Kalman-Filter Pattern Recognition and Fitting Status

Robert Johnson

U.C. Santa Cruz

June 21, 2021

Outline

- What is included in the Kalman package
- Changes since last year
- Pattern recognition overview
- Advantages of using the Kalman package
- 2019 MC performance
- 2019 data performance
- Conclusions

org.hps.recon.tracking.kalman

- pdf of LaTeX documentation
- “toy” MC test code (not loaded into hps-java)
- Linear helix fit
  - For initiating the Kalman Filter and the combinatoric pattern recognition
- Kalman-Filter track fit
- Pattern recognition
- Propagation of track and covariance in the non-uniform field
  - e.g. to the vertex and to the ECAL
  - Used in the newer track-calorimeter matching algorithms
- KalmanInterface.java: code needed to interface to hps-java
- KalmanPatRecDriver.java: driver code for production
- KalmanPatRecPlots.java: optional histograms used up to now for development work

Changes since one year ago

- Myriad technical details modified while trying to understand the pattern recognition in 2019 data. These are the major ones:
  - Switch from homebrew object-oriented matrix code to EJML procedural calls to speed up the code significantly (mainly by reducing the number of times that vectors and matrices get copied into new memory locations).
  - Cut out low pulse-height hits from the seed formation, to reduce combinations in events with large numbers of noise hits. This is to enhance speed.
  - Modifications needed to avoid negative covariance. The Kalman Filter has long been known to be susceptible to this problem, but I think remaining cases are confined to tracks that are no-good anyway.
- Vertex-constrained fitting.
  - This is not actually used, at least up to now.
- Some iterations with Norman to understand inefficiencies or badly-recognized tracks in 2019 data.
  - This is ongoing, and now with no teaching to do for the summer I think I can probably make more progress.

Pattern Recognition Overview

- The top and bottom trackers are treated separately.
- Two global iterations are done, with looser cuts in the second iteration, to try to give priority to the best tracks.
  - Loop over configurable set of “strategies”, each with 2 axial & 3 stereo layers.
    - These can now be set in the steering file. For example
      - <seedStrategy type="String">00ASBS0</seedStrategy> specifies the axial (A) detectors in layer 3, the stereo (S) detectors in layer 4, both axial and stereo (B) detectors in layer 5, and the stereo detectors in layer 6.
    - Loop over all combinations of hits from the 5 layers, doing a 5-parameter linear fit to each, to produce “seeds”. There are various cuts made here to reduce the number of combinations that have to be fit.
      - Avoid hits already on finished tracks (from the previous global iteration)
      - Avoid hits with very low signal
      - Cuts on slope and intercept of a line through the two axial layers
      - Maximum time range of the 5 hits
      - Skip a seed combination if all 5 hits are already contained in a track candidate
    - Cut on curvature, dz, and $d\rho$ of seeds following the linear fit.
    - Use Kalman Filter to propagate seeds, sorted by quality, to all layers and pick up hits, to produce “track candidates”.
      - The fit starts from the inner-most layer of the seed, using the linear fit at the initial guess.
      - First it works outward to layer 14 and then smooths back to the start of the seed, before filtering to layer 1 while picking up more hits.
        - The closest hit is picked up in a wafer only if the track extrapolation is not too far outside of the boundaries and the hit is not out of time w.r.t. the existing hits.
        - Low-signal hits are given decreased priority
        - There is no limit on hit sharing at this point (except that 100% cannot be shared!).

Pattern Recognition Overview

- Some cuts are made on the track candidate quality
  - Numbers of hits
  - Chi-squared (but first the worst hit is removed to see if that helps)
- The Kalman Filter fit is then iterated, starting from the innermost layer, filtering to the last layer, and then smoothing back to the beginning.
  - Again, an attempt is made to remove bad hits if the fit is really bad.
- At this point, exceptionally good candidates (gold plated) are moved into the output KalTrack data structure and their hits are marked as used and are removed from other, inferior candidates that were already found.
- Most candidates, however, are maintained in a list that includes lots of redundancy until all of the seeds have been exhausted.

- "Bad" candidates are thrown away, unless their hits are not being used by any other candidate, in which case there is an attempt to resurrect the bad candidate and keep it despite the fit being lousy.
- Every hit that appears on at least one candidates then is analyzed to determine which candidate should get to keep it.
  - It ordinarily is kept by just the "best" candidate and removed from others.
  - If the hit has a very low residual with two candidates, then it may be allowed to be shared, up to a maximum number of shared hits on a given candidate.
- Candidates that get too many hits removed to be viable are eliminated.

