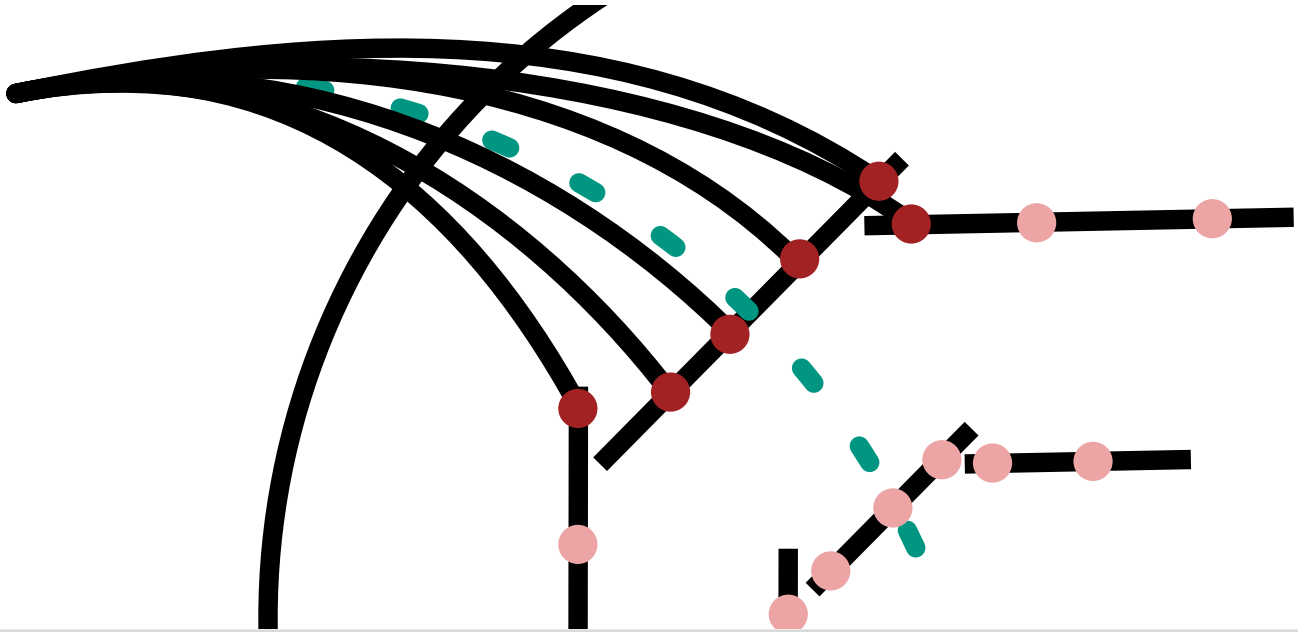
Karlsruher Institut für Technologie

Combinatorial Kalman Filter - PXD and SVD

Weekly Tracking Meeting.

Nils Braun | 19.09.2017

IEKP - KIT



Recap

1. For each Seed: find all possible matching *hit trains*, using:
   - three layerwise filters
   - extrapolations
   - Kalman updates
2. Apply a filter on each found candidate
3. Resolve Overlaps

What has happened since last time? - I

- I have found a bug in my Kalman Filter implementation, that only occurred in the PXD case.

Eigen and ROOT

Eigen and ROOT have different matrix element order definitions! (Actually, I was aware of this, but this information got lost during my implementation...)

Tests

Always write tests!

- PXD implementation finished and tested
  e.g. I have trained an MVA filter for overlap check for PXD $\Rightarrow$ now MC-free implementation possible.

What has happened since last time? - II

- Caching works now :-) Problem was with the definition of a ”plane” in GenFit (and again missing documentation...)
- Quite some code changes, making it (hopefully) easier for Miriam.
- Not only the overlap check, but also the layer filters work with a BDT now.

