Recent ACTS CKF and Vertexing developments

For EIC-ePIC tracking weekly

Andreas Stefl on behalf of the ACTS Core developers

Reminder

- There are two big packages in ACTS
  - Core: A collection of algorithms and components a track reconstruction framework can utilize
  - Examples: A reconstruction framework but not meant to be directly consumed by experiments
- The Examples can be useful for prototyping and early performance studies
  - One has to keep in mind the limitations (for instance simplified digitization)
  - This usually requires code changes (we try to be as flexible as possible, but it’s impossible to predict all use cases)
  - Don’t hesitate to clone the repository, make the necessary changes in a personal branch and eventually open a PR if you think your changes generalize

- Scope of this talk: Changes to ACTS since v31, specifically CKF and Vertexing
- A big proportion of the changes between v31 and v35 happened in Examples but depend on minor changes of the Core

Motivation

- Recent changes in CKF and Vertexing are rooted in CPU and physics performance studies and necessary improvements with ACTS and ODD
  - Track finding performance study for CTD 2023 (indico)
  - Vertexing performance study (closing in)

- At the same time, ATLAS Phase 2 Upgrade Track Finding is progressing rapidly
  - Goal is to be at least as “good” (physics and CPU) as the Legacy Athena Track Finding / Fitting
  - Closing in on achieving CPU performance, switching gears to look at physics performance

Overview of CKF changes since v31

Core

- Various control flow fixes
- Branch stopper improvements
- Separation of finding+fitting, smoothing, extrapolation
- Support for bidirectional track finding
- Various navigation simplifications
- Various CPU performance improvements

Examples

- Branch stopper #3098 (v34.1.0)
- Seed deduplication #3088 (v34.0.0)
- Stay on seed #3089 (v34.1.0)

Overview of Vertexing changes since v31

Core

- Track linearization with time
- Vertex seed finding with time
- Remove templating
- Minor edge case fixes

Examples

- Exercise finding and fitting with time
- Rewrite of `VertexPerformanceWriter`
- Filtering secondary vertices

CKF

CKF in General

- Conceptually not too complicated
  - Start propagation at initial parameters (Seed)
  - Stop at next sensitive surface
    - Query and select measurements
    - Branch in case of >1 measurements
    - KF update for each measurement
  - Continue propagation and search for each branch
  - Stop propagation at the end of the detector or on user request (`BranchStopper`)
- The ACTS implementation is rather complicated
  - CKF is modeled as an Actor which is triggered after each step inside the propagation
  - This occludes the control flow and does allow for simple local stack variables
  - Branching makes the situation even worse
- Recent developments modularized the CKF
  - Rather a component of a track finding implementation than a full blown track finding solution

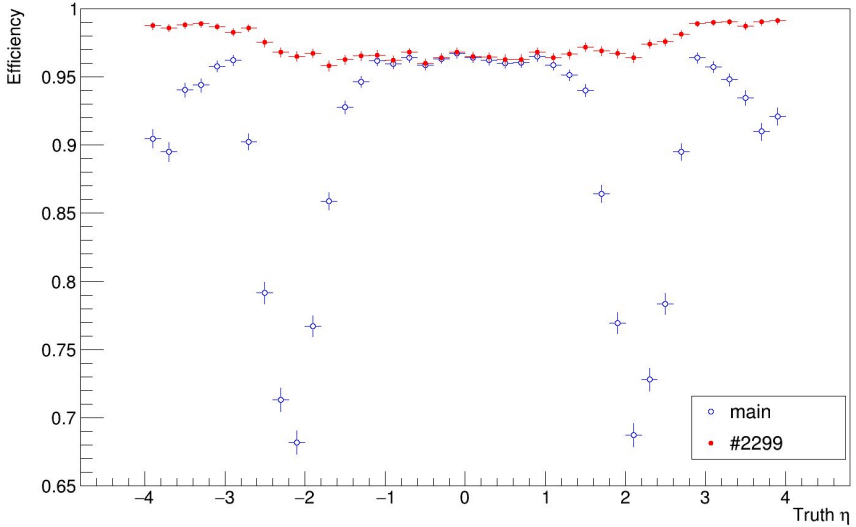
Branch Stopper Improvements

- Reminder: Branch stopper allows the user to stop the track finding early
  - This can save significant amount of CKF and memory
  - An easy and reasonable criteria is number of holes and/or outliers
- Branch stopper itself is a long standing feature