Pattern Recognition Overview

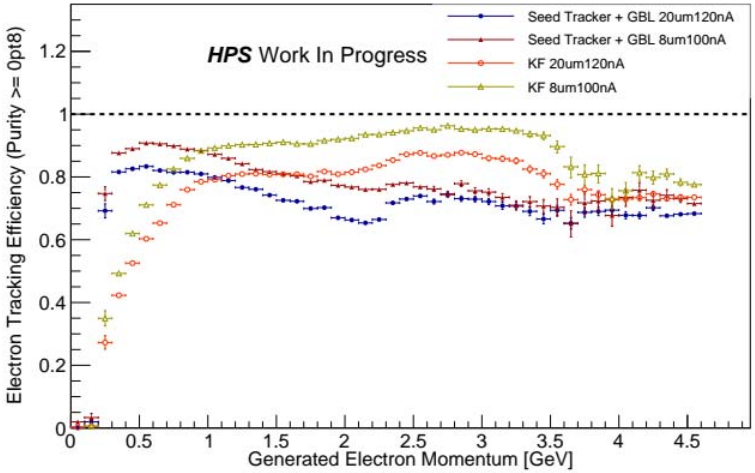
- Remaining track candidates then get made into a KalTrack instance, and the Kalman Filter fit is iterated.
  - There is some more logic to try to fix badly fitting tracks by removing hits, and there is also an attempt to add hits that may have been missed earlier.
  - An origin-constrained fit is attempted for 5-hit tracks, if those are allowed (normally not).
- The next global iteration is then executed.
  - No hits from the previous global iteration are allowed to be considered, unless they satisfy the tight constraints on hit sharing.
  - The second global iteration with looser cuts is mainly intended to pick up lower momentum tracks with few hits. Avoiding those in the first iteration gives priority to the more interesting tracks and also saves some CPU time.

Kalman Advantages for 2019 Processing

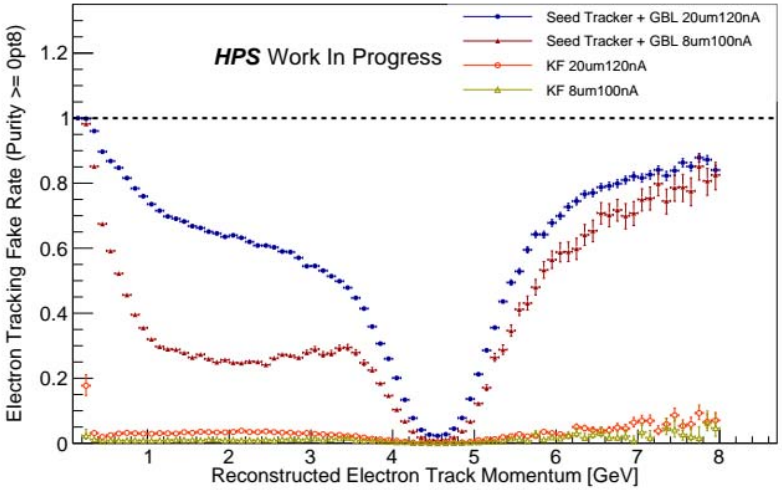
- Higher speed than with SeedTracker pattern recognition
  - I'm not sure why this is, being unfamiliar with the old code
  - The speed depends strongly on the number of hits per layer; it is important to get rid of junk hits beforehand.
- Less sensitivity to missing layers
  - 3D hits are never used, so odd numbers of silicon layers are okay.
- Fewer errors in Monte Carlo and higher efficiency
  - It's not yet clear whether this will hold up in 2019 data, based on Norman's recent analyses, but I hope to make improvements soon.

- *A lot of work went into making use of the non-uniform field map in the fitting, but this seems not to make a significant difference in the end.*

MC Performance (2019 tri-trig + beam)



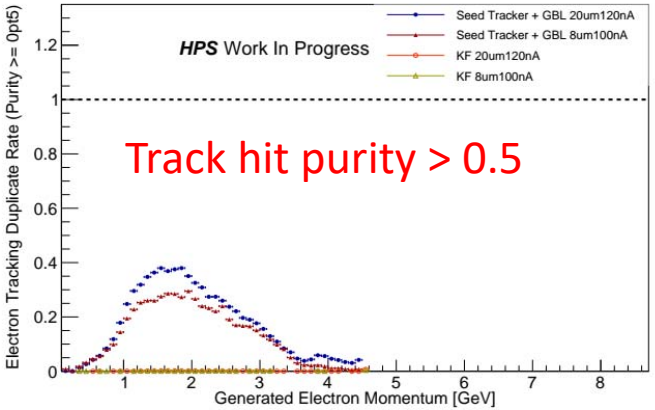
In MC the Kalman code achieves higher efficiency above around 1 GeV, but appears to be lower than the SeedTracker efficiency below that.

