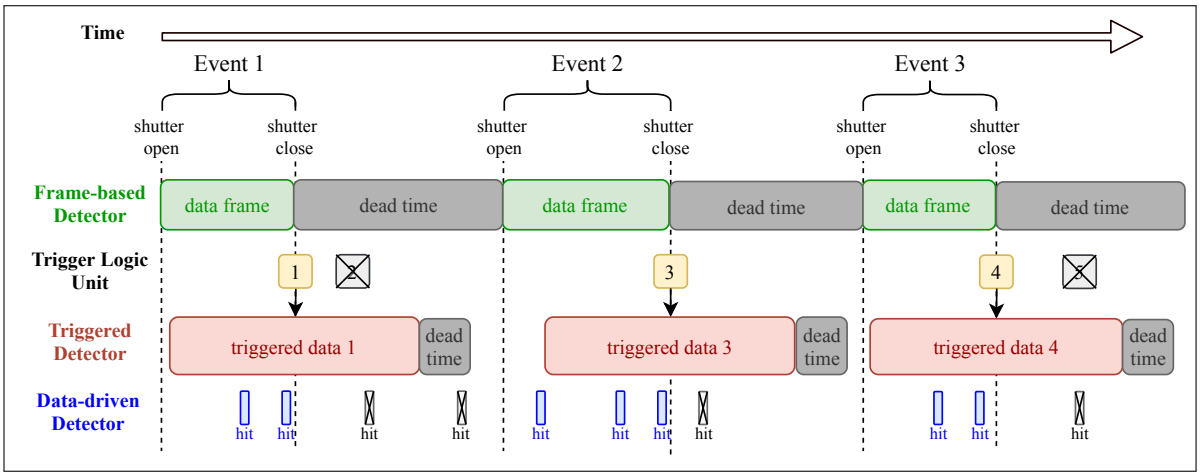


Figure 9: Schematic of the event building algorithm used for defining events based on a shutter-based device with a triggered shutter-close signal.

4.4 Event Building with a Frame-based Device with Triggered Shutter

Using the event building algorithms described above, also more complex events can be constructed from input data. Here, a frame-based device whose shutter is controlled by the trigger signal may serve as an example.

The frame-based device has been configured such that its shutter is opened immediately after the dead time of the detector readout, and is kept open until a trigger signal arrives at the detector. Such a setup can be used, for example, to synchronise the shutter with the arrival time of the particle. If this detector serves as DUT, only data from within its active shutter period should be taken into account for the reconstruction. This can be achieved by placing its *EventLoader* module topmost in the reconstruction chain, basing the event definition on its shutter timestamps as depicted in Figure 9. Next, the data stream from the TLU is used in order to add trigger IDs with timestamps within the event boundaries to the definition. Trigger IDs recorded outside the event window defined by the first module are discarded. Finally, additional data can be loaded, either based on the begin and end times of the event for data-driven detectors, or the associated trigger IDs for trigger-based detectors.

5 Reconstruction & Analysis of Frame-based, Triggered & Data-Driven Detectors

This section presents an example for a track reconstruction and analysis with *Corryvreckan*, performed on a data set recorded at the DESY II Test Beam Facility [35] with an electron beam of 5.4 GeV. The data acquisition was performed using the EUDAQ2 data acquisition software [25]. In the following, the experimental setup is introduced and the reconstruction chain is discussed in detail. Finally, a selection of performance parameters of the DUT is analysed.

5.1 Experimental Setup

The experimental setup used to record the data presented in the subsequent sections is shown in Figure 10. The following devices have been operated:

The AIDA Trigger Logic Unit [33] is used to provide a system-wide time reference as well as the discrimination and coincidence logic to generate trigger signals. Up to six scintillators equipped with PMTs can be

