A Common Tracking Software (ACTS) Project

Paul Gessinger

CERN / JGU Mainz

31.07.2019 - IRIS-HEP topical meeting
Outline

■ Track reconstruction
  ▶ What is it?, Challenges

■ The Acts project
  ▶ Design goals and basic principles, project structure

■ Components:
  ▶ Geometry description
  ▶ Numerical integration, propagation, magnetic field access
  ▶ Multiple events in flight and context handling
  ▶ Track fitting: the Acts Kalman Filter and Event Data Model

■ Status and outlook
Track reconstruction
Track reconstruction

Seeding

Track finding

Track fitting
Track reconstruction

- Turn hits on sensors into trajectories of particles that produced them
- Multi-Stage process:
  - Pattern recognition to reduce combinatorics
  - Exploration of all compatible measurements
  - Selection of best candidates and precise fit $\rightarrow$ best estimate of trajectory
- Remove overlap between different solutions: ambiguity resolution (crucial for performance)
Challenges

- This is routinely the largest CPU consumer in event reconstruction
- Pileup affects performance significantly:

  ![Diagram](image.png)

- Looking at *HL-LHC* era and beyond: tracking needs to improve!
The Acts project
The Acts project

- Based on the ATLAS tracking software\cite{2}
- Got started to enable R&D in faster cycles, test thread-safety concepts

Contributions come mostly from ATLAS people, but there is increasing involvement from CEPC, Belle-2, FCC-hh and more

Make ACTS const-correct (again)

1. Drop all occurrences of "mutable" in ACTS (ACTS-256)
2. Get rid of static variables and class members (ACTS-309)
3. Replace pointer-to-mutable by pointer-to-const where appropriate (ACTS-208)
4. Drop incorrect usage of pointer-to-const where the target is modified (ACTS-269)
5. Clean up ugly interfaces discovered in ACTS by introducing const-correctness (ACTS-270), and in particular rework geometry building (ACTS-276)
Basic design goals

- Provide a toolbox with experiment-independent components
- Should allow building (possibly) experiment-specific applications like seeding, track finding, vertex finding
- Minimal dependencies (e.g. no required ROOT dependency)
- Minimize dynamic memory allocation
- Be thread-safe out of the box (thread local state)
- Limit (ideally eliminate) virtual inheritance: concepts!
- Rigorously tested, use continuous integration
Tool interface design

- Constant configuration as config struct at construction
- Invocation configuration as *options* struct at invocation
- Thread-local state as arguments: no *mutable* members

```cpp
struct MyTool {
    struct Config {
        double value{42};
    }
    struct State{};
    MyTool(Config cfg)
        : m_cfg(std::move(cfg)) {}
    void doSomething(State& state) const {
        /* ... */
    }
    Config m_cfg;
};

int main() {
    MyTool::Config cfg;
    MyTool tool{std::move(cfg)};
    MyTool::State state;
    tool.doSomething(state);
}
```
Project structure

- **acts-core**: main library
  - Contains tools and components
  - Doesn’t assume anything about event-processing framework

- **acts-framework**: small GaudiHive-inspired event processing framework
  - Event-level parallelism for testing
  - Has *generic* geometry, TGeo and DD4hep plugins

- **acts-fatras**: Acts-based fast track simulation
  - Can be used to create scenarios for testing and validation

- Licensed under **MPLv2**
Geometry and navigation
Geometry modeling and navigation

- Concepts from ATLAS: fully detailed geometry for precise simulation, simplified **tracking geometry** with only sensitive sensors for faster navigation and propagation
- Individual sensors are grouped into **layers**
- Layers are binned to allow fast retrieval of compatible surfaces
Geometry modeling and navigation

ACTS documentation

Geometry module

DiscLayer with two rows of modules

2-dimensional \([r,\phi]\) binning (equidistant)

ACTS documentation

Geometry module

x

y

Surface objects ordered in a binned SurfaceArray object.

representing surface

inner approach surface (including envelope clearance)

outer approach surface (including envelope clearance)

A. Salzburger

x-y view of a CylinderLayer (with planar detection elements)
Geometry modeling and navigation