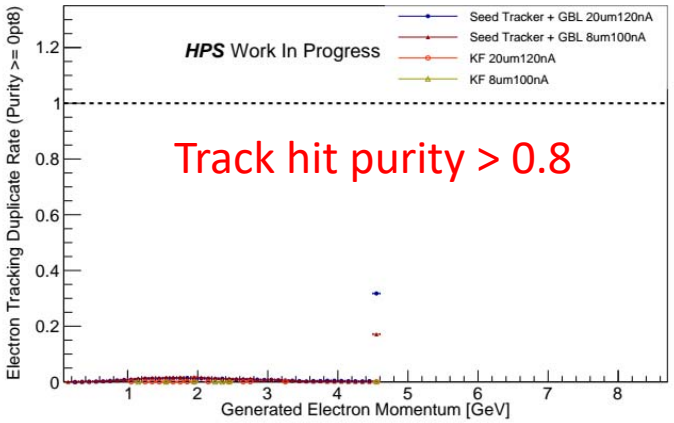


The rate of fake tracks is much lower in the Kalman code. That is especially true in the low-momentum region, making it unclear whether the higher SeedTracker efficiency in that region is meaningful.

These plots are from Alic Spellman. See his recent presentations for many more details

MC Performance (2019 tri-trig + beam)



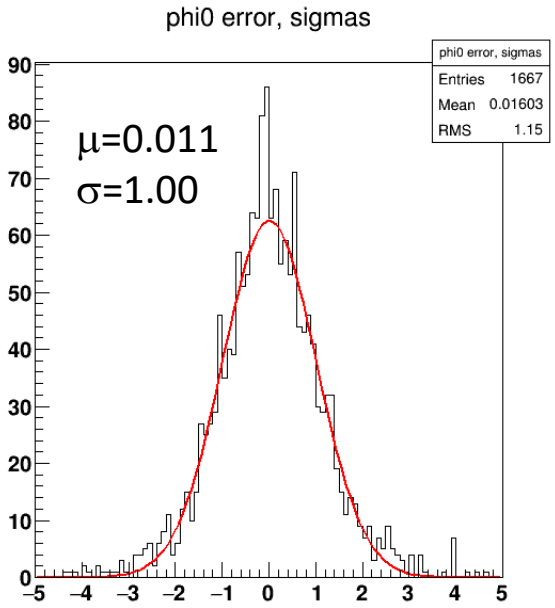


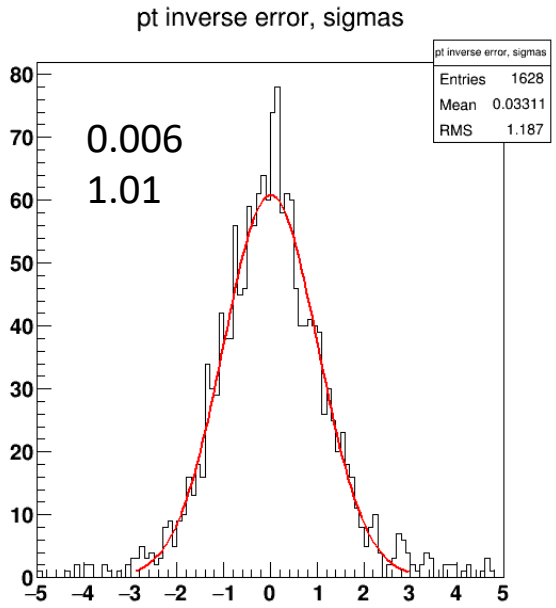
By design, the Kalman code produces almost no duplicate tracks. There is quite a large number of duplicates from SeedTracker, but almost all of them are tracks with bad matches to MC truth.