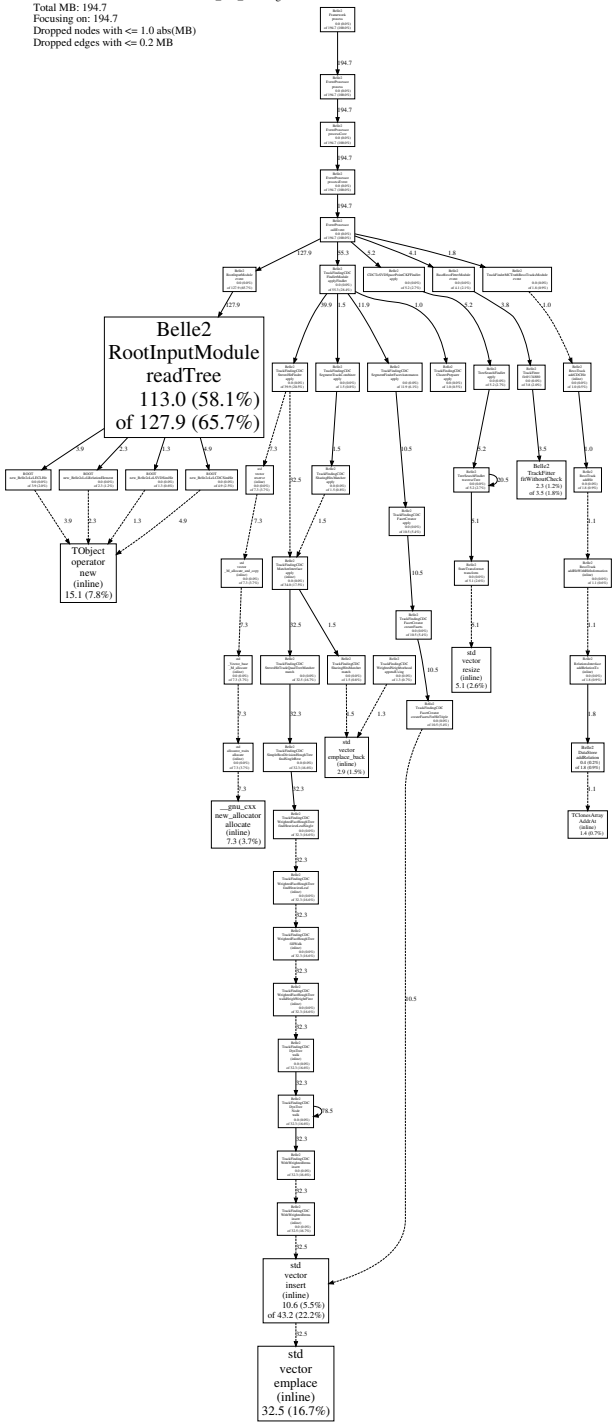
recoHit.constructMeasurementsOnPlane

Be aware that `recoHit.constructMeasurementsOnPlane` creates a new object, you have to delete yourself afterwards!

`jemalloc`

Use `jemalloc` to test for memory leaks and heap usage.

- New subdetector efficiencies/purities are very hard to understand...


/local/scratch/ssd/nbraun/basf2/bin/Linux_x86_64/clang/basf2
Total MB: 194.7
Focusing on: 194.7
Dropped nodes with <= 1.0 abs(MB)
Dropped edges with <= 0.2 MB
Belle2 RootInputModule readTree 113.0 (58.1%) of 127.9 (65.7%)
TObject operator new (inline) 15.1 (7.8%)
std vector emplace (inline) 32.5 (16.7%)
std vector insert (inline) 10.6 (5.5%) of 43.2 (22.2%)
__gnu_cxx new_allocator allocate (inline) 7.3 (3.7%)
std vector emplace_back (inline) 2.9 (1.5%)
std vector resize (inline) 5.1 (2.6%)
Belle2 TrackFitter fitWithoutTrack 2.3 (1.2%) of 3.5 (1.8%)
TClonesArray::ConstructAt (inline) 1.4 (0.7%)

BDT in layerwise filters

Benefits:

