Karlsruhe Institute of Technology

Title

Title

Master thesis
submitted by

Nils Braun

DATUM

Institut of Experimental Nuclear Physics (IEKP)

Advisor:	Prof. Dr. Michael Feindt
Coadvisor:	Prof. Dr. Ulrich Husemann

Editing time: November 2014 – November 2015

Karlsruhe Institute of Technology

Title

Title

Masterarbeit
eingereicht von

Nils Braun

DATUM

Institut für experimentelle Kernphysik (IEKP)

Referent:	Prof. Dr. Michael Feindt
Korreferent:	Prof. Dr. Ulrich Husemann

Bearbeitungszeit: November 2014 – November 2015

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, DATUM

. . . . . .

(**Nils Braun**)

Masterarbeit angenommen.

Karlsruhe

.....

(**Prof. Dr. Michael Feindt**)

Contents

1. Introduction

1.1. Belle II and the High Luminosity Frontier

Standard Model, Belle discovery, CKM, how Belle II will help, new Physics Working principle of detectors

1.2. Track Finder for Particle Physics Experiments

The purpose of track finding and why it is that important to be as good as possible Possible inefficiencies Background

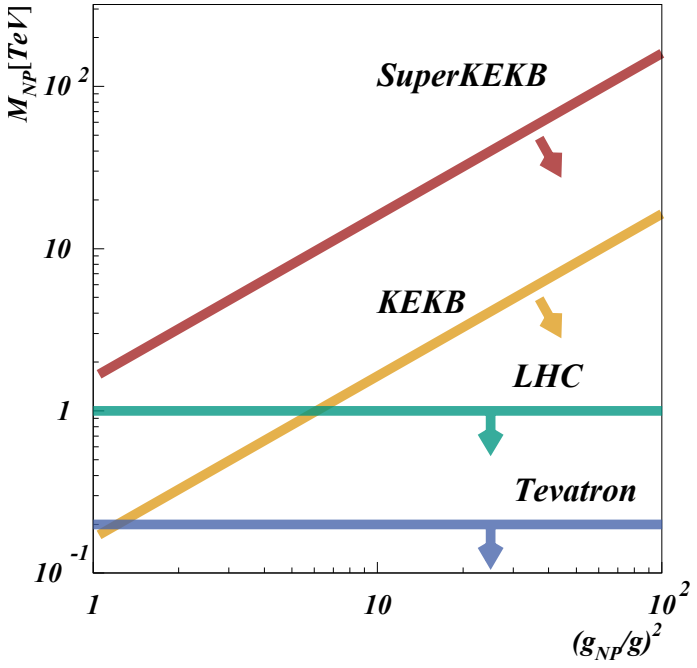

M_{NP}[TeV]
10^2
10
1
10^{-1}
SuperKEKB
KEKB
LHC
Tevatron
1
10
10^2
$(g_{NP}/g)^2$

Figure 1.1.: [1]

2. Experimental Setup

In the following sections the experimental setup for this thesis - the Belle II detector and the SuperKEKB collider - are described as far as it is necessary for the tracking software. Detailed information can be found elsewhere [1].

2.1. SuperKEKB and the Belle II Detector

The future Belle II is currently being build at the KEK high energy research facility in Tsukuba, Japan. It is a general purpose 4π detector for high energy particle experiments. A scheme of the whole detector with the planned measuring devices can be found in figure 2.1.

The Belle II detector is used for measuring the electron positron collisions induced by the SuperKEKB collider. As its predecessor KEKB its center of mass energy is tuned in such a way that it lays near the $\Upsilon(4S)$ resonance. This beam energy is chosen that way because a $\Upsilon(4S)$ particle decays in nearly all cases into a pair of B-mesons. The daughter particles of these B mesons open a window to a very wide window of physical studies like CP violation or B-meson spectroscopy. The energy of the colliding electrons an positrons is asymmetric which leads to a slightly asymmetric collision as well. This fact is exploited in measuring the decay length of the two formed B-Mesons - one of the main ingredients for quantifying the CP violation in the B-Meson system. The usage of electrons and their antiparticles for collisions leads to very clean events compared to other collision partners like protons or even gold atoms. Clean means in this context a reduced number of created decay products (only 11 charged tracks on average) and more or less no pile up (only one collision occurring at the same time) are expected. These circumstances can be used for high precision measurements on the formed particles.