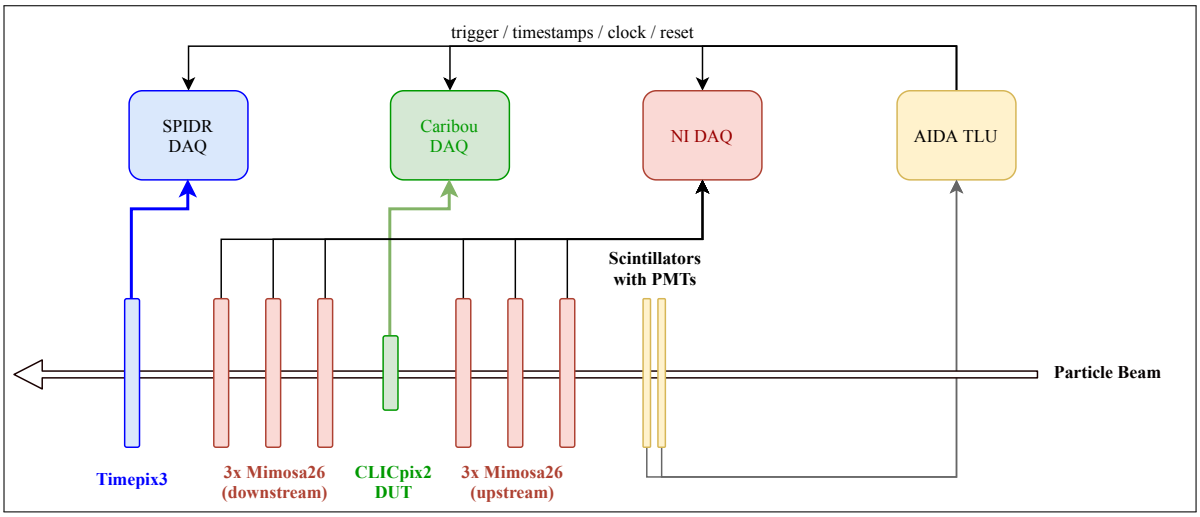


Figure 10: Schematic drawing of the experimental setup as operated at the DESY II Test Beam Facility.

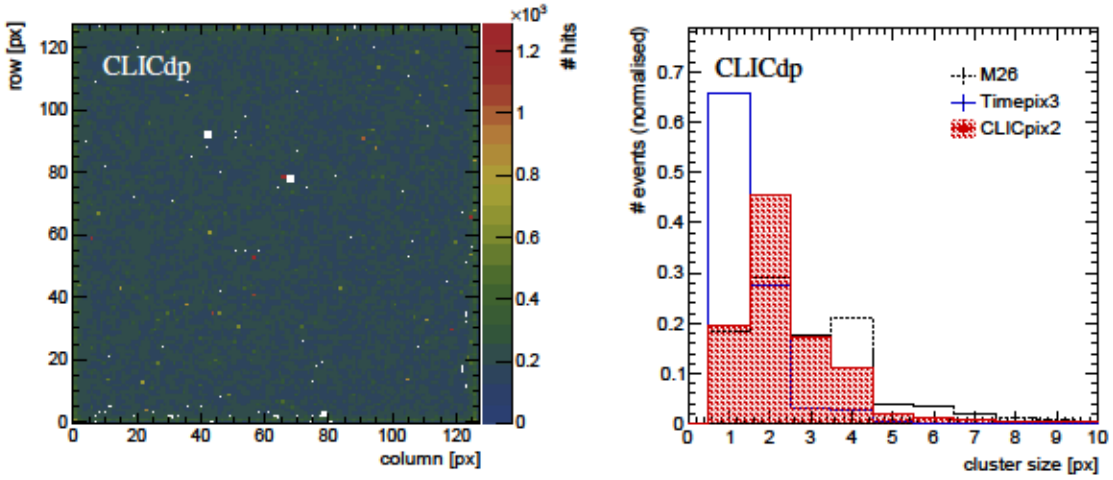
connected to the trigger inputs of the TLU, and an arbitrary coincidence pattern can be configured. For the data set presented, two scintillators mounted upstream of the beam telescope were operated in coincidence. In addition to distributing the generated trigger signal to the attached data acquisition systems, the TLU also provides a reference clock as common basis for time measurements. For this setup, a frequency of 40 MHz has been used. The clock counter is synchronised between all devices of the setup using a so-called *T0* reset signal distributed at the beginning of a run.

