News & Discussion

Recent Issue on the Mailing List

Hi Jakob,

I know the code is copied from somewhere else. Somewhere else inside the tracking package unfortunately.

Do we really want to do something like this in order to avoid having such opaque concepts as enums, integers or a list of classes where they register themselves? Or, if this really needs to be hyperdynamic, why not do it in python? I mean, this code definitely isn't fast anyway.

I would claim that when you have to call abi::__cxa_demangle(), then you should either have a very good rationale for doing so or something is broken very badly.

Sorry about the outburst.

Cheers,
- Tobi

I would indeed recommend against using such code, because I can't understand it.

In this specific module, it might be OK, if both the programmer and future maintainer agree, but this seems not optimal.

while I don't have any say on the tracking software I really have to agree with Tobias. This code looks horribly over-engineered. Not least because boost::regex is used for a simple char replacement (std::replace anyone?).

I also would like to point out that this has the potential for pitfalls along the way. The new libstdc++ uses inline namespaces and abi tags for backwards compatibility which will end up in the symbol names and thus in the typename. And libc++ has been using inline namespaces from the beginning to be able to coexist with the libstdc++. As an example, using Jakobs code, the output for std::string

```
Belle2__{std__string}
```

gets converted to

```
Belle2__{std____cxx11__basic_string{char, std__char_traits{char}, std__allocator{char} }}
```

with the new libstdc++ abi and looks like

```
Belle2__{std____1__basic_string{char, std____1__char_traits{char}, std____1__allocator{char} }}
```

for clang++ -stdlib=libc++. We plan to switch to the new libstdc++ ABI as soon as clang/rootcling supports the symbol names properly.

I know this code is probably only intended for custom classes so std::string might not be a likely candidate but I still advise against using something this low-level[1] if not "absolutely necessary(TM)".

Cheers,

Martin

Some Recent Changes I made to code, were

- adding dots at the end of comments and inside comments after the part, that doxygen should take as “auto brief”.
  - → auto brief needs a dot in longer comments. To not forget to make one in longer comments, simply make always a dot at the end of the doxygen comment part intended as “brief”.
  - A dot at the end of a comment helps as well clarifying, that you indeed intended to end your statement at that point.
    ⇒ Please use dots!
- removing “\n” statements in comments, sometimes in mid-sentence, where no paragraph was meaningful.
  - doxygen makes html, and the character width may not be as in your comment, this is only a useful doxygen comment element, when you want to make a paragraph, not to make shorter lines in the html, where the different sized lines completely irritate.

Discussion on typedef Usage

- A typedef doesn't define a type per se, e.g.
  typedef int NumberOfWires;
  doesn't make a type, that would tell the compiler not to take an int. So, e.g.

```
/// Typedef to indicate that an axial segment is expeected.
typedef CDCRecoSegment2D CDCAxialRecoSegment2D;
```

  and

```
/// Typedef to indicate that an stereo segment is expeected.
typedef CDCRecoSegment2D CDCStereoRecoSegment2D;
```

  don't tell the compiler not to use a CDCRecoSegment2D in both places, where either of the typedef'd "types" is required.
  So the following doesn't prevent to use e.g. 3 axial segments:

```
/// Constructor taking the three segments the triple shall be made of.
CDCSegmentTriple(const CDCAxialRecoSegment2D* startSegment,
                 const CDCStereoRecoSegment2D* middleSegment,
                 const CDCAxialRecoSegment2D* endSegment);
```

Solutions

```
/// Constructor taking the three segments the triple shall be made of.
CDCSegmentTriple(const CDCAxialRecoSegment2D* startSegment,
                 const CDCStereoRecoSegment2D* middleSegment,
                 const CDCAxialRecoSegment2D* endSegment);

/// Typedef to indicate that an axial segment is expeected.
typedef CDCRecoSegment2D CDCAxialRecoSegment2D;
```

- My preferred solution:

```
/// Constructor taking the three segments the triple shall be made of.
CDCSegmentTriple(const CDCRecoSegment2D* startAxialRecoSegment2D,
                 const CDCRecoSegment2D* middleStereoRecoSegment2D,
                 const CDCRecoSegment2D* endStereoRecoSegment2D);
```

  - Use the identifier to clear up both the type and additional information.
  - Simply eliminate the typedef (and optimally use a const ref instead of a non-const pointer, but that is probably too late...).

    → you know everywhere in the function what thing you are handling;

    → you don't have "fake type safety";

    → you can jump with one jump to the real declaration, instead of going first to the typedef;

    → compiling is faster;

    I don't see, how this wouldn't indicate as much as typedef'd type.

    Small disadvantage: If one later splits the class to several types, things become either a bit more work or a little bit less clear, but really hardly. Given the maturity of these classes, I don't see a big issue here.

- Second reasonable solution:

```
/// Typedef for type safely expeecting an axial segment.
typedef CDCRecoSegment2D<AxialTag> CDCAxialRecoSegment2D;
```

  - Make the typedef really indicating a different type by using a tag, as e.g. RAVE does with vectors in LOCAL, GLOBAL coordinates or POINTS vs. true VECTORs.