2.2. The Tracking Detectors

As every general purpose high energy particle physics detector the Belle II detector is composed of several single measuring devices - each one with a different task. For collecting as many information on the physical process in the collision as many infromation on the created decay products are needed. These include

(1) Identification of the created particles (for example by their mass),

Figure 2.1.: Scheme showing the planned Belle II detector in top view. The single measuring devices are shown with their names. Some more information on the tracking devices can be found in the text. Taken from [2].

(2) Measurement of the kinematic properties (like momentum and energy) of the charged particles,

(3) Resolution of their origin position - also called their vertex.

The tracking detectors described in this thesis are mainly for recording the momenta of the particles as well as their vertex position. Further more special applications are identification of the particle type by their energy loss.

The measuring principle of the momentum and also the vertex is to let the charge particles fly throw a sensor region with space-resolved measurement and obtain their full path through the detector - a so called track - by combining the different location information - the so called hits. By applying a magnetic field - in Belle II it has a nominal field strength of $B = 1.5\,\mathrm{T}$ parallel to the beam axis the charged particle are curled. By measuring the curvature of the curled tracks an estimation of the momentum in the layer perpendicular to the beam axis can be made. The remaining direction parallel to the beam axis can be computed with the angle of the track to the beam axis.

There exist many different ways to measure the position of a charge particle. Each of them exploits the charge of the particles and their interaction with materials which makes these techniques unusable for neutral particles. In the following two of the used ways in Belle II are described in more detail.

2.2.1. CDC

CDC is an acronym for Central Drift Chamber and is the largest tracking detector in Belle 2. The CDC is a typical gas chamber detector with 14336 sense wires arranged in 56 layer and 9 superlayers. These wires are strained mostly parallel to the beam axis in a cylindrical tank flooded with gas. The gas is a mixture of 50 % helium and 50 % ethane. Each charged particle produced in the collision of the electron and position interacts with the gas and can ionize the helium atoms. The energy deposit of a charged particle is given by the Bethe formula [3]

$$-\left\langle \frac{\mathrm{d}E}{\mathrm{d}x} \right\rangle = \frac{4\pi n z^2}{m_e c^2 \beta^2} \cdot \left(\frac{e^2}{4\pi\varepsilon_0} \right)^2 \left[\ln\left(\frac{2 m_e c^2 \beta^2}{I(1-\beta^2)} \right) - \beta^2 \right] \tag{2.1}$$

which describes the average energy per travel length. This energy is transfered to the shell electrons of the helium gas atoms. As the ionization energy of helium is very low (about 20 eV) compared to the typical particle energies (up to several 100 eV) the helium atoms release their electrons very easily. These free electrons get accelerated in the electric field induced between the sense wires and field wires with a high negative voltage attached. An instructive simulation of the typical electronic field distribution between 7 sense wires and 34 field wires can be found in figure 2.2. As the electric field near the sense wires increases with $\propto 1/r$ the free electrons gain more and more energy so that they can ionize helium atoms by themselves as well. This whole flood of free charge can be detected by the electronics attaches to the sense wires and produce a sharp peak there. The more immobile helium atomic cores get absorbed later by the field wires. Because the free electrons have to drift through the gas the measured peak has a delay up to 500 ns for the largest drift cells to the initial charged particle passage. With analysis the time and location information of each measured peak the path of the charged particle through the CDC can be reconstructed. A typical event display of the simulated passage of one charge pion with MOMENTUM can be seen in figure **??**. Each wire and each drift cell is almost rotationally symmetric. Therefore there is no way in obtaining information on the direction of the electron flood hitting the sense wire. Therefore each hit produced a so called drift circle rather than a single hit point. These drift circled can also be seen in the figure.

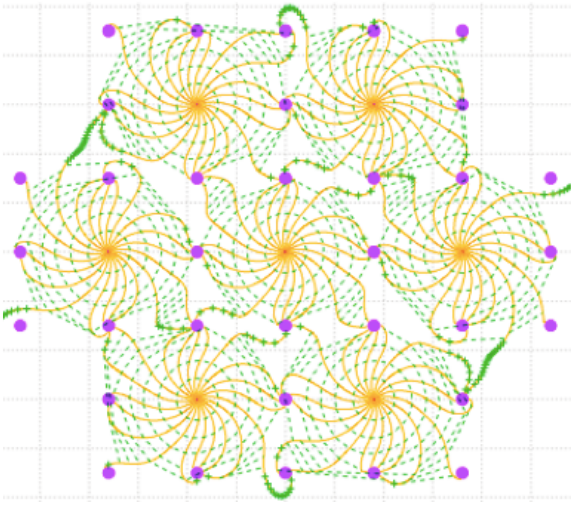


Figure 2.2: Simulation of the electron field in an extract of the wires in the central drift chamber. The violet circles depict the field wires whereas the red small points are the sense wires. The yellow paths are examples for the drift paths of ionized electrons. This simulation includes the distortion by the magnetic field. Taken from [4].