- Concepts from ATLAS: fully detailed geometry for precise simulation, simplified **tracking geometry** with only sensitive sensors for faster navigation and propagation
- Individual sensors are grouped into **layers**
- Layers are binned to allow fast retrieval of compatible surfaces
- Layers are grouped into **volumes**
Navigation

- Optimize navigation for **speed**
- Idea: pre-resolve transitions as much as possible
- Volumes are *glued* together using *boundary* surfaces
- Navigation works volume to volume, then layer to layer

![Diagram](image-url)

**Figure 9:** Illustration of a sample navigation following a particle trajectory through three fully connected volumes A, C, and B. The volume boundary surfaces hold information about the attached volumes respectively volume rays, such that a simple projection onto the surface normal vector enhances a step by step navigation scheme.

5. The ATLAS TrackingGeometry

The building of the ATLAS TrackingGeometry requires the parsing of the full ATLAS detector description GeoModel. Specific information about detector structures has to be accessed for this task, therefore the sub-detector constructions introduce dependencies on these associated software repositories. The new ATLAS tracking realm, on the other hand, has been designed to be subsystem independent and the same concept is also applied to the content of the TrkDetDescr container package. To integrate the TrackingGeometry into this software structure, the building of the sub-detector TrackingGeometry instances has been outsourced into the associated detector repositories, while still using only common classes from the Tracking repository that do not refer to specific detector technologies. Various different concrete implementations of an IGeometryBuilder interface class, each for one sub-detector or for different detector setups, are retrieved at run-time and are steered by a central AlgTool, the so-called GeometryBuilder (located in the TrkDetDescrTools package). Details of the building process and the structure of the sub-detector geometries are described in the following sections.

5.1 The Inner Detector TrackingGeometry

The reconstruction geometry for the Inner Detector is created by the InDetTrackingGeometry-Builder AlgTool that makes use of several other AlgTool classes for the creation of Layer and Volume objects that are contained by the ID. The building process is evoked by calling the PixelLayerBuilder and the SCT LayerBuilder, respectively, that parse the associated sensitive GeoModel detector description source for the pixel detector and the silicon strip detector (SCT). The overall dimensions of the silicon detector are determined and Layer objects created, while the sensitive detector elements are sorted in binned arrays with a fast access mechanism, Figure 10 illustrates this simplified model for a SCT endcap disk. The ID TrackingGeometry automatically adapts to different layouts and misalignment configurations. The Layer objects for the Transition Radiation Tracking (TRT) are, in general, not built by parsing the sensitive detector elements. This is due to the fact that the material in the TRT is almost continuously distributed and can be — for performance reasons — simplified to a few layers in the reconstruction geometry. Modeled layers for condensed material information are inserted in the corresponding TRT volumes to represent the inert material of the TRT detector.

A UML sequence diagram for the creation of the Inner Detector TrackingGeometry is shown in [8].

---

8 E.g. the number of pixel barrel and endcap layers differs for the ATLAS-Rome-Initial and ATLAS-CSC layouts. The PixelLayerBuilder adopts through parsing of the full description to the actual number of layers, and the PixelVolumeBuilder encloses the layers dynamically.

9 For the use in FATRAS, the TRT straws are grouped and ordered on layers similarly to the pixel or SCT detector. As this operation requires the parsing of about 300,000 elements this is omitted for the standard reconstruction job setup.
Numerical integration
Numerical integration

FORTRAN

CALL THEFLSP ( IPR(3) ) ! track propagation through
! precision detectors
CALL THEFILP ( 1 ) ! Fill output track bank without TRT
CALL THEFLST ( IPR(4) ) ! Track propagation through TRT
CALL THEFILT ! Fill output track bank with TRT
CALL THEBRE ( IBREM ) ! Brem. fit possibility investigation
IF ( IBREM.NE.0 ) GO TO 20 ! Repeat fit with brem. conditions
    GO TO 10 ! Go to next track candidate
CALL THERRO ( IER ) ! Test errors list
CALL THELOOK ! Tracks comparison
Numerical integration