    → type safety;

    But not my preferred solution because:
    - takes longer to compile;
    - means more code;
    - it is conceivable, that in some cases we really do want to use both classes interchangably;

This is just one case, but I personally think in general that a typedef should NOT be used for indicating similar things. I prefer (`int nWires`) strongly to (`NumberOfWiresType wires`) and an additional typedef in general.

Talk@dEdx Meeting in the afternoon

I will restate parts of our wishlist, see next page; (talk in the backup)

Actual Wish List

from https://belle2.cc.kek.jp/~twiki/bin/view/Software/SpacePointCreatorModule

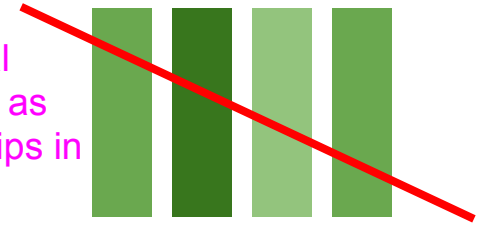
SpacePoints shall contain information, that the underlying clusters may have for potential filters [..] used as quality indicator for [..] Tracks[..]. This information can stem

- in the SVD e.g. from
  - timing information (how well do the u- and v-coordinate match in time [...], what is the best time estimate vs. T0, the which best match of opposite strips is this match (1st, 2nd,...))
  - energy deposit information (similar as for the time based information **plus** an estimate of the most probable expected energy loss (this is “dE”), if this can be estimated better than simply taking the best estimate of the factually deposited energy loss due to detailed cluster analysis [as one cluster has multiple hits/strips]),
  - shape information (minimum/maximum incident angle compatible with the cluster shape, favoured direction of inclination etc.)

    VxdID, and other special information (e.g. was the SpacePoint created from a dead/noisy channel, of course one might as well under some circumstances not create a SpacePoint from a noisy channel, despite there is a cluster).

Why this list?

"expected dE" and actual estimated dE may differ, as there are several hits/strips in one cluster;

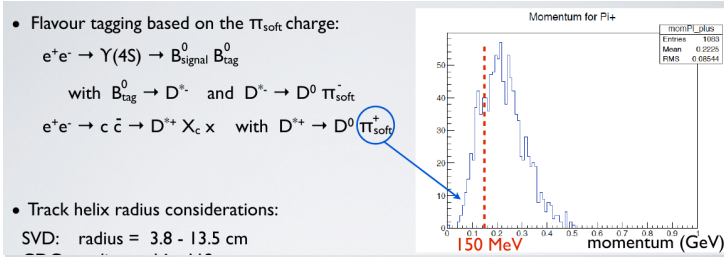


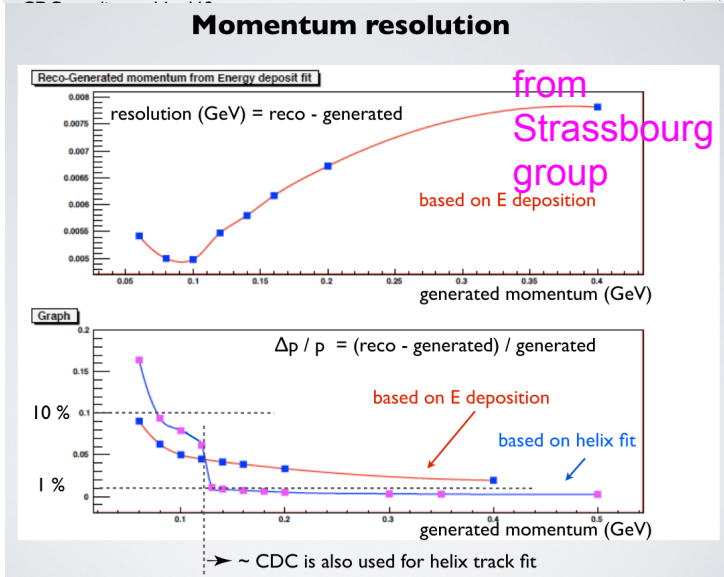
- Most of the information can help to distinguish between fake tracks and real tracks, which is fairly important for low momentum tracks;

- timing helps to distinguish between curlers and primary outgoing arms;
  could perhaps help as well with PID in connection with time of flight…, but such slow particles are "fish in a barrel" using dE/dx?; perhaps not for hyperons… [see later]

- best estimate of deposited energy can be put into track fit directly, otherwise (usually) we just take the most probable energy loss for the momentum and given mass hypothesis; **We want this for CDC, too!**

- most probable expected energy loss can be used to estimate the momentum, assuming a mass hypothesis





The RecoTrack
&&
Energy Loss in the Fit

Martin & Nils

The RecoTrack

- After next release, tracking wants to introduce a new object:

  - The `RecoTrack` is a new dataobject in the tracking package which inherits from `genfit::Track`.
  - It has therefore the same capabilities than the track and:
    - Relation-ready interface for the added hits (not reco hits).
    - `RecoHitInformation` object to save additional hit information (like RL-information in the CDC or sorting parameters).
    - Fitting preparation procedure build into the track (time-of-flight, AbsRep creation etc.)
    - Flag if last fit was successful.
  - `genfit::TrackCandidate` objects can be build out of `RecoTrack` objects and vice versa.