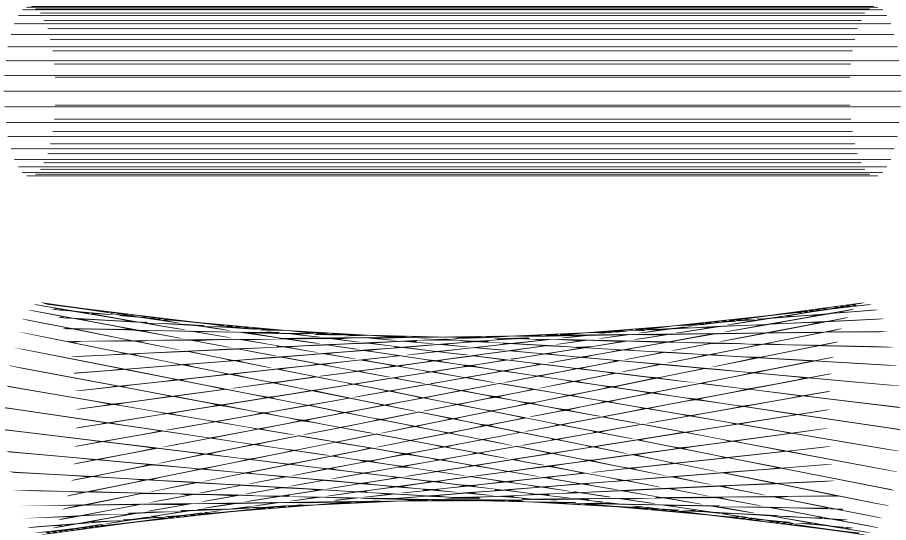


Figure 2.3.: Drawing sketching the axial (top) and stereo wires in the CDC. The skewing against the beamline of the stereo (bottom) wires is exaggerated. Taken from [5].

picture

If all CDC wires were strained parallel to the beam axis there were no way in measuring the angle between the charged particles and the beam axis. That is the reason why some of the wires are installed with a small tilting angle. These wires are called stereo wires in contrast to the axial wires without tilting angle. A measured hit on the stereo wires does not give information neither on the position along the beam axis nor in the layer parallel to it. Only the information collected among all axial and stereo hits lead to a correct estimation of the momentum in all directions. A sketch of some axial and stereo wires sharing the same distance from the beam axis - also called a layer of wires - are shown in 2.3.

2.2.2. VXD

3. Track Finder Theory and Multivariate Classification

Before explaining the implemented changes to the track finder for the Belle II experiment in more detail, the principles of the Belle II software framework are explained briefly. More information can be found elsewere

quote

. Afterwards common figures of merit for all track finders are explained and discussed and the working principles of the already implemented track finders are illustrated.

3.1. The Belle II Analysis Framework (`basf2`)

For simulation, data acquisition, data processing and analysis of the Belle II experiment the Belle Analysis Software Framework 2 (`basf2`) is used. Although - guided by its name - it seems to be build on top of the old software framework used for the Belle experiment it is a complete rewrite of the software using modern programming principles in the coding languages C++ [**?**] and Python 2.7 [6]. Together with external programming libraries like ROOT [7] or EvtGen [8] that are already on the market this framework builds the base for every software written for the experiment.

The software is divided into several packages - each serving a single purpose or summarizing code for a single detector. Examples for the packages are CDC, SVD or the tracking packaged which is described in more detail in later chapters.

Each usage of the Belle Analysis Software Framework - if it is either a simulation, a reconstruction or an analysis does not matter - consists of processing one or more so called *paths* build with *modules*. These modules perform a dedicated small task like simulating the hard scattering event (the so called `EvtGen` module), writing out data to a root file (the module is called `RootOutput`) or performing a track reconstruction (for example with the module `TrackFinderCDCAutomaton`). The presence, the order and the parameters of the modules are determined in *steering* files written with python. The modules itself can be written in C++ or python.