- Related changes
  - refactor!: Give CKF BranchStopper access to TrackState #2757 (v32.0.0)
  - refactor!: Refactor CKF branch stopper to allow stop and keep tracks #3102 (v35.0.0)

Control flow fixes for CKF

- Flow of the algorithm is complicated and hard to follow
  - Bugs are hard to isolate and identify
  - Bugs usually don’t show up as error logs / exceptions / signals but rather as low physics performance
- Small changes in the code can have drastic, non-local effects
  - fix: Fix CKF finalization #2299 (v27.2.0)
- Other related changes
  - refactor: Refactor Core CKF finalization #3188 (v35.1.0)
  - refactor: Common function to store tracks in Core CKF #3164 (v35.0.0)



Separation of CKF

- Motivation: CKF is quite complex and does multiple things sequentially
  - This is not mapped by simple control flow but signaled via flags being turned on
- Modularize CKF into finding+fitting, smoothing, extrapolation
- Smoothing and extrapolation can be used by other algorithms like KF
- Simplifies control flow of the CKF and therefor reading logs and debugging
- Related changes
  - refactor!: Remove smoothing and extrapolation from core CKF #2722 (v34.0.0)
  - feat: Method to add components to an existing track state #3075 (v34.0.0)
  - refactor: Do not allocate smoothed track state components in CKF #3086 (v34.0.0)
  - feat: Allow extrapolation from filtered states in TrackHelpers #3078 (v34.0.0)

Bidirectional Track Finding

- Limitation: CKF can only find tracks in one direction
  - Given a seed the CKF will not necessarily find all hits left by a particle
  - Especially the innermost hits are crucial for optimal IP resolution
- A series of changes have been proposed and merged to allow running the CKF in two directions
  - feat: Two-way CKF example #3066 (v34.0.0)
  - refactor!: Remove smoothing and extrapolation from core CKF #2722 (v34.0.0)
  - refactor: Make MaterialInteractor noise mode independent from direction #2723 (v31.1.0)
  - feat: Target surface for filtering phase of CKF #2319 (v28.1.0)
- The track EDM was extended to allow stitching two tracklets together
  - feat: Add (optional) Jacobian reversal to Track #2571 (v30.3.0)
- A track smoothing utility was implemented to re-smooth stitched tracks
  - refactor!: Remove smoothing and extrapolation from core CKF #2722 (v34.0.0)

Measurement Selector

- Generalization of `MeasurementSelector` to provide more flexibility #3198 (v35.1.0)
- Motivation: Default workflow of CKF was not CPU efficient
  - Source link creation -> temporary track state creation -> calibration -> selection
  - Calibration of all measurements is wasteful if only a fracture is selected afterwards
  - Showed up in Phase 2 ITk Track Finding profiling
- Low level delegate for the measurements selection can be provided to customize all these steps
- A default implementation of this newer delegate is included which provides the previous workflow
- Note: Current implementation of the `MeasurementSelector` flags outliers not holes
  - feat: Stop branches based on number of outliers in Examples track finding #3116 (v34.1.0)

Various CPU Performance Improvements

- perf: Improve stepper performance with sqrt over hypot #3137 (v34.1.0)
- refactor: Remove blockMult from boundToCurvilinearTransportJacobian #3127 (v34.1.0)
- refactor: Align stepper benchmarks #3133 (v34.1.0)
- perf: Cache particle hypothesis #3151 (v35.0.0)
- perf: Try fast pow for EigenStepper step size scaling #3153 (v35.0.0)
- perf: Remove positive definite check from smoothing #3128 (v35.0.0)
- fix: Fix EigenStepper step upscaling #3152 (v35.1.0)
- feat: SympyStepper #3150 (v35.1.0)