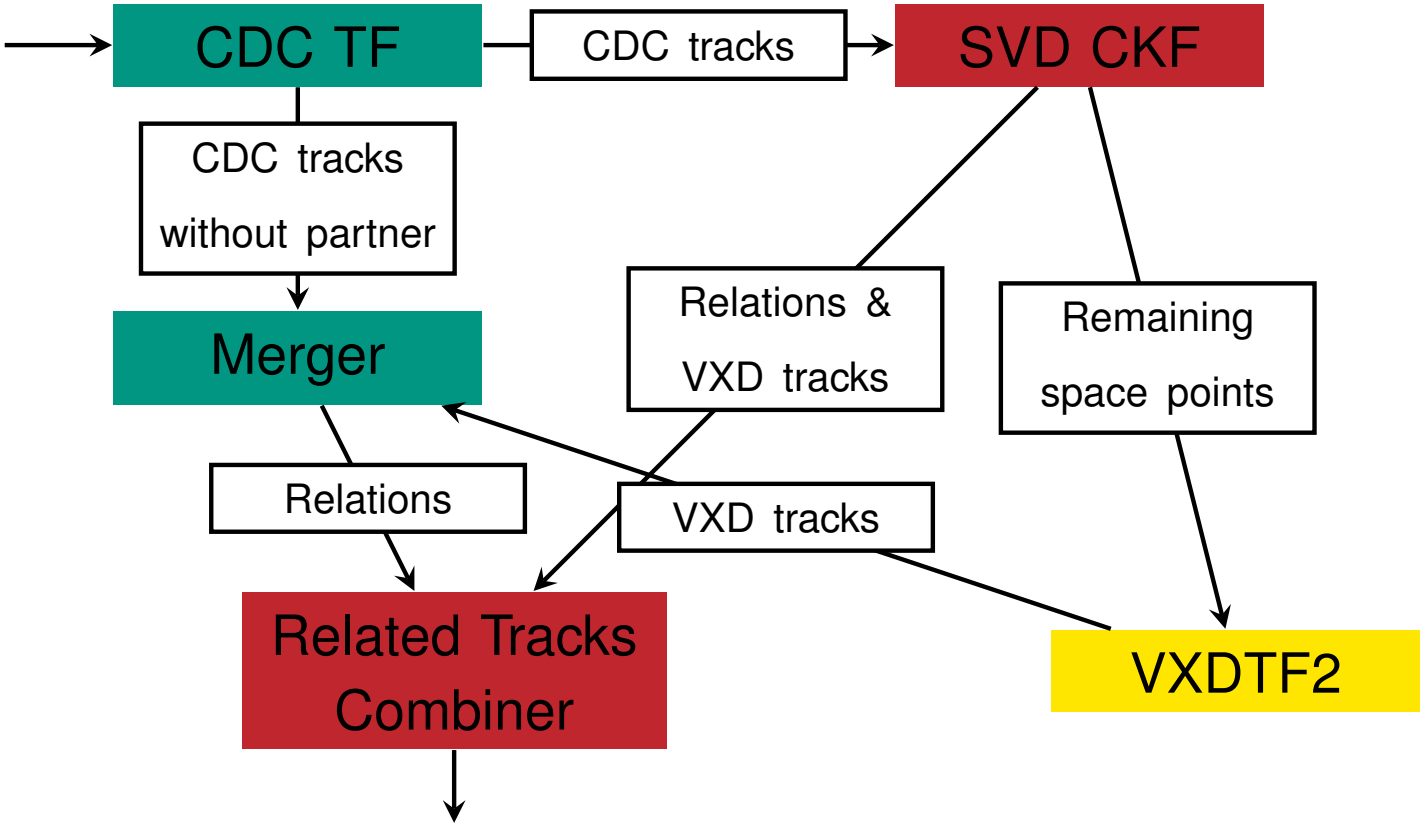
- Multivariate method
- Training is ”simple”
- Variable importance order gives interesting insights

Drawbacks:

- **Slow**
- Training on data needed?

Combinatorial Kalman Filter from CDC to SVD

Proposed Workflow



Setup

Events for the following

Typical generic $\Upsilon(4S)$ with 1000 events using 15th campaign background.

Current CDC + VXDTF2 for SVD only, default merger

CKF CDC + CKF, no merger needed

CKF with MC CDC CDC track finding is replaced by MC

MC CKF As CKF, but using MC information in the CKF filters

CKF + VXDTF2 as described before

MC Matching

Comparing the performance by hit-matching doing on:

- All Hits
- SVD hits only

Performance SVD only

	Current	CKF	CKF + VXDTF2
Finding Efficiency (prim)	0.946993	0.729256	0.965953
Finding Efficiency (all)	0.913839	0.712227	0.94654
Hit Efficiency (prim)	0.952381	0.94865	0.951127
Hit Efficiency (prim)	0.901898	0.691809	0.918744
Hit Efficiency (all)	0.944942	0.941783	0.942848
Fake Rate	0.180379	0.0574972	0.193046
Clone Rate	0.00387263	0.011274	0.023548
Hit Purity (SVD)	0.987022	0.994279	0.989803

- As overlap between CKF and VXDTF2 is large, no huge performance boost expected
- Fake Rate + Purity of CKF alone is good
- Combination gives reasonable results

Performance

	Current	CKF
Finding Efficiency (prim)	0.950516	0.951217
Finding Efficiency (all)	0.834396	0.836262
Hit Efficiency (prim)	0.778926	0.793219
Hit Efficiency (prim)	0.740382	0.754524
Hit Efficiency (all)	0.747453	0.763059
Hit Efficiency (SVD)	0.842577	0.904544
Fake Rate	0.139165	0.135205
Clone Rate	0.100124	0.0711744
Hit Purity	0.92874	0.929191

- Finding efficiency can not increase by design
- Hit efficiency (especially SVD) increased
- Clone rate dropped

Is this now good or not?