The DATURA telescope is a EUDET-type beam telescope [34] deployed at the beam line 21 of the DESY II Test Beam Facility. It consists of six planes of monolithic Mimosa26 sensors with a thickness of 50 µm each. The planes are grouped in an upstream and downstream arm, located before and after the DUT. The sensors are operated in a rolling shutter mode with a period of 115.2 µs, but only data frames flagged by a trigger signal from the TLU are stored to disk by the NI DAQ system. A track pointing resolution of about 2 µm at the DUT can be achieved with this telescope, depending on the geometry of the setup. Data stored from these devices contain the trigger ID distributed by the TLU.

The Timepix3 timing reference [8] is used for time-tagging of the tracks measured by the DATURA telescope within its data frames. This allows multiple particle tracks to be disentangled and to unambiguously select those that arrived during the active window of the DUT. The Timepix3 detector is a hybrid pixel detector with a 100 µm thick n-in-p planar silicon sensor, a pixel pitch of 55 µm × 55 µm with 1.56 ns binned timestamps and a 10-bit charge measurement. It is operated in a data-driven readout mode providing a data stream of pixel hits with timing information, which is received and processed by the SPIDR DAQ system [36].

The detector was placed downstream of the telescope and is synchronised with the TLU via the common reference clock and the *T0* signal.

The CLICpix2 prototype [37] is operated as device-under-test. CLICpix2 is a readout chip designed as a technology demonstrator for the vertex detector of an experiment at the linear lepton collider CLIC. It features a 128 × 128 pixel matrix with square pixels of 25 µm pitch and is operated in a frame-based scheme with external signals controlling the opening and closing of the shutter. The front-end is capable of simultaneously recording 5-bit Time-over-Threshold (ToT) and 8-bit Time-of-Arrival (ToA) information for each pixel with a time binning



(a) Pixel hit map of the CLICpix2 prototype. Masked pixels appear white and pixels with increased occupancy in yellow.

(b) Cluster size distributions: Mimosa26 (M26, dashed black line), Timepix3 (solid blue line) and CLICpix2 (red hatched area).

Figure 11: Pixel hit map of CLICpix2 (a) and comparison of the cluster size distribution for the different detectors (b).

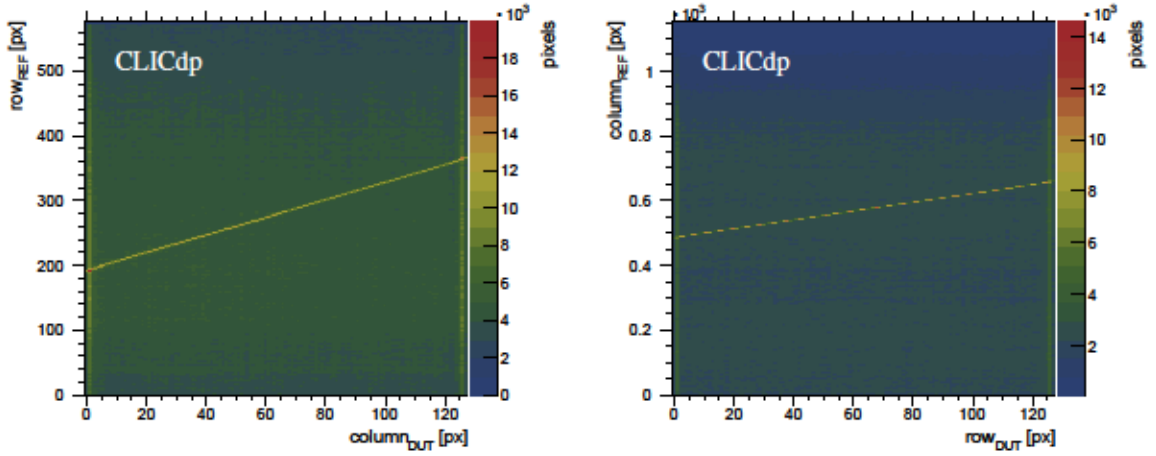
of 10 ns. The assembly used as DUT in the test beam campaign presented consists of a CLICpix2 readout chip bump-bonded to a planar silicon sensor with a thickness of 130 µm. The DUT was operated at a threshold of around 870 electrons, with a bias voltage of −25 V applied. For this study, the chip was configured to record data in ToT and counting mode, i.e. replacing the ToA measurement with a particle counter for each pixel. Therefore, no pixel-level timing information is available for this device. However, the timestamps of the shutter opening and closing are recorded using the reference clock and *T0* signal from the TLU, allowing the determination of the active window of the detector. CLICpix2 is operated and read out by the Caribou DAQ system [38].