Navigation

Navigation Simplifications

- Reminder: Navigation is part of the propagation and decides on the step size and when we hit a surface
- Apart from its original purpose the navigation and its state leak into the stepping, actors and aborters
  - Some aborters even overwrite the step size which can lead to all sorts of funny behavior
  - Some actors communicate “end of propagation” by setting flags inside the navigation state
  - All of this is non-local and occludes the control flow
  - Leads to bugs that are hard to isolate and fix
- A series or changes have been proposed and merged to improve the situation
  - refactor: Remove target reached from standard aborters #2912 (v33.0.0)
  - refactor: Simplify layer handling in Navigator #3190 (v35.0.0)
  - refactor: Backport navigation rewrite changes #2846 (v35.1.0)
  - refactor: Remove target volume estimation from Navigator #3217 (v35.1.0)

Alternative Navigator Implementations

- Motivation: Default navigation is optimized for performance and correctness
  - But we cannot compare the results to anything
  - Idea: Implement a simple navigation schema that focuses just on correctness

- feat: Add TryAllNavigator #2849 (v35.2.0)
  - Interects all surfaces in a volume before each step
  - Does not rely on any acceleration structure
  - Reduces chance for missing surfaces which can be reduced to practically 0 by limiting the step size

- feat: Add TryAllOverstepNavigator #2850 (v35.2.0)
  - Very similar to `TryAllNavigator` but intersects backwards which should give a better ray vs path approximation

Gen2+ Navigation

- Gen2+ simplifies tracking geometry significantly by replacing layers with volumes
- Simplifies the navigation: one logical state and its transitions are removed

- As soon as we can built the ODD we can start validating the propagation
  - Surface sequence / hit position / track finding
  - This should give practically the same results as the Gen1 version
- Usually this involves some edge case hunting which can be tedious
- CPU performance has to be monitored in parallel to physics performance

Vertexing

Vertexing in General

- Long standing component in ACTS
- Already in production for ATLAS Run 3 primary vertex reconstruction
- In general, the existing algorithms are built for primary vertex reconstruction
- The underlying components should generalize to secondary vertex reconstruction

- Recent developments focused on vertex finding and fitting with time
- A large scale refactor was undertaken to remove templating from the implementation
- Recent vertexing performance studies with ACTS+ODD required improvements of the Examples

Vertex Finding and Fitting with Time

- Motivation: Future trackers may come with precise time measurements
  - Can be used for pile-up rejection which ultimately improves CPU and physics performance
  - ACTS can already use time measurements for track finding and fitting
- Vertex seeding with time allows to split vertices close in space but far in time
  - refactor: Refactor AdaptiveGridTrackDensity #2830 (v32.0.0)
  - feat: time vertex seeding #2460 (v30.2.0)
- Vertex fitting with time combines the time measurements of the tracks
  - fix: cross-covariance matrix in FullBilloirVertexFitter #2771 (v32.1.0)
  - feat: Add time to impact point estimation #2414 (v30.1.0)
  - feat: extract impact parameters and their covariance matrix #2464 (v30.1.0)
  - refactor: Adding time to HelicalTrackLinearizer #2179 (v29.0.0)
  - refactor: revisit FullBilloirVertexFitter #2196 (v27.2.0)
- There was/is also more recent work to improve on the seeding

