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

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

2. Experimental Setup

2.1. Belle II Detector

SuperKEKB

2.1.1. Overview of the detector parts

Picture, different parts and their purpose

2.2. The Tracking Detectors

2.2.1. Overview

2.2.2. CDC

Axial + Stereo!

2.2.3. 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 aquisition, 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. Together with external programming libraries like ROOT or EvtGen thare are already on the market this frameworkbuilds the base for every software written for the experiment.

quotes

The software is devided into several packages - each serving a single purpose or summerizing 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 paramters 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 fidning can be found in listing **??**.

listing

Caused by this extremly modular structure not only parallel processing but also debugging of intermediate steps can be performaed 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 vizualisation 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 serialisation mechanism together with data member dictionaries for the C++ classes created by the C++ interpreter of ROOT called CINT.

3.2. The used Figures Of Merit

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

Global vs Local

3.3.1. The Legendre Track Finder

Legendre Principle, QuadTree, Stereo

3.3.2. The Local Track Finder

Clusterizer, Automaton-Principle

3.4. Multivariate Classification

Classification, BDTs

4. Track Finding in `basf2`

After having described the principle of the track finders implemented in `basf2`in 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

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

Figure 4.1.: The proposed workflow and combination of the two track 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.

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