5.2 Reconstruction Chain

In this section, the reconstruction chain for the test beam data is described in detail following the flow chart presented in Figure 2, from data decoding and event building to tracking and alignment.

Event building and raw data processing The event building is based on the shutter opening and closing time of the CLICpix2 prototype detector. Since all detectors have been operated using the EUDAQ2 DAQ framework, the *EventLoaderEUDAQ2* module is used to read and process the raw data. It uses the converter plugins implemented directly in the DAQ framework in order to decode the raw detector data to pixel hit information with column, row, charge, and time measurement where applicable.

These decoded data are placed on the *Corryvreckan* clipboard in the order of appearance of the respective *EventLoader* modules in the *Corryvreckan* main configuration file, i.e. CLICpix2, TLU, Mimosa26, and Timepix3. Here, it is important that TLU data is processed before Mimosa26 data because it adds the relevant trigger IDs recorded between shutter open and close signals of the CLICpix2 detector to the event. These trigger IDs are then used to assign the appropriate Mimosa26 frames to the data block of the event. The TLU trigger timestamp is assigned to pixel hits received from Mimosa26 sensors.



(a) Cross-correlations of the DUT x axis

(b) Cross-correlations of the DUT y axis

Figure 12: Cross-correlations between CLICpix2 (DUT) and the last upstream telescope plane (REF) in x (a) and y (b). The two detectors are rotated by 90° with respect to each other, which makes it necessary to compare opposite coordinates (columns versus rows). These plots are provided by the *Correlations* module.

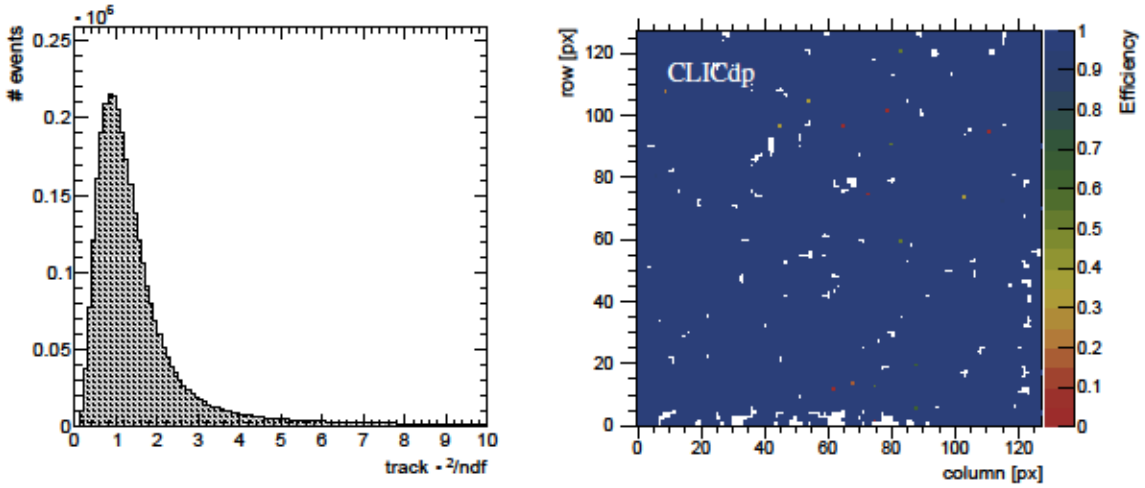
Calibration of the pixel charge and timestamps is taken care of by the corresponding EUDAQ2 converter plugins, the relevant calibration data can be supplied either through the *Corryvreckan* geometry description or as configuration parameter of the event loader module. Pixels can be masked offline by providing a mask file to the framework, and hits from masked pixels are filtered out before being stored on the clipboard and therefore appear empty in hit maps. Mask files can either be populated manually or by using the *MaskCreator* module, which masks pixels based on their firing frequency with respect to either the total chip frequency or a local density estimate. A hit map for the CLICpix2 DUT is shown in Figure 11a, which shows that not all pixels with increased occupancy have been filtered out. The level of acceptable noise hits and therefore the selected masking criterion depends on the individual application situation.