Major contributors: Felix, PF

Remove Templating from Vertexing

- The interfaces were polluted with templates but there was no benefit of that
  - Vertexing function calls are usually not hot ➜ function pointers are a valid choice for CPU performance
  - Function pointers are easier to make configurable at runtime
  - Compile time was unnecessarily long with templates
- A series of changes removed the templating and replaced them with virtual inheritance / delegates
  - refactor!: Vertex InputTrack becomes concrete type #2876 (v33.0.0)
  - refactor!: Untemplate Vertex #2877 (v33.0.0)
  - refactor!: Untemplate VertexInfo and VertexingOptions #2878 (v33.0.0)
  - refactor!: Remove input_track_t template parameters #2880 (v33.0.0)
  - refactor!: Use Delegate for parameter extraction #2881 (v33.0.0)
  - refactor!: Use BasePropagator interface in vertexing #2886 (v33.0.0)
  - refactor!: Use Delegate for track linearizers #2946 (v33.0.0)
  - feat!: Add IVertexFinder interface, use in vertexing #2948 (v33.0.0)
  - refactor!: Hard-code vertex fitter, finder + density combinations #2952 (v33.0.0)
  - refactor!: ImpactPointEstimator moves to cpp file #2971 (v33.0.0)
  - refactor!: Move and Grid Density finders to cpp #2973 (v33.0.0)

Major contributors: Paul

Refactor / Rewrite VertexPerformanceWriter

- Motivation: Unmaintained code, very difficult to understand, change and debug
  - Also missed crucial features like handling of secondary vertices, vertex classification (clean, merged, split, fake), sumPt2, pile-up density, ...
- A series of changes tried to improve the situation gradually until this resulted in a complete rewrite
  - feat: save vertex seed #2885 (v33.0.0)
  - feat: Write sumPt2 in VertexPerformanceWriter #2929 (v33.0.0)
  - feat: Use Barcode for vertex ID in VertexPerformanceWriter #2970 (v33.0.0)
  - feat: Write truth matching in VertexPerformanceWriter #2969 (v33.1.0)
  - feat: Use truth vertex EDM in Examples #2998 (v33.1.0)
  - refactor: Use track weight for vertex truth matching in Examples #3024 (v33.1.0)
  - feat: Vertex classification for VertexPerformanceWriter in Examples #3044 (v34.1.0)
  - feat: Add primary vertex density and contamination to VertexPerformanceWriter in Examples #3085 (v34.1.0)

Filtering Secondary Vertices

- Motivation: Studying primary vertexing performance requires filtering for vertices and tracks from the primary interaction
  - Displaced vertices have to be identified and removed for truth based vertex finding and fitting
  - If a secondary vertex is reconstructed we can filter it out
- Problem: Python8 does not distinguish between prompt and displaced vertices
- Examples flagged everything as secondary which did not have bit identical position origins
- Related changes
  - fix: Flag for secondary vertex labeling in Pythia8ProcessGenerator #2989
  - feat: Pythia8 label secondary vertices based on proximity in Examples #3048

Summary and Outlook

Summary

- Most changes happened in the Examples
  - But they are built on top of smaller changes in Core
- Track finding model seems to hold up for ITk
  - CPU performance looks quite good, switching gears to validate and improve physics performance
- Vertexing types and interfaces changed substantially, functionality is the same
- Navigation remains a challenging topic

Outlook

- v36.0.0 is on its way
  - Harmonize propagation options for stepper and navigator #3181
  - Progress on Gen3 geometry #3065
  - Rewrite of `BoundaryCheck` #3170
  - Model CKF branches as tracks #3161

- Thinking further
  - Detector region constraint propagation / track finding / track fitting
  - Navigation and track finding with surface bounds tolerance
  - Convergence on tracking geometry with Gen3
  - Higher level track finding component for Core
  - Dense propagation with finding and fitting