Relevance for dE/dx

- The “dx” essentially comes from it;
- Currently you probably use the genfit::Track, but
  - you can’t use the usual Relation interface with the genfit::Track,

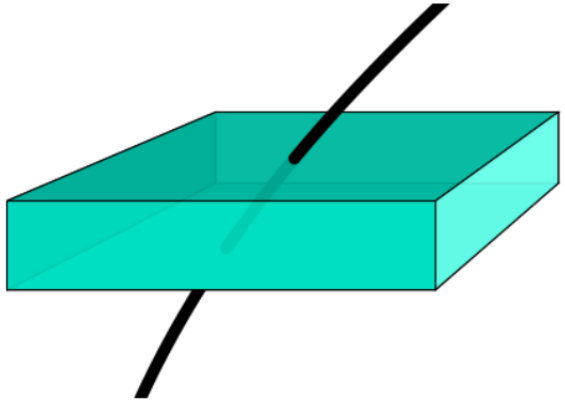
    ```
    class RecoTrack : public RelationsInterface<genfit::Track>
    ```
  - you may want a finer granularity of hit association, which the tracking might be able to provide:
    - Hits, that almost certainly belong to the RecoTrack, e.g. ones, which are used in the fit and have a reasonable pull,
    - Hits, that probably belong to the track, but have perhaps a bad time measurement,
    - Hits, that probably don’t belong to the track, e.g. that were added by the track finder algorithm, but thrown out by the detailed fitting **or have a moderate pull only for some/one mass hypothesis** (as the drift time estimation uses time of flight, this can happen → there is an interplay between fitting and dE/dx)!
  - This can be easier communicated with Relations to the

    ```
    class RecoHitInformation : public RelationsObject
    ```

    SUMMARY → Stay Tuned!

Energy Loss in the Fit: Workflow

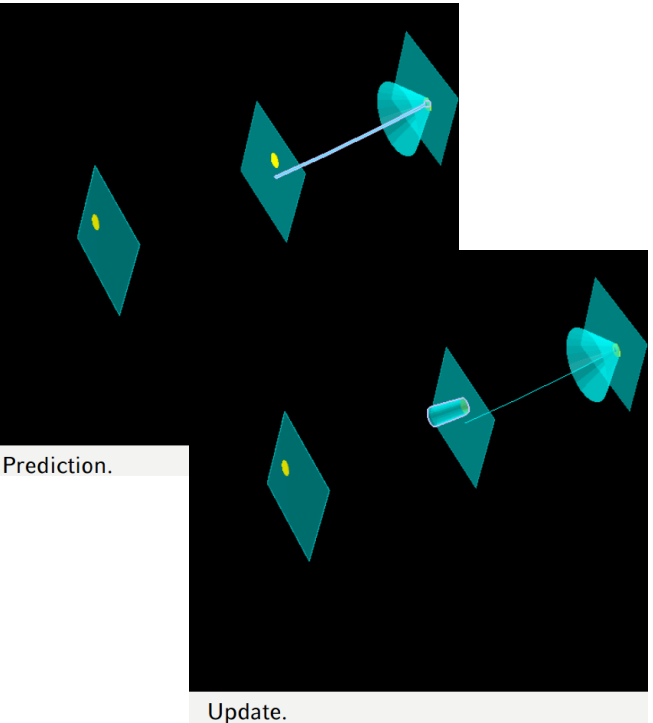
- Study for $p_{Pion} < 100$ MeV;
- Estimate for dE/dx from ADC and Track parameters;
- Create genfit::Measurement with only this measurement;
- Fit with Kalman or DAF