Clustering The clustering of pixel hits stemming from the same particle is performed using the *Clustering4D* module, which takes into account the time information assigned to the individual pixel hits. Adjacent pixel hits are grouped into clusters if their timestamps are within a time window defined in the *Corryvreckan* configuration for each detector individually, taking into account effects such as time walk.

A comparison of the cluster size distributions of the different detectors is shown in Figure 11b. While Timepix3 predominantly generates single-pixel clusters owing to its comparatively large pixel pitch, the thin sensor, and the perpendicular particle incidence, Mimosa26 and CLICpix2 produce significantly larger clusters. In Mimosa26 the reason for this is the signal collection relying on diffusion processes, while CLICpix2 has an increased charge sharing between pixels due to the pixel pitch of 25 µm in combination with the sensor thickness of 130 µm.

The *EtaCorrection* module is used in order to perform an η-correction for non-linear charge sharing of the cluster position for the CLICpix2 DUT.

Correlations Correlation plots are an important means to gauge data quality, synchronisation between devices and mechanical alignment of the different detector planes. Figure 12 shows two cross-correlation plots produced by the *Correlations* module. Here, pixel hit positions are compared between the CLICpix2 DUT and the



(a) Track χ^2 divided by the number of degrees of freedom, provided by the *Tracking4D* module.

(b) Efficiency as a function of the track position over the full chip area, masked areas appear white.

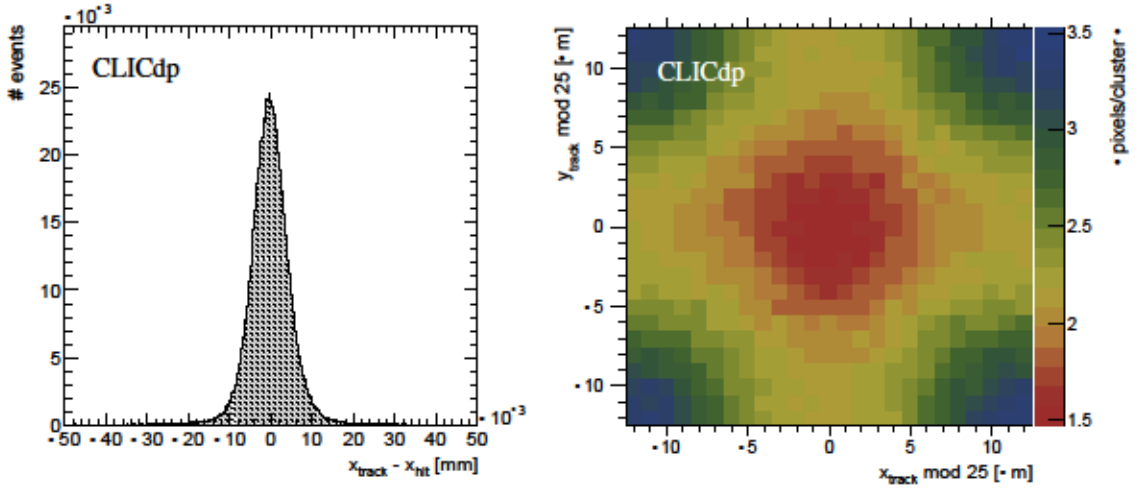
Figure 13: General Broken Lines track χ^2 as measure for the goodness of fit (a) and CLICpix2 efficiency map (b).

last upstream Mimosa26 plane of the beam telescope, which has been marked as reference detector in the *Corryvreckan* geometry description. Shown are the hit column address of one detector versus the hit row address of the other detector and vice versa since the two planes are rotated by 90° with respect to each other. A clear correlation is visible as bright line above a background of uncorrelated hits stemming from residual noise contributions as well as hits of the Mimosa26 plane outside of the DUT acceptance. The line does not coincide with the diagonal of the histogram since the two detectors are different in size, with the CLICpix2 occupying an area of only 3.2 mm × 3.2 mm in the centre of the 20 mm × 10 mm large Mimosa26 sensors.