In these steering files a path is created, filled and passed to the framework which handles loading the corresponding C++ libraries and calling the modules for every event that should be processed. An example of a small steering file for track finding can be found in listing **??**.

listing

Caused by this extremely modular structure not only parallel processing but also debugging of intermediate steps can be performed much easier.

Because many modules need the data produces by other modules before there is a need for intermodular communication. This communication is performed within the framework with the help of the data store. This class as a wrapper around a collection of named `TClonesArrays` from the ROOT library

quote

which can store lists of instances of nearly arbitrary C++ classes. It is used widely in the framework to store all sorts of things like the hit information produced by the particles in the simulation or the found tracks after the track finding modules. The modules have read and write access to every so called store array in the data store. A visualization of the data flow between the modules created with the steering file in **??** can be found in figure **??**. The data store can be written to or read from disk using ROOTs own serialization mechanism together with data member dictionaries for the C++ classes created by the C++ interpreter of ROOT called CINT.

3.2. Working Principle of the implemented Track Finder in `basf2`

Global vs Local Hit and track is already explained - just explain stiching of hit here!

3.2.1. The Legendre Track Finder

Legendre Principle, QuadTree, Stereo

3.2.2. The Local Track Finder

Clusterizer, Automaton-Principle

3.3. The used Figures Of Merit

For testing and developing and also for later usage in the experiment setup we need to compile numbers from the implemented track finder algorithms to show how well they work. There are three different classes of figures of merit to describe a track finder. All three classes listed here are described in more detail below. The three classes are:

- the efficiency (like the hit efficiency or the finding efficiency, also split up for different particle types, momentum regions or areas in the detector)
- the error rate (like the clone or fake rate)
- the computational performance (like timing and memory consumption)

The last class should be quite clear and is measured by the basf2-own measuring algorithms. As the tracking is part of the online reconstruction and may be once adopted for the high level trigger, the timing performance is very important.

The other two classes can best be described with the algorithm how they get computed. It starts with a full Monte Carlo simulation of generic BB events with the full detector simulation afterwards. The created hits can then be parted into distinct sets - each describing a simulated tracks, called MCTrackCand (they are called track candidates to fit the naming convention of the track finder). Only those tracks are kept as a MCTrackCand, that have at least 3 hits in the CDC - otherwise they can not be fitted and even if there were a chance to find them via track finding, they could not be used for physics. After that, the simulated hits with the stripped MC information are used for track finding with

the method to be evaluated. This can be done easily as the simulated hits are transformed in a format that is similar to the format that will be used later for the data coming from the experiment. After the track finder has produced a list of track candidates also, we can use the saved MC information to match tracks from the track finder to tracks from the mc algorithm (also called MC track finder) by counting the number of hits they share. The different cases are depicted and described in table 3.1.

The finding efficiency now describes the rate of MCTrackCands which are labeled matched by their total amount. Building this ratio can also be done for bins in various variables, like perpendicular momentum (p_T), angle in the curling plane (phi), number of tracks per event (multiplicity) and many more. A perfect track finder would have a finding efficiency of 100 %. In most of the cases, the finding efficiency drops for tracks in a certain region of these variables - like for low momentum tracks. The hit efficiency is the mean of the ratios between the number of hits in the MCTrackCands matched to a non-fake track candidate to the number of hits in total to this track. Here also a perfect track finder would have a hit efficiency of 100 %. As in most of the cases the track finder looses some outlaying hits the hit efficiency drops. The fake and clone rates are just the number of track candidates labeled as fake or clone by the matching algorithm divided by the number of found tracks in total. A perfect track finder would have both number set to 0 %. A high fake rate is caused by a track finder with too loose cuts when putting together single pieces of tracks to a big track or by one which picks up background hits often. A track finder with a high clone rate on the other hand has to harsh cuts and splits up tracks into more than one piece.