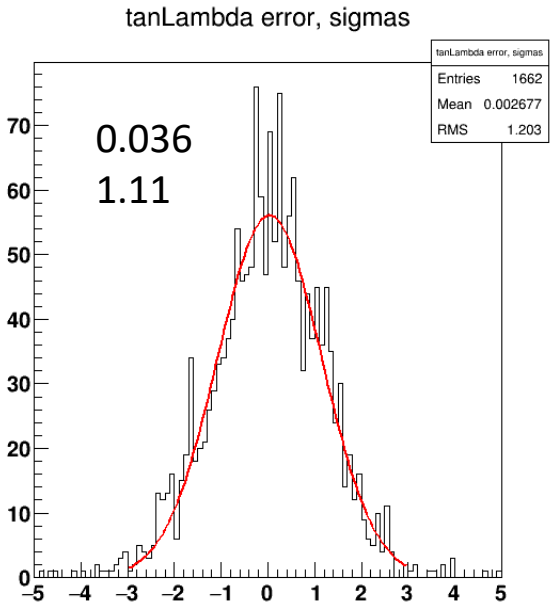
2019 Data Performance

- Norman has found lower efficiency for the Kalman code compared with SeedTracker, at least in some regions, by tagging events in data using the ECAL.
- We have iterated a couple of times with me looking in detail at some of the tracks found by SeedTracker but missed by Kalman. This has led to some improvements already.
- Of particular interest to me now are some events in which the SeedTracker track matches the ECAL cluster but the Kalman track curves away from it and misses.
  - Apparently in these cases the Kalman code chose the wrong hit in earlier layers.
  - I'm working to see if I can address this by giving more priority to candidates with lower curvature and/or with hits in the outer layers, even if their fit chi-squared isn't favorable.
    - *Usually the code found a candidate with the "correct" hit choices but then rejected it in favor of a lower-momentum candidate. Poor alignment, but also physical hard-scattering processes, can exacerbate this problem, because the code expects more scattering at lower E.*

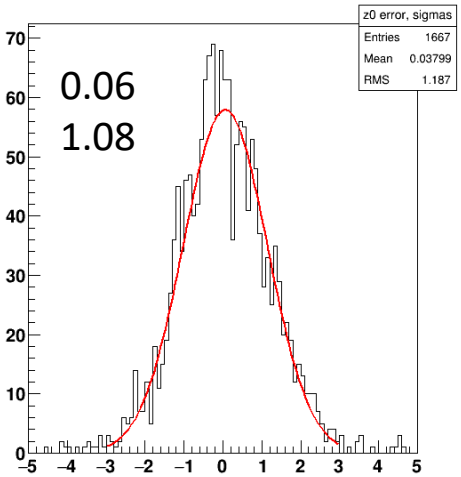
2019 Tri-Trig-Beam Helix Pull Distributions

