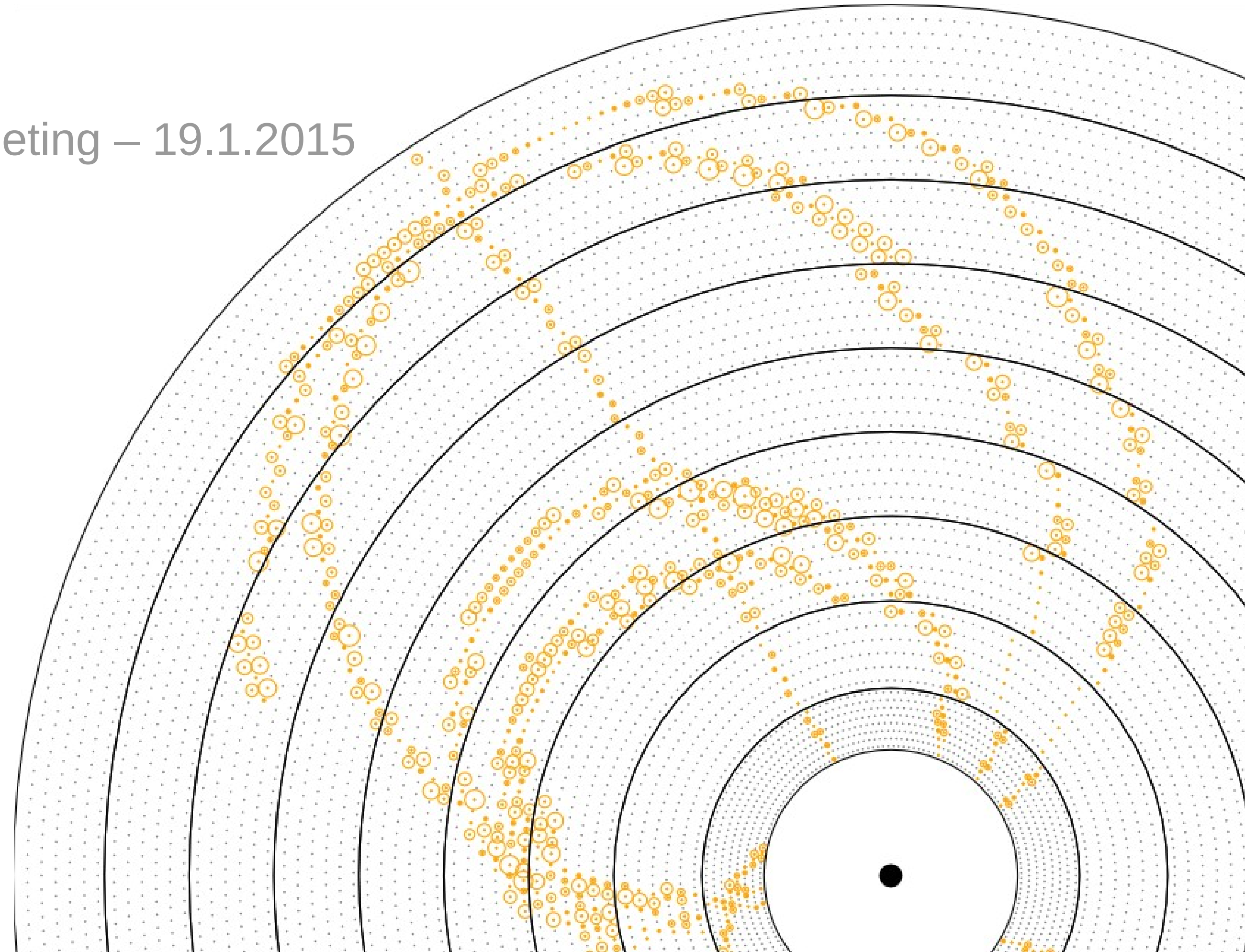
Common Validation Code Base Discussion

Thomas Hauth

Belle II F2F Tracking Meeting – 19.1.2015



Introduction

- Shortly before Christmas last year, I started to investigate the possibility to have a common code base for tracking validation purposes

  This common code can be useful in the following areas:
  - Production of the validation plots for the nightly builds website
  - Documenting quality improvements to tracking code by developers
  - Quick & local feedback loop for developers modifying the tracking code
  - Smaller codebase: better to maintain and new features allows everyone to profit
  - Zero work (ok, 0.01) to setup a validation for a new track finder or event/particle type

- My intention was to look for (already existing ?) code which offered a superset of all features of the current validation scripts:
  - Track finding efficiencies and fake rate etc.
  - Fitting quality criteria: residual and pull distributions of fitted tracks etc.