3.4. Multivariate Classification

Classification, BDTs

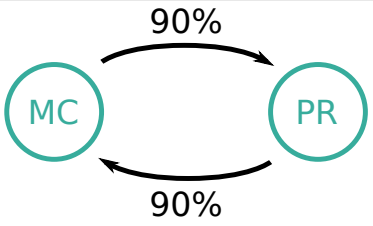
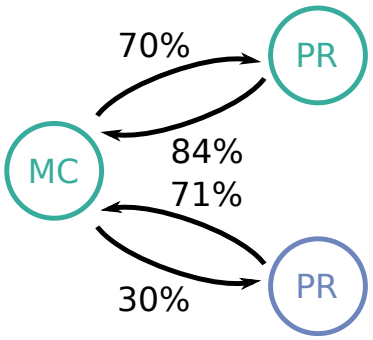
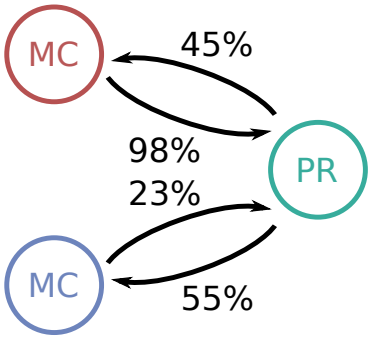
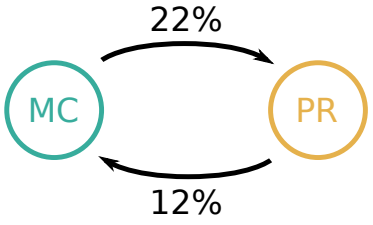
	There is a one to one connection between a MCTrackCand and a track from the track finder. The MCTrackCand is labeled found and the other track is labeled matched.
	The MCTrackCand is found twice. The track from the track finder with the higher percentage (the green one in this example) is labeled matched, the other one cloned. The MCTrackCand is nevertheless labeled found.
	The track from the track finder is created with hits from many different MCTrackCands. As none of the corresponding hit ratios exceeds 66 %, the track is called fake. There is no precise reason why the number 66% was chosen. The hit ratios of the MCTrackCands itself do not play any role here. TODO: Is found for MC possible?
	The found track does not describe any of the MCTrackCands well (or well enough) - but is made out of background hits. This track is also called a fake.

Table 3.1.: This tabular shows the four different cases for the matching between tracks found by the track finder (on the left side of the pictures) and MCTrackCands (shown on the right side). The different colors differentiate between different tracks. The connection between tracks shows that these two tracks share hits. The two percentages on the arrows are the percentages of hits they share in respect to the total number of hits in the MCTrackCand/track candidate from the track finder.

4. Track Finding in basf2

After having described the principle of the track finders implemented in basf2in the previous chapters we can now go on to show the actual implementation. The two track finders were already written and one of the tasks of this thesis was to put them together and improve them. Because we want to make use of the benefits of both track finders we use the workflow shown in figure 4.1. The shown steps are all described in more detail further down.

4.1. The TrackFindingCDC Package

Common code basis

4.2. The Background Hit Finder

The beforementioned track finders are tuned to not pick up background hits into a found track candidate or even form a candidate from background hits only. Nevertheless even a single background hit per track can lead to a reduced momentum resolution or can even cause the fit to fail completely. Together with the reduced combinatorics and therefore reduced computing time when throwing away unusable hits there is the need to distinguish between signal and background hits even before the tracking starts. This is the purpose of the BackgroundHitFinder module described in this section.

For deciding if a hit is to be used in the track finder, the module uses implemented filters. As these filters are used widely in the whole CDC tracking framework, they will be described here in more detail. To filter out bad hits, not only the information of one hit but the infromation of so called clusters of hits is used. These clusters are formed by one of the presteps in the local track finder - but can also be used standalone. They are created by clusterizing all CDCHits in such a way as one cluster includes the maximum number of connected hits. Two hits are called connected if there is no non-fired geometrically wire between them. Figure **??** shows an event display in the r-ϕ-plane with the clusters drawn in different colors. Figure **??** shows the hit purity of ANZAHL of the clusters from typical events. The hit purity is computed as the ratio between the number of hits in a clusterwhich belong to a signal track divided by the number of all hits (including the ones coming from background only). As can be seen clearly in this histogramm a cluster

Figure 4.1.: The proposed workflow and combination of the two track CDC finders in basf2. The green boxes refer more or less to one module. The arrows describe parts of the data flow between the modules. For clarity not all necessary parts are shown here.

is either full of background or full of signal hits - for that throwing away a background cluster does not involve deleting hits than are needed for signal tracks.

In every event, each formed cluster is passed to the chosen filter with the chosen filter parameters. This filter can return every number as a result including the C++ std::nan, which is chosen as the value for indicating that the cluster should not be used further for track finding as it is marked as background by the filter. Actually, the non NaN numbers returned by the filter are not used further in the following algorithms and are just described here for later reference.