Projections of these correlation diagrams along the diagonal are used in the prealignment in order to correct the relative position of detectors with respect to the reference plane.

Tracking & telescope alignment Owing to the beam energy of 5.4 GeV used at the DESY II Test Beam Facility and the expected multiple scattering along the particle trajectory, General Broken Lines has been selected as track model for the *Tracking4D* module. The module is configured to require hits from at least six out of seven detector planes available for tracking, while always requiring a hit from the Timepix3 time reference plane. Subsequently, the cluster timestamp from the Timepix3 data is assigned as track timestamp.

The DATURA telescope planes and the Timepix3 detector are aligned using the iterative approach minimising the global χ^2 value of the track fit implemented in the *AlignmentTrackChi2* module. Alignment is performed in the x and y position and all three orientation degrees of freedom. The procedure of refitting tracks and optimising the detector placements and orientation is repeated until a sub-micron alignment precision is reached. The z position constitutes a weak mode and was fixed to the values measured in the beam area. The χ^2 distribution over the degrees of freedom resulting from the track fits with the final alignment is shown in Figure 13a with a peak close to one, indicating a good fit of the track to the detector data.



(a) Unbiased residual distribution of the x axis for CLICpix2, provided by the *AnalysisDUT* module.

(b) Mean cluster size as function of the track impact position within a pixel cell.

Figure 14: Unbiased residual distribution of CLICpix2 for clusters of all sizes (a) and mean cluster size (b) at vertical track incidence within a single pixel cell of the CLICpix2 prototype.

5.3 Analysis of the DUT Performance

After tracking and alignment of the detectors of the reference beam telescope have been performed, the DUT data can be analysed in a separate run. According to the modular structure of the framework, the analysis of the various performance parameters is also divided into individual modules. This facilitates the flexible analysis of different prototypes and to adjust the configuration parameters. In this example, the CLICpix2 data are first assigned to the tracks reconstructed from the reference beam telescope. Afterwards, the efficiency and the spatial resolution are evaluated. For the measurement of these observables, only telescope tracks with $\chi^2/\text{ndof} < 3$ have been selected within the respective analysis modules.

It should be noted that this section does not present a detailed analysis of the DUT but should serve only as an example for the capabilities and functionality of the *Corryvreckan* framework. More detailed studies of the CLICpix2 prototype, using laboratory and test beam data, can be found elsewhere [18, 19].

DUT cluster association to tracks DUT clusters are assigned to telescope tracks using the *DUTAssociation* module with the *closest distance* matching criterion comparing the cluster edge to the track position. The maximum matching distance is chosen as one pixel pitch in either direction, i.e. 25 µm.

Efficiency The efficiency of the DUT is calculated using the *AnalysisEfficiency* module as the number of tracks with an associated DUT cluster divided by the total number of tracks passing through the detector. Removed from this selection are tracks that penetrate the detector in areas with masked pixels with a one-pixel tolerance, as well as the outermost rows and columns of pixels close to the sensor edge. Apart from these masked areas, the prototype exposes a very uniform efficiency as shown in Figure 13b, with only a few pixels that are less efficient. The overall efficiency for the data analysed is evaluated to be above 99.97 %.

Position resolution Finally, the position resolution of the detector can be determined via the *AnalysisDUT* module by quadratically subtracting the telescope track resolution from the width of the residual distribution

presented in Figure 14a. Here, the residual width is defined as the root mean square of the truncated distribution containing the central 96 % of the statistics, i.e. $\pm 2\sigma$, and evaluates to $\sigma_{res} = 4.1\,\mu\text{m}$.

Using the high pointing resolution of the reference tracks, the *Corryvreckan* analysis modules facilitate the evaluation of different quantities as a function of the track incidence position within a single pixel cell of $25\,\mu\text{m} \times 25\,\mu\text{m}$ of the CLICpix2 DUT. As an example, Figure 14b show the average cluster size as a function of the track impact position within the pixel cell, which is a quantity used to gauge charge sharing between neighbouring pixel cells. Single-pixel clusters occur almost exclusively in a clearly defined area in the centre of the pixel cell, since strong charge sharing effects arise from the ratio of pixel pitch and sensor thickness. All tracks incident outside this area will form multi-pixel clusters by means of charge sharing.