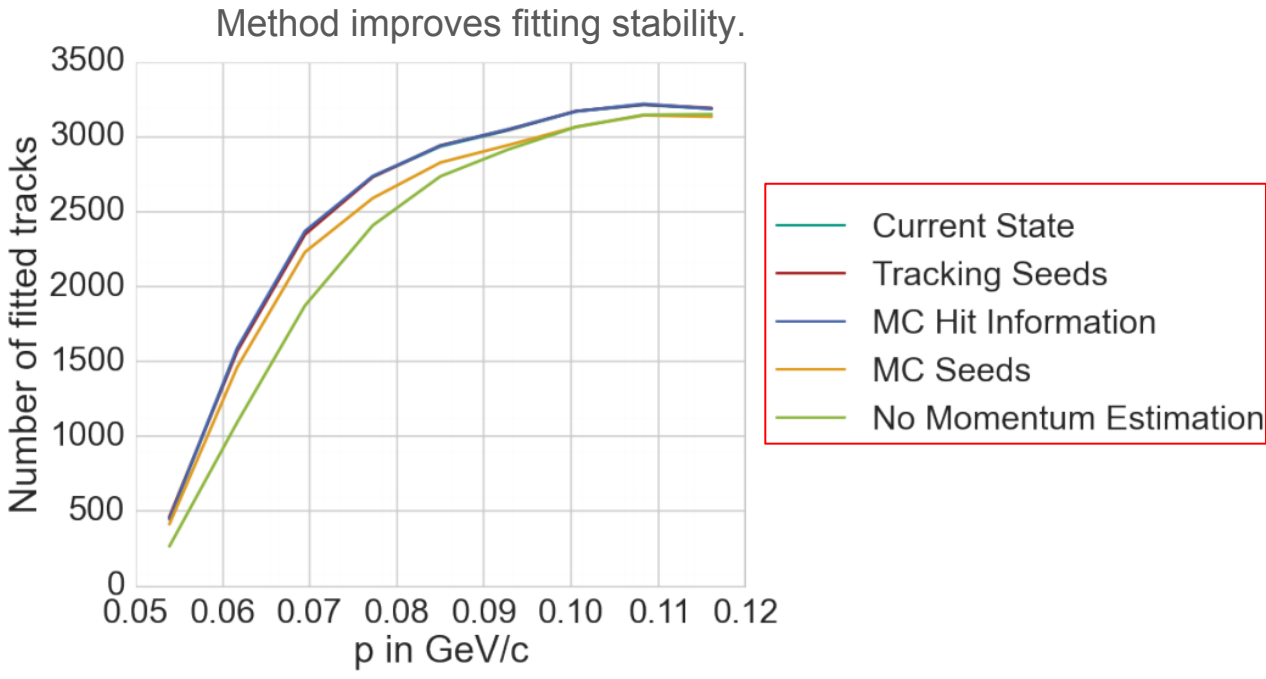


Actual work done by Nils, using stuff from Robert and Isabelle.

Path Length Estimation

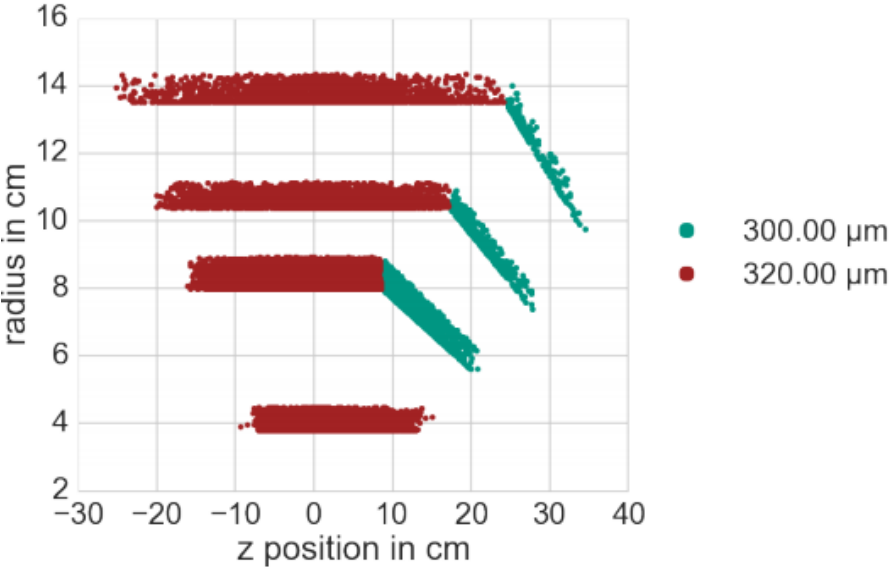
- Tracking Seeds good enough and better than MC Seeds (MCParticle momentum at IP).
- Current State will be used later in the real code.