Current Status of Tracking Validation

Karlsruhe Institute of Technology

- In the following, a review of the current tracking validation code located under "tracking/" will be given
- All have different implementation and most share zero code (except for some using the MCTrackMatcherModule for reco track to MCParticle mapping
- Most implementations are dedicated to a specific validation purpose (e.g. genfit fit quality, performance of one track finder)
- The following symbols will be used to mark the features implemented by a specific validation step:

F it quality (Residuals, Pulls, etc.)

E fficiencies and Purity (of Track finding and fitting)

H it efficiency

R esolution

Current Status of Tracking Validation

as of subversion revision r14927

Code run in the Validation

tracking/validation/**01_steering_genFitStudy.py**

- Reads genfit::Track and MCParticle information
- Runs TrackFitCheckerModule and stores results in ROOT-file tree
- Reads genfit::TrackCand to MCParticles relation to determine truth information

tracking/validation/**02_plot_genFitStudy.py**

- Plots the data stored by the previous step

F R

Used for: Genfit fitting evaluation (no track finding) **Input:** Particle Gun with Pi+

tracking/validation/**12_tracking_Efficiency_runTracking.py**

F E R

- Runs the StandardTrackingPerformanceModule which reads genfit::Tracks
- Uses genfit::TrackCand and method getMcTrackId() to find related MCParticle
- Stores results in in a ROOT TTree

tracking/validation/**13_trackingEfficiency_createPlots.py**

- Plots results of previous step
- Code remarkably similar to 02_plot_genFitStudy.py, mostly filling histograms & profiles

Used for: Validation of tracking in the full reconstruction

Input: Particle Gun with Muons in various Pt bins

Current Status of Tracking Validation

as of subversion revision r14927

tracking/validation/[**cdcLegendreTracking|cdcLocalTracking|cosmicsTrackingValidation**].py

- MCTrackMatcher module to create MC <> RECO association to compute efficincies, residuals etc. (more on MCTrackMatcher on following slides)
- Python-based basf2 module reads tracking information and fills root histograms and profile plots
- Python library code for the plotting part is located in the tracking/scripts and shared between the three validation scripts

F E H R

Used for: Validation of cdcLegendre & cdcLocal Tracking **Input:** Comsics generator or EvtGen

Tracking Validation Code not run by default

tracking/modules/**trackingEvaluation**/

- Reads information set by MCTrackMatcherModule
- Stores information about each reconstructed track in a ROOT TTree

F E H R

tracking/modules/**trackingPerformanceEvaluation**/

- Reads information set by MCTrackMatcherModule
- Stores results of analysis to ROOT file in histograms and profiles

F E H R

Introduction to MCTrackMatcher module

- This module written by Oliver performs a matching between genfit::TrackCands and MCParticles solely on the shared hits between a reconstructed track and a MCParticle
- Reconstructed and MC tracks are grouped in categories

Reconstructed Tracks

Matched

Matches to a MCParticle with sufficient hit efficiency and the best purity of all reco tracks

Clone

Matches to a MCParticle but has not the highest hit efficiency of all reco tracks of this MCParticle

Background

Matches to no MCParticle hits and is made up of background hits

Ghost

Matches to no MCParticle as the mininum hit efficiency and/or purity is not reached

MCParticles

Matched

Matches to a reco track with the best hit efficiency and purity

Merged

Contained in a reco track, but another MCParticle has more hits this reco track

Missing

No match with sufficient purity to any MCParticle

A more detailed description can be found here:
http://kds.kek.jp/contributionDisplay.py?sessionId=9&contribId=71&confId=15329

Features for a common validation code base

I compiled this list with input from the Twiki web page
https://belle2.cc.kek.jp/~twiki/bin/viewauth/Software/TrackingValidation
and discussions with colleagues from the Belle II Tracking community

- Ability to validate either one track finder or track fitter standalone or run the whole reconstruction chain
- Flexible input/event generation: multiple generators (ParticleGun/EvtGen) with specific settings and reading from a pre-produced ROOT file
- Plots of finding efficiency, fake rate, clone rate and hit efficiency over
  [momentum, pt, phi, theta, d0, charge, particle id, occupancy]
  - For the definition of a found track, the MCTrackMatcher nomenclature can be used, the defaults are:
  - 66% purity: for each false hit added to a track, two correct ones must be present
  - 5% hit efficiency: at least 5% of the MCParticle's hit must be contained in the track
    - These values can be re-tuned for concrete sub-detector track finders
  - A special set of plots to show the background contribution to the fake tracks
- Resolution, Errors, Residuals, Pulls of fitted parameters
- Persistent naming and telling descriptions of the plots

Features for a common validation code base (cont.)