First answer: Well...

Is this now good or not?

First answer: Well...
Second answer:

- Influence on hit efficiency visible, also because ”merging” is not needed anymore
- It is hard to increase hit purity even more...
- All in all, I hoped for more impact...
- Problem: CDC tracks!

Performance SVD only

	CKF	CKF with MC CDC	MC CKF
Finding Efficiency (prim)	0.729256	0.918145	0.7579
Finding Efficiency (all)	0.712227	0.906919	0.745213
Hit Efficiency (prim)	0.94865	0.95482	0.962688
Hit Efficiency (prim)	0.691809	0.876663	0.729622
Hit Efficiency (all)	0.941783	0.947573	0.955452
Fake Rate	0.0574972	0.0477185	0.0211002
Clone Rate	0.011274	0.00507009	0
Hit Purity (SVD)	0.994279	0.993382	(1)

Performance

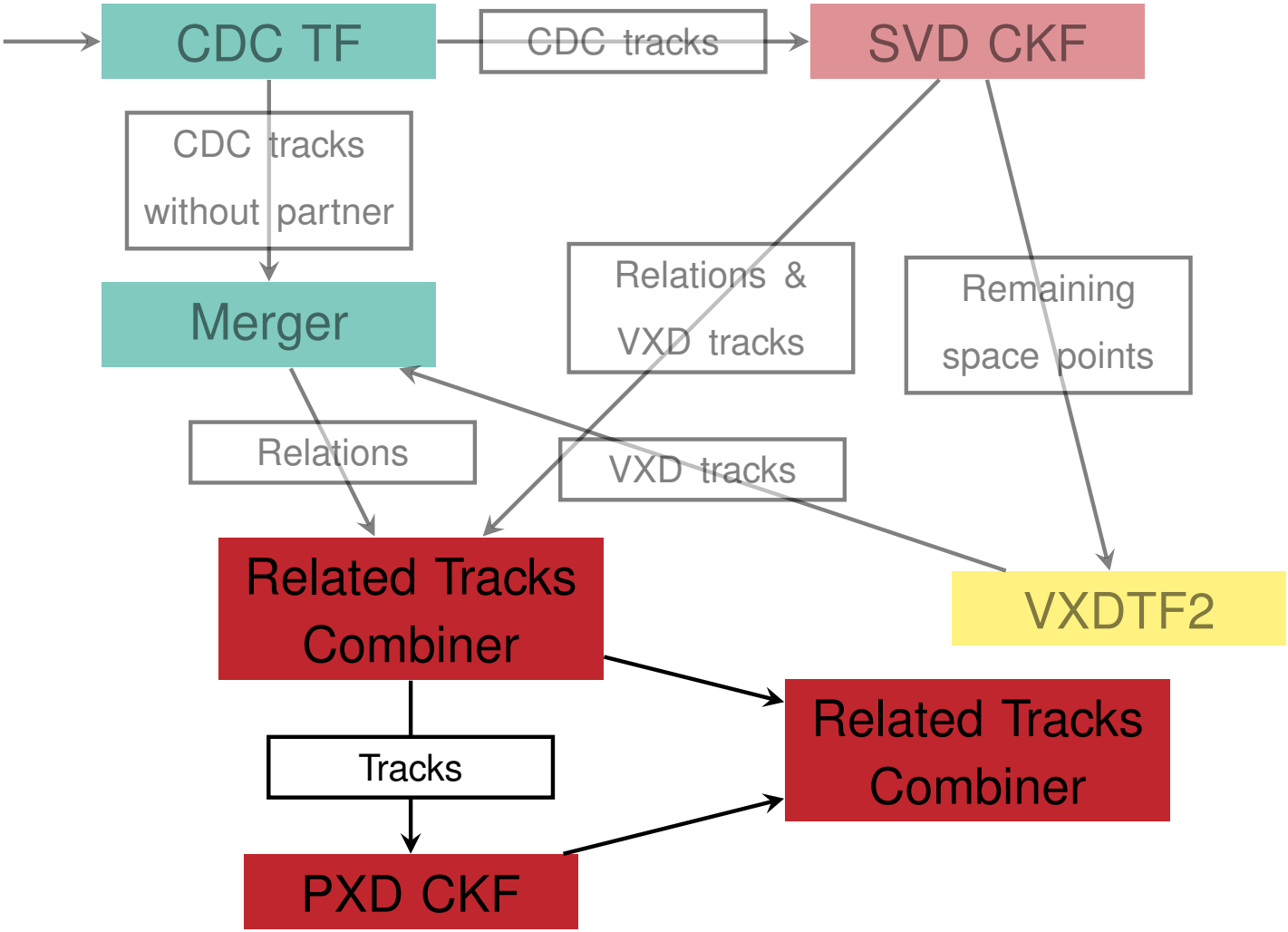
	CKF	CKF with MC CDC	MC CKF
Finding Efficiency (prim)	0.951217	0.980567	0.951618
Finding Efficiency (all)	0.836262	0.951718	0.836184
Hit Efficiency (prim)	0.793219	0.965391	0.806137
Hit Efficiency (prim)	0.754524	0.94663	0.767135
Hit Efficiency (all)	0.763059	0.966506	0.776334
Hit Efficiency (SVD)	0.904544	0.941253	0.952157
Fake Rate	0.135205	0.115833	0.124333
Clone Rate	0.0711744	0.0142805	0.0504676
Hit Purity	0.929191	0.992893	0.933843