This filter algorithm is a very general one and leaves the possibility open to compare many different filters. For this as well as for anyl other used filter in the CDC tracking, more or less 5 filters are implemented by default which can be switched by using module parameters. These filters are:

BaseFilter is a filter that neglects every item - it is just for basing every other filter on a common ground class and not for usage.

AllFilter is the counterpart of the BaseFilter and accepts all items. It can be used for testing purposes to make a filter fully transparent.

SimpleFilter is a filter based on self-implemented cuts. The variables to cut depend on the planned usage of the filter and need to be implemented by the programmer.

TMVAFilter is based on a trained boosted decision tree. It passed the variables defined in a so called VarSet to the BDT and returns the result to result between 0 and 1. If the result is lower than a certain cut definable on runtime, the TMVAFilter returns std::nan. The variables and their calculation can be freely defined by the programmer. In most of the cases the performance of a well-trained TMVAFilter exceeds the one of a simple filter.

RecordingFilter returns a constant defineable on runtime and can therefore not directly be called a filter. Instead its purpose is to write the same variables as the TMVAFilter uses together with a truth information of every incming item into a ROOT TNTuple. The output file with this TNTuple can later be used to train the BDT to categorize according to the truth information. This filter can of course only be used on simulated data.

MCFilter filters the items according to the truth information compiled with the MC information that also the recording filter uses.

As an example for the mentioned VarSet, the variables for distinguishing the background clusters from the signal clusters is described in table **??**. The truth information used for the BDT training is also described there. The trained BDT has PARAMETERS... EVENTS were used for training.

After the training procedure the corresponding TMVAFilter can be used. The results are summerized in **??**.

Results for Clusters/Hits, Results for Legendre Finder

4.3. Improvements on the Legendre Track Finder

4.3.1. The class `QuadTreeProcessorTemplate`

4.3.2. Postprocessing after the track finding

4.3.3. Results

Timing

4.4. Improvements on the Stereo Hit Finder

4.4.1. Principle of the StereoQuadTree

4.4.2. Results

Timing

4.5. The `SegmentTrackCombinerModule`

4.5.1. Principle of the Segment Track Combiner

+ Task

4.5.2. Used Filters

4.5.3. Results

4.6. Further Approaches

4.6.1. A quad tree for segments

4.6.2. Filter for tracks

4.6.3. VXD-Merger??

4.7. Additional software changes

4.7.1. The class `RecoTrack`

4.7.2. The `ipython_handler`

5. Analysis of the implemented Track Finder

5.1. Comparison of the two track finder

5.2. Tracking efficiency with different input parameters

5.2.1. Tracking efficiency with different particle and background types

5.2.2. The TMVA filters

6. Momentum estimation of slow particles with the ADC in the VXD

6.1. Prestudies

6.2. Incooperation in the helix fit

7. Summary

8. Acknowledgement

A. Listings

B. List of Figures

C. List of Tables

D. Bibliography

Bibliography

[1] **Belle II Collaboration**, T. Abe *et al.*, "Belle II Technical Design Report," tech. rep., 2010. arXiv:1011.0352 [physics.ins-det].

[2] C. Pulvermacher, "dE/dx particle identification and pixel detector data reduction for the Belle II experiment," Master's thesis, KIT, 2012. https://ekp-invenio.physik.uni-karlsruhe.de/record/48263.

[3] H. Bethe and J. Ashkin, "Passage of radiation throgh matter," *Experimental Nuclear Physics* **1** no. II, (1953) .

[4] **KEK High energy accelorator research organisation**, "New electronics tested for Belle II central drift chamber,". http://www2.kek.jp/proffice/archives/feature/2010/pdf/BelleIICDCDesign.pdf.

[5] O. Frost, "A Local Tracking Algorithm for the Central Drift Chamber of Belle II," Master's thesis, KIT, 2013. http://ekp-invenio.physik.uni-karlsruhe.de/record/48172.

[6] **Python Software Foundation**, "Python 2.7 pogramming language," 1991. https://www.python.org/.

[7] R. Brun and F. Rademakers, "ROOT – an object oriented data analysis framework," *Nucl. Instrum. Meth.* **A389** no. 1, (1997) 81–86.

[8] D. Lange, "The EvtGen particle decay simulation package," *Nucl. Instrum. Meth.* **A462** (2001) 152–155.