6 Conclusions & Outlook

In this paper, the test beam data reconstruction framework *Corryvreckan* has been presented. It is a modular framework that enables the correlation of detectors with different readout schemes through its flexible event building algorithm. Through a direct interface to the EUDAQ data acquisition system it is capable of handling any detector with the necessary conversion plugins available through the DAQ.

The reconstruction chain is built from individual modules, each performing a specific task or implementing a single algorithm. A range of modules has been presented briefly, and the event building process has been described in detail using different application scenarios. This includes setups with detectors using the same synchronisation method, like triggers or common timestamps, as well as a combination of these devices or advanced configurations using triggered shutters. The joint operation of three different detectors and a Trigger Logic Unit was used as an example demonstrating the reconstruction and analysis of data recorded at the DESY II Test Beam Facility.

Several extensions of the *Corryvreckan* framework as well as additional modules and track models are already under development in order to further extend the functionality and to serve an even wider community. This includes features such as tracking in magnetic fields or the reconstruction of data from detectors with radial geometries, as well as parallel processing of data on multi-core machines.

Acknowledgements

This work was carried out in the framework of the CLICdp Collaboration. The measurements leading to these results have been performed at the DESY II Test Beam Facility at DESY Hamburg (Germany), a member of the Helmholtz Association (HGF). This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168. We would like to thank Matthew Buckland, Manuel Colocci, Alexander Ferk, Adrian Fiergolski, Magnus Mager, Andreas Nürnberg, Mateus Vicente Barreto Pinto and Jin Zhang who have provided valuable feedback and contributed to the *Corryvreckan* software.

References

[1] T. Bisanz et al., *EUTelescope: A modular reconstruction framework for beam telescope data*, *J. Instr.* **15** (2020) P09020.

[2] G. McGoldrick, M. Červ and A. Gorišek, *Synchronized analysis of testbeam data with the Judith software*, *Nucl. Instr. Meth. Phys. A* **765** (2014) 140.

[3] M. Kiehn, *Proteus beam telescope reconstruction*, *Zenodo* **v1.4.0** (2019) .

[4] C. Hombach, T. Evans and H. Schindler, "Kepler: Timepix3 test beam offline analysis software based on the Gaudi framework." https://gitlab.cern.ch/lhcb/Kepler.

[5] G. Apollinari et al., *High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report*. CERN Yellow Reports: Monographs. CERN, Geneva, 2015, 10.5170/CERN-2015-005.

[6] CLICdp & CLIC collaboration, T. K. Charles et al., *The Compact Linear Collider (CLIC) - 2018 Summary Report*, *CERN Yellow Rep. Monogr.* **1802** (2018) 1, [1812.06018].

[7] T. Behnke et al., *The International Linear Collider Technical Design Report - Volume 1: Executive Summary*, 1306.6327.

[8] T. Poikela et al., *Timepix3: a 65k channel hybrid pixel readout chip with simultaneous toa/tot and sparse readout*, *JINST* **9** (2014) C05013.

[9] The Corryvreckan Authors, "Corryvreckan - A Modular 4D Track Reconstruction and Analysis Software for Test Beam Data (Version 2.0)." Zenodo, Dec., 2020. 10.5281/zenodo.4384186.

[10] J. Kröger, S. Spannagel and M. Williams, *User Manual for the Corryvreckan Test Beam Data Reconstruction Framework, Version 1.0*, 1912.00856.

[11] J. Kröger, S. Spannagel and M. Williams, "Corryvreckan User Manual, latest version." http://cern.ch/go/db9Z.

[12] S. Spannagel et al., Allpix^2*: A modular simulation framework for silicon detectors*, *Nucl. Instr. Meth. Phys. A* **901** (2018) 164, [1806.05813].

[13] F. Pitters et al., *Time resolution studies of timepix3 assemblies with thin silicon pixel sensors*, *J. Instr.* **14** (may, 2019) P05022.