Combinatorial Kalman Filter from CDC + SVD to PXD

Final CKF setup

- Only start with tracks that actually do have SVD cluster attached (only CDC information is very unprecise)
- ”Simple” sector-map like structure for selecting only valid hits
- Make simple cuts on distance, χ^2 etc. (extrapolation and Kalman update are very precise for PXD as expected)
- Overlap Filter based on MVA method (e.g. χ^2 of full track, distance of PXD part compared to rest track etc.)
- All shown results include data reduction (although it is possible to run without it)

Proposed Workflow



Setup

Events for the following

Typical generic $\Upsilon(4S)$ with 1000 events using 15th campaign background, default data reduction.

PXD CKF Use CDC + VXDTF2 + Merger. Then add PXD hits with CKF (only on SVD parts).

VXDTF2 VXDTF2 6-layer tracking.

MC CKF As PXD CKF, but using MC information in the CKF filters.

CKF^2 As described before.

MC Matching

Comparing the performance by hit-matching doing on:

- All Hits
- PXD+SVD hits only

Performance on VXD only

CDC + VXDTF2	PXD CKF	VXDTF2	MC CKF	CKF^2
Finding Efficiency (prim)	0.939255	0.861416	0.942206	0.959198
Finding Efficiency (all)	0.904712	0.827025	0.908592	0.937831
Hit Efficiency (prim)	0.943487	0.935308	0.941277	0.942186
Hit Efficiency (prim)	0.886175	0.80569	0.886877	0.903743
Hit Efficiency (all)	0.936428	0.928516	0.934078	0.934711
Hit Efficiency (PXD)	0.898676	0.910465	0.894867	0.893895
Hit Efficiency (SVD)	0.953914	0.942511	0.952495	0.95235
Ratio with 100 Eff	0.702971	0.603236	0.705242	0.731927
Fake Rate	0.138063	0.179119	0.135138	0.154934
Clone Rate	0.00289	0.00345	0.00289	0.0202749
Hit Purity (PXD)	0.91384	0.869401	1	0.914572

Performance

CDC + VXDTF2	PXD CKF	VXDTF2	MC CKF	CKF^2
Finding Efficiency (prim)	0.946905	0.932807	0.948505	0.947105
Finding Efficiency (all)	0.828261	0.819022	0.830823	0.829658
Hit Efficiency (prim)	0.78024	0.77069	0.783382	0.795191
Hit Efficiency (prim)	0.738813	0.718905	0.743042	0.753129
Hit Efficiency (all)	0.750035	0.740053	0.752488	0.766414
Hit Efficiency (SVD)	0.852902	0.78214	0.847838	0.911699
Hit Efficiency (PXD)	0.835694	0.812261	**0.903872**	0.857076
Fake Rate	0.142922	0.177969	0.139526	0.137715
Clone Rate	0.0911653	0.0908829	0.092594	0.0635128
Hit Purity (SVD)	0.983385	0.97639	0.981929	0.970009
Hit Purity (PXD)	0.912553	0.857998	1	0.916476

Summary and Outlook

SVD case:

- comparable performance to VXDTF2, but merging is better
- still some room for improvements, but not that many (compared to MC truth)
- limited by CDC finding efficiency and track quality

PXD case:

- works good (but still not perfect) as expected, because VXDTF2 is not tuned for 6 layer tracking
- Very fast (3 ms/Event)
- Try to further increase purity (maybe use MVA here also to make smooth transition between purity and efficiency)

In my opinion, there is no reason to *not* include this in the November release.