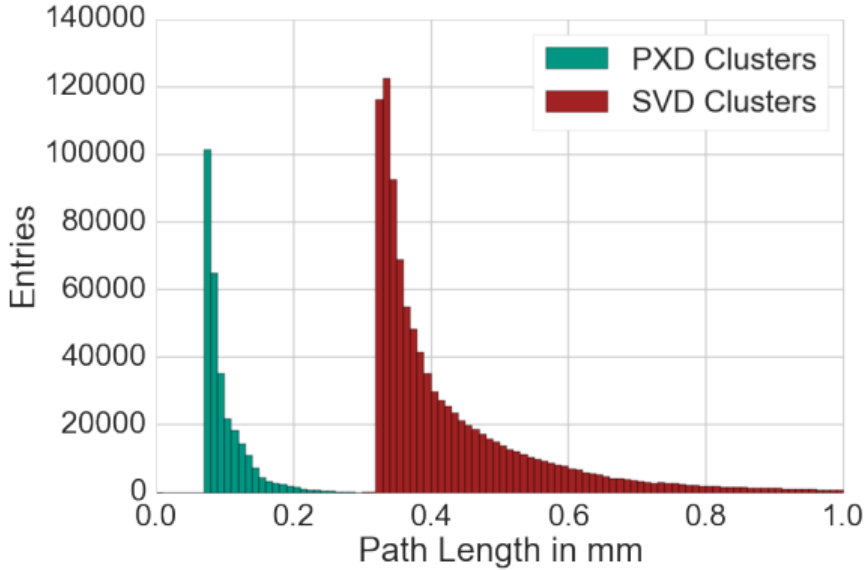




Path Lengths calculated with the Tracking Seed

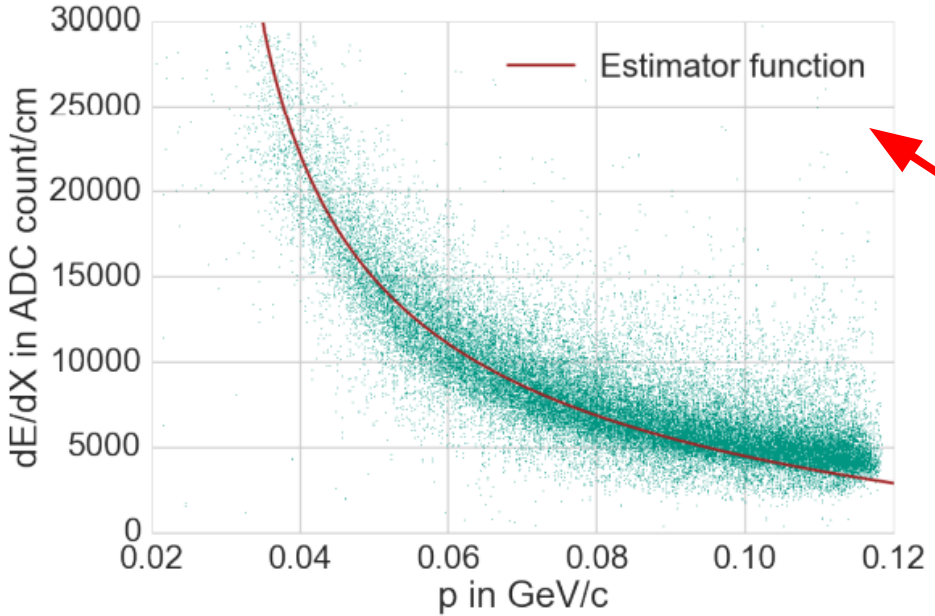
Two different vendors for barrel and forward region sensors in the SVD → Not a bug.

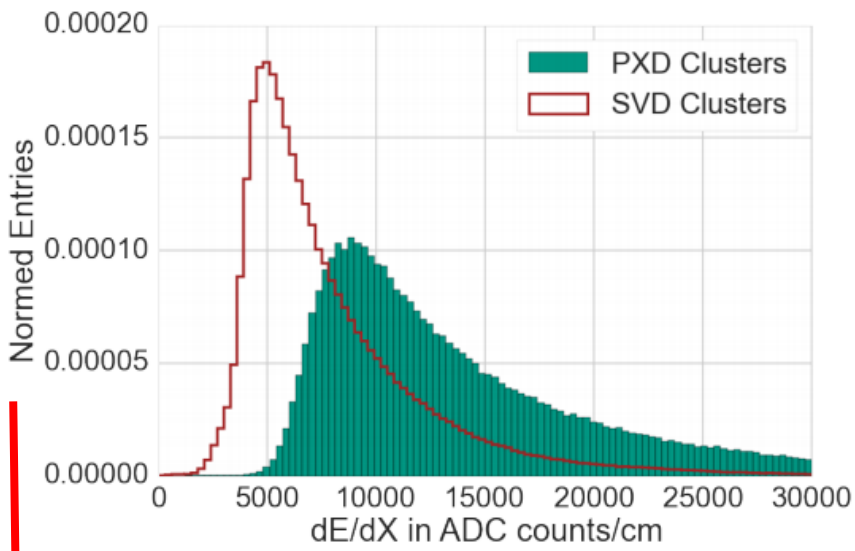


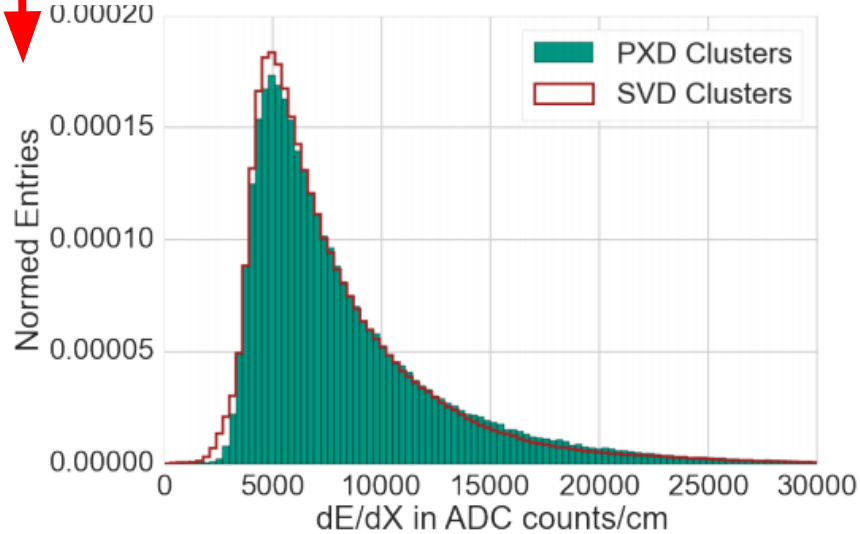


Meaning of ADC count within one detector the same, but not across detector. However, as material is the same, simply multiplying works for this exercise (not in real life, where ADC converters have non-linear behaviour). Calibration….