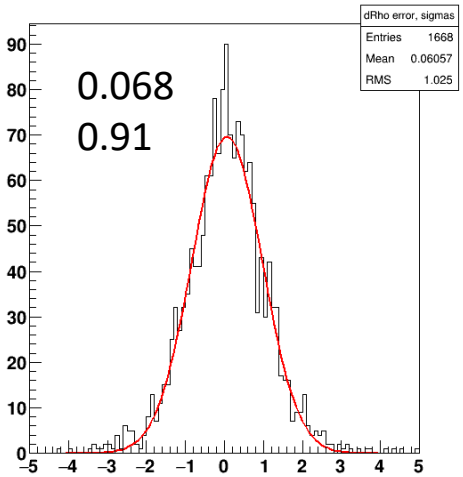




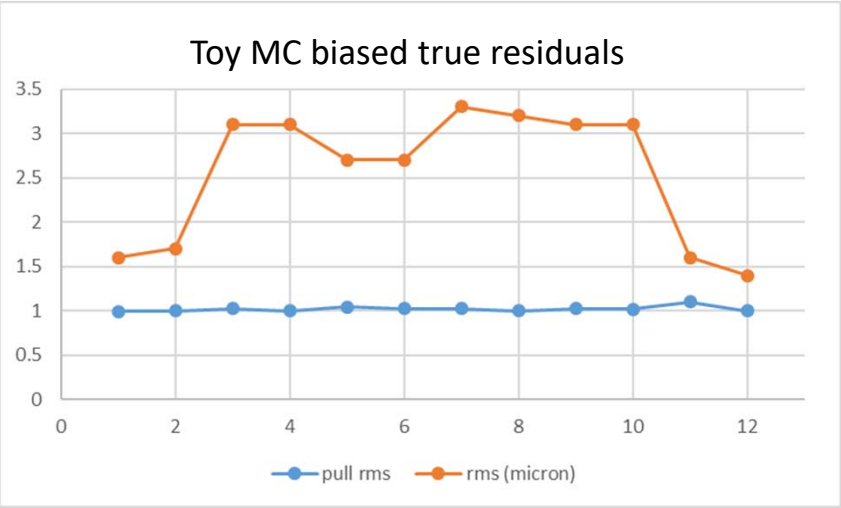


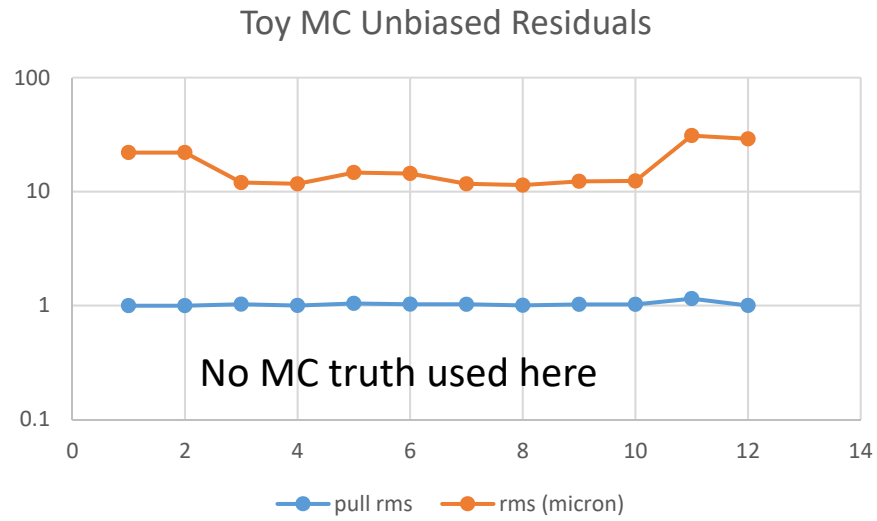
Alic sees some pull offsets in both Kalman and GBL in 2019 MC, depending on charge sign, so this needs to be looked at more.





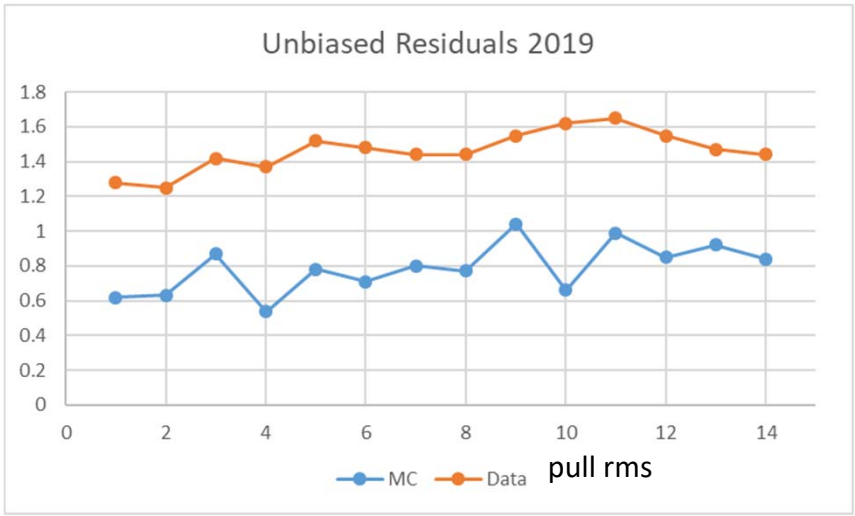
Fit Residuals, Biased and Unbiased





- The toy MC demonstrated (a year ago) that the Kalman-Filter math works correctly when truly Gaussian hit errors and MC scattering are present.
- The 2019 full MC seems to overestimate the errors a bit, whereas they are underestimated in 2019 data.

Note: the toy MC assumed the non-uniform HPS B-field, 6-micron hit errors, 12 silicon layers (as in 2016 data) and Gaussian multiple scattering from 0.32 mm thick silicon.



Conclusion

- The Kalman Filter fitting code is largely unchanged since a year ago, except for the move to the EJML matrix package.
- A lot of detailed improvements have gone into the pattern recognition, to improve both accuracy and speed.
- The code is, hopefully, nearly ready to use in processing 2019 data.
  - I would like first to try to tune the pattern recognition better to address the issues found by Norman.