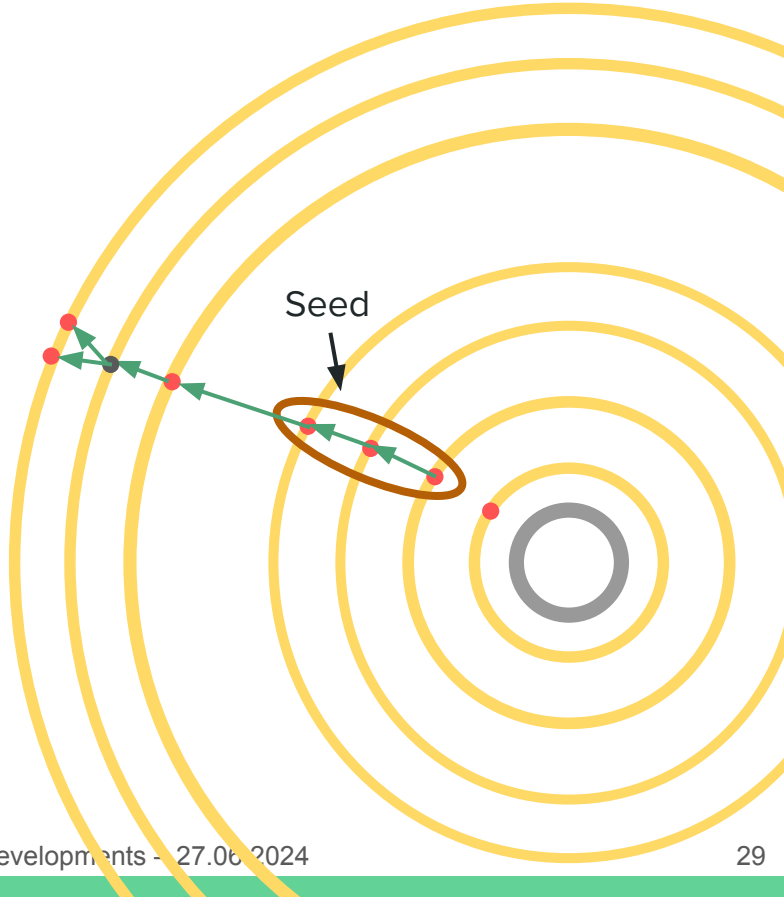
Backup

Evolution of Acts `CombinatorialKalmanFilter`

- Produces track candidates based on initial track parameters
- Code is quite involved
  - Relies on various other components in Acts: Geometry, Magnetic Field, Stepping, Navigation, Propagation, Track EDM
  - CKF runs inside the `Propagator` - control flow occluded, difficult to debug
  - Branching is handled during propagation - control flow occluded, difficult to debug
  - Many customization points: `SourceLinkAccessor`, `Calibrator`, `MeasurementSelector`, `BranchStopper`
- Previous to #2722 it also included smoothing and extrapolation
  - Splitting the CKF into smaller components improved readability and maintainability
- Small changes to the code had big, unexpected implications in the past #2299
- To be more user friendly we might want to consider another track finding interface on top of the current one, which handles bidirectional finding and extrapolation

Acts-based track finding

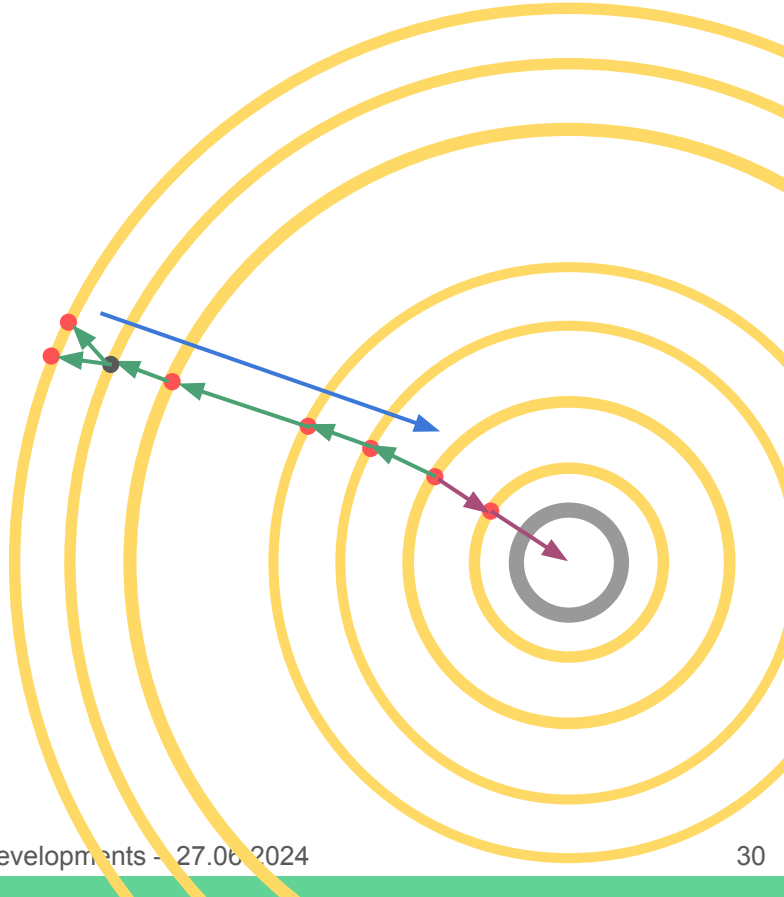
1. Track finding starts with estimated track parameters at the innermost cluster (in case of pixel seed)
2. Extrapolate, select measurements, branch, repeat