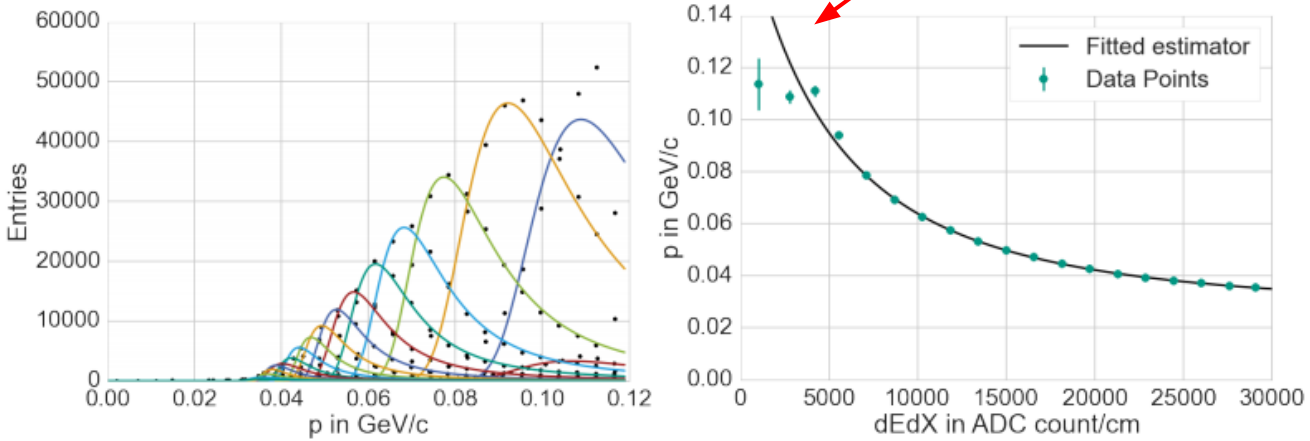
Probably LUT is better than Estimator function…, but across some range the function describes the median (but you see where it stops to work...)