- Plotting runtime and memory consumption of tracking modules run during the validation
  - Could this be done even on validate_basf2.py level?
  - To consider: Reliable runtime measurements are, depending on the environment, hard to achieve:
    - On a shared system (->cluster) other factors (->users) can influence the runtime
    - On a local system, the validation is executed in parallel an all cores which makes reproducible runtime measurements difficult
  - Including the runtime and memory numbers in the validation history (as any other quantity, plot) allows for a convenient way to spot regressions in this area
- The option to disable/enable whole set of plots (e.g. all fit quality plots in case only track finding should be validated)
- Optionally add a curve describing the inofficial/official goals (e.g. final goal, next milestone, …)
  - This allows to easily see which parts can still be optimized
- Also add the ideal track finding / fitting results produced by MCTrackFinder

ANTI-Features for a common validation code base

Karlsruhe Institute of Technology

To prevent feature-bloat and and a unmaintainable code base, we should agree on some anti-features, which are not part of this validation code base:

- Sub-detector specific plotting code, which is only used to debug very detailed aspects of one implementation
  - One can possibly think about a low-threshold way to extend the validation code with specific plots without touching the common code base
- Track finding / Track fitting module specific data structures
  - The common language should always be genfit::TrackCand/genfit::Track and MCParticles and the relations between them

Way to go forward

- The MCTrackMatcher is a good module to base further tracking validation developments on
  - Offers consistent categorization of both MCParticles and reconstructed tracks
- Around 70% of the listed features are already implemented in the cdcLocal/cdcLegendre/Cosmics validation scripts
  - These scripts (and the underlying python classes) are very modular and will be the most easiest to extend and maintain
  - I implemented a replacement for the **12_tracking_Efficiency_runTracking.py** as a test using the library in tracking/scripts
    - This took around 3 hours, the necessary flexibility is available
- My proposal is to not implement all the features required to replace existing validation code all at once
  - Rather the way should be to replace one validation script with the common code at a time and add the necessary features and flexibility as needed
  - When replacing script n+1 already more features are available

BACKUP

Central construct - the confusion matrix

Confusion matrix

	MC tracks			Background
	...	...	...	...
PR tracks	...	Common hit / NDF content	...	...
	...		...	...
	...	...	...	...
Unassigned	...	...	...	...

Confusion matrix of the example

	mc_1	mc_2	mc_3	mc_4	Background
pr_1	24	0	0	0	0
pr_2	0	6	8	0	0
pr_3	0	0	19	0	0
Unassigned	0	0	0	21	0

http://kds.kek.jp/contributionDisplay.py?sessionId=9&contribId=71&confId=15329

Purity matching

Purity matrix

Find purities by normalizing the rows of the confusion matrix.

Purity matrix of the example

	mc_1	mc_2	mc_3	mc_4	Background
pr_1	**1**	0	0	0	0
pr_2	0	0.43	**0.57**	0	0
pr_3	0	0	**1**	0	0
Unassigned	0	0	0	**1**	0

Row-wise matching - purity matching

- Search highest purity Monte Carlo track for each pattern recognition track.
- Look for highest entry in each row.
- Relation *hp*: n $\leftrightarrow$ 1 - **Refinement necessary**

http://kds.kek.jp/contributionDisplay.py?sessionId=9&contribId=71&confId=15329

Efficiency matching

Efficiency matrix

Find efficiency by normalizing the columns of the confusion matrix.

Efficiency matrix of the example

	mc_1	mc_2	mc_3	mc_4	Background
pr_1	1	0	0	0	0
pr_2	0	**1**	0.3	0	0
pr_3	0	0	**0.7**	0	0
Unassigned	0	0	0	**1**	0

Column-wise matching - efficiency matching

- Search highest efficiency pattern recognition track for each Monte Carlo track.
- Look for highest entry in each column.
- Relation *he*: 1 $\leftrightarrow$ m - **Refinement necessary**

http://kds.kek.jp/contributionDisplay.py?sessionId=9&contribId=71&confId=15329