- Branch stopper allows user to stop finding early
  - Current implementation of the `MeasurementSelector` requires to cut on holes and outliers to be efficient #3116

Acts-based track finding

3. Get tracks from this outwards pass, smooth them →
4. Start inwards pass with smoothed params at innermost measurement state →
5. Terminate at perigee surface, extract parameters →
6. Reverse the inner tracklet, stitch them together
7. Output tracks



refactor: Refactor SurfaceReached aborter #2603

-

- Affected components: Core, Propagation, KF, CKF, GSF
- Released with v31.1.0

refactor: Make MaterialInteractor noise mode independent from direction #2723

-

- Affected components: Core, Extrapolation with material interactions
- Released with v31.1.0

fix: Fix CKF pathlimit abort #2744

-

- Affected components: Core, CKF
- Released with v31.2.0

feat!: Wire time to spacepoints and seeds #2829

-

- Affected components: Core, Seeding, Space Point Formation
- Released with v32.0.0

refactor!: Give CKF BranchStopper access to TrackState #2757

-

- Affected components: Core, CKF
- Released with v32.0.0

refactor: Use common direction transform Jacobian #2782

-

- Affected components: Core, Jacobian and covariance transport
- Released with v32.0.0
  - Reverted in v32.0.1 due to output changes in Athena

Others released with v32.0.0

- refactor!: Remove CovarianceTransport #2781
- refactor!: Remove navigator layer bounds check option #2851
- refactor: Reuse JacobianEngine #2789
- refactor: Move actor state into propagator state #2552
- refactor: Refactor navigation #2768
- refactor: Remove BoundaryCheck to bool conversion #2860
- refactor: Remove resetState from navigator #2808

fix: DirectNavigator causes incorrect propagation finalization #2913

-

- Affected components: Core, KF with Direct Navigator
- Released with v32.0.2

fix: Initial track param covariance inflation #2295

-

- Affected components: Examples, Track Finding
- Released with v32.1.0

fix: Propagate initial vertex time variance for AMVF w/o time #2936

-

- Affected components: Core, AMVF
- Released with v33.0.0

fix: Fix AMVF find single track vertices #2931

-

- Affected components: Core, AMVF
- Released with v33.0.0

fix!: Fix vertex finding for seeds with z=0 #2917

-

- Affected components: Core, Vertexing, IVF, AMVF
- Released with v33.0.0

refactor: Assert to be on surface for surface functions with free param input #2932

-

- Affected components: Core, Surface, Propagation
- Released with v33.0.0

refactor: Central truth matching for tracks in Examples #2904

-

- Affected components: Examples
- Released with v33.1.0

fix: Robust perigee propagation in HelicalTrackLinearizer #2930

-

- Affected components: Core, Propagation, Vertexing
- Released with v33.1.0