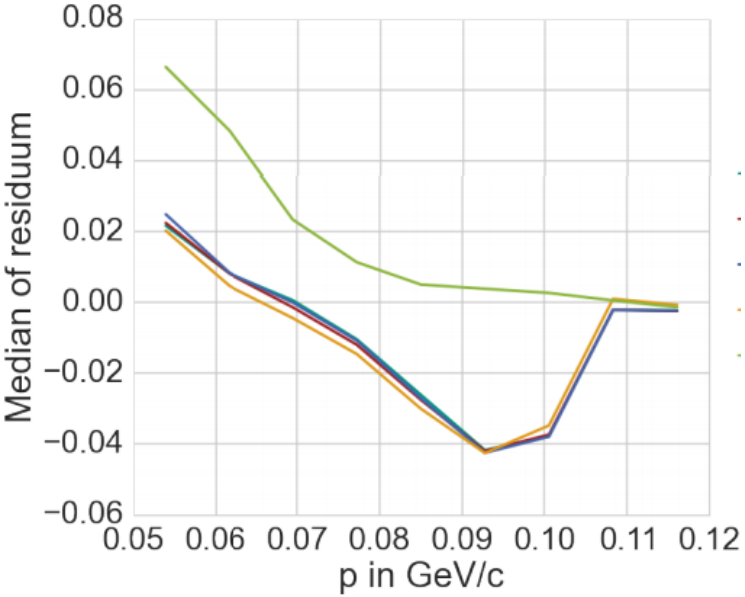
Estimating the function

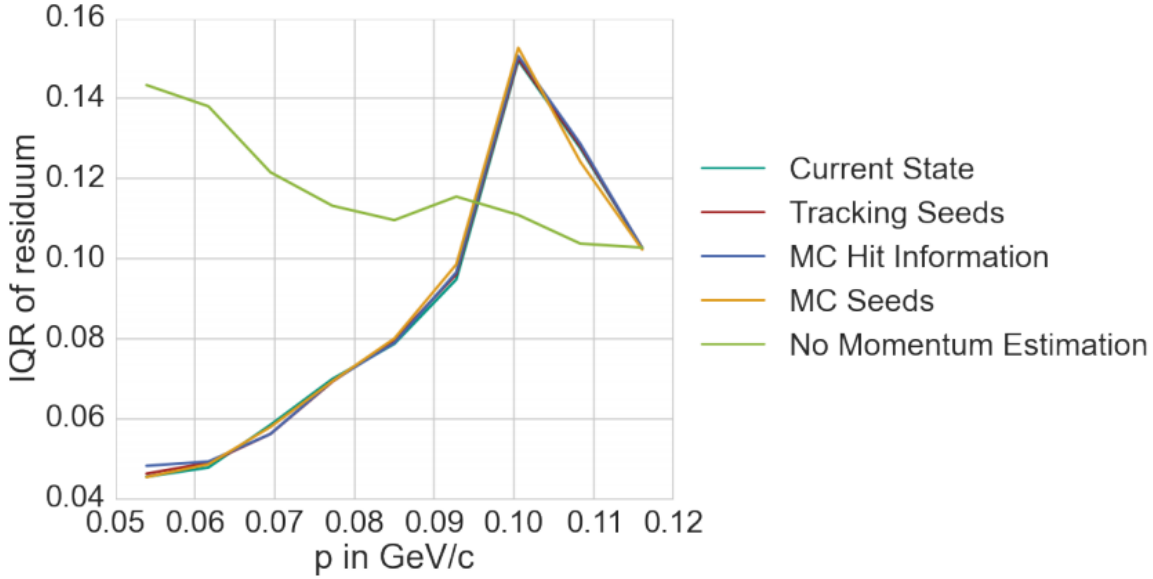


Transforming dEdX to p

- For each dEdX bin: fit the momentum distribution with a landau/gauß function
- Use the parameters of the distributions to calculate a function p(dEdX).

Results on Fit





Some moderate bias probably due to the tail of delta electrons.

At high momenta, the measurement is dominated by the fit to positions.

For higher momenta, the estimator function is simply wrong, probably a LUT will eliminate bias and improve p estimation.

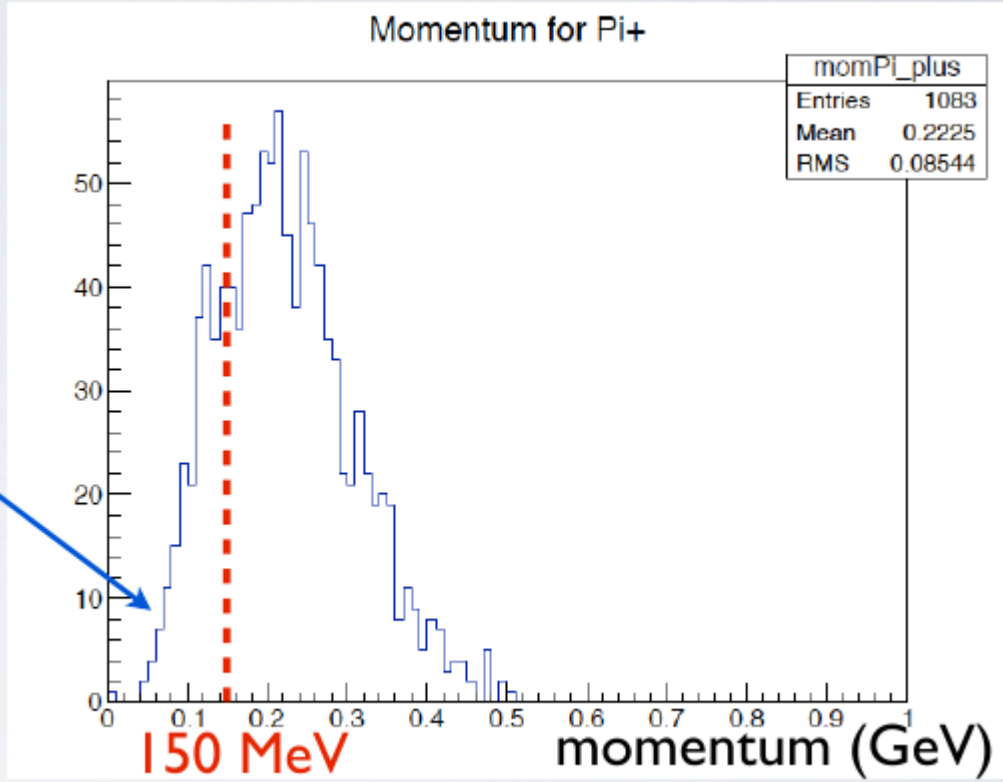
Factor 2 improvement for momentum@70 MeV

Flavour tagging based on the π_{soft} charge:

$e^+e^- \rightarrow \Upsilon(4S) \rightarrow B^0_{signal}\, B^0_{tag}$

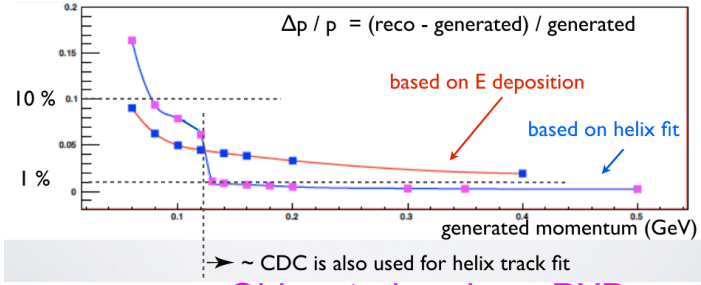
with $B^0_{tag} \rightarrow D^{*-}$ and $D^{*-} \rightarrow D^0\, \pi^-_{soft}$