[14] R. Bugiel et al., *High spatial resolution monolithic pixel detector in SOI technology*, Tech. Rep. CLICdp-Pub-2020-004, CERN, Geneva, Aug, 2020.

[15] K. Dort, *Silicon vertex and tracking detector R&D for CLIC*, Tech. Rep. CLICdp-Conf-2020-006, CERN, Geneva, Oct, 2020.

[16] J. Kröger, *Silicon pixel sensor R&D for the CLIC tracking detector*, *J Instr.* **15** (aug, 2020) C08005.

[17] M. Munker, *Vertex and tracking detector R&D for clic*, *Nucl. Instr. Meth. Phys. A* **980** (2020) 164475.

[18] M. Williams, *R&D for the CLIC vertex and tracking detectors*, *J Instr.* **15** (mar, 2020) C03045.

[19] M. Williams, *Evaluation of Fine-Pitch Hybrid Silicon Pixel Detector Prototypes for the CLIC Vertex Detector in Laboratory and Test-Beam Measurements*. PhD thesis, University of Glasgow, to appear.

[20] T. Quast, *Qualification, Performance Validation and Fast Generative Modelling of Beam Test Calorimeter Prototypes for the CMS Calorimeter Endcap Upgrade*. PhD thesis, RWTH Aachen University, Jul, 2020.

[21] International Organization for Standardization, *Information technology – Programming languages – C++*, ISO/IEC 14882:2014, Geneva, Switzerland, 2014.

[22] D. van Heesch, "Doxygen - generate documentation from source code." https://www.stack.nl/~dimitri/doxygen/.

[23] The Corryvreckan Authors, "Corryvreckan project website." https://cern.ch/corryvreckan.

[24] P. Ahlburg et al., *EUDAQ – a data acquisition software framework for common beam telescopes*, *J. Instr.* **15** (2020) P01038.

[25] Y. Liu et al., *EUDAQ2 – A flexible data acquisition software framework for common test beams*, *JINST* **14** (2019) P10033, [1907.10600].

[26] K. Akiba et al., *Charged particle tracking with the Timepix ASIC*, *Nucl. Instr. Meth. Phys. A* **661** (2012) 31.

[27] V. Blobel, C. Kleinwort and F. Meier, *Fast alignment of a complex tracking detector using advanced track models*, *Comput. Phys. Commun.* **182** (2011) 1760.

[28] C. Kleinwort, *General broken lines as advanced track fitting method*, *Nucl. Instr. Meth. Phys. A* **673** (2012) 107.

[29] F. James and M. Roos, *Minuit - a system for function minimization and analysis of the parameter errors and correlations*, *Comput. Phys. Commun.* **10** (1975) 343.

[30] V. Blobel and C. Kleinwort, *A New method for the high precision alignment of track detectors*, in *Proceedings of Advanced Statistical Techniques in Particle Physics*, (Durham, UK), 2002. hep-ex/0208021.

[31] V. Blobel, *Software alignment for tracking detectors*, *Nucl. Instr. Meth. Phys. A* **566** (2006) 5.

[32] R. Brun and F. Rademakers, *ROOT – an object oriented data analysis framework*, *Nucl. Instr. Meth. A* **389** (1997) 81 – 86.

[33] P. Baesso, D. Cussans and J. Goldstein, *The AIDA-2020 TLU: a flexible trigger logic unit for test beam facilities*, *JINST* **14** (2019) P09019.

[34] H. Jansen et al., *Performance of the EUDET-type beam telescopes*, *EPJ Tech. Instrum.* **3** (2016) 7, [1603.09669].

[35] R. Diener et al., *The DESY II test beam facility*, *Nucl. Instr. Meth. Phys. A* **922** (April, 2019) 265.

[36] J. Visser et al., *SPIDR: a read-out system for Medipix3 and Timepix3*, *JINST* **10** (2015) C12028.

[37] E. Santin, P. Valerio and A. Fiergolski, "Clicpix2 user's manual." https://edms.cern.ch/document/1800546/1, 2017.

[38] T. Vanat, *Caribou – A versatile data acquisition system*, *PoS* **TWEPP2019** (Dec, 2019) 100. 5 p.