$e^+e^- \rightarrow c\,\bar{c} \rightarrow D^{*+}\, X_c\, x$ with $D^{*+} \rightarrow D^0\, \pi^+_{soft}$



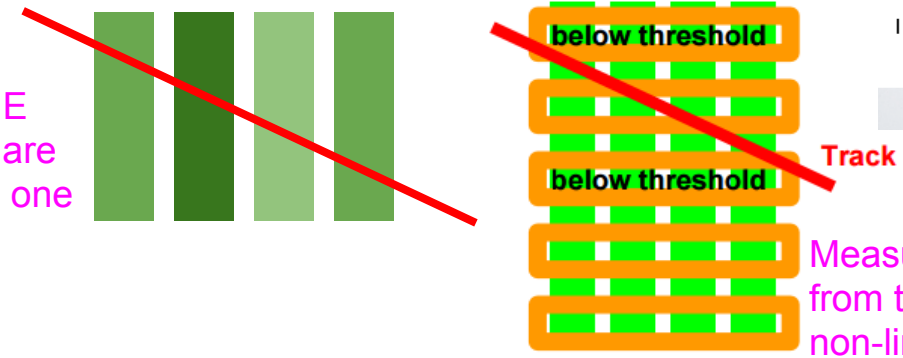
Energy Loss in the Fit: Other Comments

- At the F2F tracking meeting, it was suggested to go the other way around and use a Chi^2 term with the energy directly in it instead of the momentum estimation, but
  - central problem is finding out the momentum <-> energy loss relation with its uncertainties; estimating the energy loss given the momentum or the momentum given the energy loss doesn't make a big difference then.
  - The energy loss estimation used in the track extrapolation can't be used directly as estimation of measured energy loss, because it doesn't make any assumptions about the read-out behaviour.
    - Calibration of ADC count <-> measured energy loss of the particle is key!
- Study underestimates importance, because better momentum estimation with SVD only improves capability to associate the right PXD hits.



Older study, where PXD hits were likely broken.

"Expected dE" and actually estimated dE may differ, as there are several hits/strips in one cluster;



Measurements can be biased from thresholds, noise, ADC non-linearities...

Summary: Energy Loss in the Fit

- At very low momenta, the momentum estimation can be improved considerably;
- Path Length estimation is no issue; → Good News for dE/dx
- Momentum <--> energy loss relation is an issue;
  - Need for modelling the distribution (median, probability for deviation of X at momentum Y);
- Preliminary study performed, further development useful;

- Further needs for tracking, that can be at least partially related with calibration of dE/dx was communicated before, see backup;
  - especially best estimate of actual energy loss in both the SVD and the CDC!
  - PXD is less important, because the amount of energy loss simply isn't that much.

Back Up

Actual Wish List

from https://belle2.cc.kek.jp/~twiki/bin/view/Software/SpacePointCreatorModule

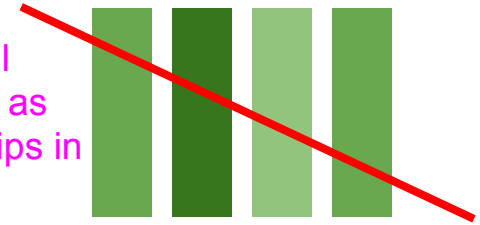
SpacePoints shall contain information, that the underlying clusters may have for potential filters [..] used as quality indicator for [..] Tracks[..]. This information can stem

- in the SVD e.g. from
  - timing information (how well do the u- and v-coordinate match in time [...], what is the best time estimate vs. T0, the which best match of opposite strips is this match (1st, 2nd,...))
  - energy deposit information (similar as for the time based information **plus** an estimate of the most probable expected energy loss (this is “dE”), if this can be estimated better than simply taking the best estimate of the factually deposited energy loss due to detailed cluster analysis [as one cluster has multiple hits/strips]),
  - shape information (minimum/maximum incident angle compatible with the cluster shape, favoured direction of inclination etc.)

    VxdID, and other special information (e.g. was the SpacePoint created from a dead/noisy channel, of course one might as well under some circumstances not create a SpacePoint from a noisy channel, despite there is a cluster).

Why this list?

"expected dE" and actual estimated dE may differ, as there are several hits/strips in one cluster;



- Most of the information can help to distinguish between fake tracks and real tracks, which is fairly important for low momentum tracks;

- timing helps to distinguish between curlers and primary outgoing arms;
  could perhaps help as well with PID in connection with time of flight…, but such slow particles are "fish in a barrel" using dE/dx?; perhaps not for hyperons… [see later]

- best estimate of deposited energy can be put into track fit directly, otherwise (usually) we just take the most probable energy loss for the momentum and given mass hypothesis; **We want this for CDC, too!**

- most probable expected energy loss can be used to estimate the momentum, assuming a mass hypothesis