**FORTRAN++**

```fortran
double H1[3] = {f[0]*PS2,f[1]*PS2,f[2]*PS2};
double A5 = 2.*A4-A[0] ;
double B5 = 2.*B4-A[1] ;
```

Paul Gessinger 31.07.2019 - IRIS-HEP topical meeting
// Update the track parameters according to the equations of motion
state.stepping.pos +=
    h * state.stepping.dir + h2 / 6. * (sd.k1 + sd.k2 + sd.k3);
state.stepping.dir += h / 6. * (sd.k1 + 2. * (sd.k2 + sd.k3) + sd.k4);
state.stepping.dir /= state.stepping.dir.norm();

- Matrix operations abstracted into operators
- Eigen should produce near-optimal code for operations
- Readable → more maintainable
Propagation in ATLAS

- Uses Runge-Kutta-Nyström integration\cite{1, 3}
- Lots of \textbf{virtual method calls} and \textbf{dynamic memory allocation}
- Was packaged in Acts as ExtrapolationEngine as baseline (now gone)
Propagation in Acts

- Generalize *optional* components into **Actors** and **Aborters** (e.g. MaterialActor)
- Keep component structure for *required* components: **Stepper** and **Navigator**
- All components are template parameters: **no virtual calls**
- ATLAS’ different propagators $\rightarrow$ *integration-term extensions* to our main integrator **EigenStepper**
Propagation

- **EigenStepper**: primary Acts integrator
- **AtlasStepper**: transcription of ATLAS numerical integrator
- Infrastructure changes enable significant speedup:

- Covariance transport and *STEP* mechanism\[^{4, 5}\] implemented
- Possibility for alternative / specialized implementations

[7]

\[^{4, 5}\]: Paul Gessinger 31.07.2019 - IRIS-HEP topical meeting
Magnetic field access
Magnetic field access

- Observation: magnetic field is queried **very frequently** during propagation
- Most of the time: **very little distance** between queries
- Mitigation: keep field cell in thread-local cache, interpolate linearly from corners
Timing information

- Timing is integrated into numerical integration
- Data structures allow for time measurements naturally (see KF later)
- Tests indicate no negative performance impact

\[ \vec{x} = (x, y, z, t, T_x, T_y, T_z, q/p) \]

with time, without time
Multiple events in flight and context handling
Handling of context

- Some aspects of the detector change over time: e.g. magnetic field, temperatures, calibration and **alignment**
- Especially non-trivial when multiple events in flight
- Need to be able to communicate what the current context is
Example: alignment handling

- Use context objects: geometry context, calibration context, magnetic field context
- Create at event level, pass down to where needed
Example: alignment handling

Current ATLAS code:

```cpp
const Amg::Transform3D & SiDetectorElement::transform() const
{
    std::lock_guard<std::recursive_mutex> lock(m_mutex);
    if (!m_cacheValid) updateCache();
    return m_transform;
}
```
Example: alignment handling

- Use **context objects**: geometry context, calibration context, magnetic field context
- Create at event level, pass down to where needed
Example: alignment handling
Acts implementation in Athena:

```cpp
const Acts::Transform3D&
ActsDetectorElement::transform(
    const Acts::GeometryContext& anygctx) const
{
    const ActsGeometryContext* gctx
        = std::any_cast<const ActsGeometryContext*>(anygctx);
    const ActsAlignmentStore* alignmentStore = gctx->alignmentStore;
    const Transform3D* trf = alignmentStore->getTransform(this);
    return *trf;
}
```
Example: alignment handling

- Acts implementation: lock free, no synchronization needed!
- Flexible, doesn’t change for different sources
- Want to extend concept to magnetic field and calibration
Track fitting
Kalman Filter in Acts

- Kalman Filter is implemented as an *Actor*
- Gets called automatically during regular propagation
- Can update direction, uncertainties after filtering step
- Aim to minimize heap allocation
Kalman Filter aspects

- Runtime performance: So far no direct comparison, comparable test setup is not trivial
- Study of numerical performance (see here and here by X. Ai)
Event Data Model
Kalman Filter EDM: Measurements

- Local sensor frame is same as measurement frame
- Measurement mapping function $H$ is a projection matrix
- Input to KF: lightweight SourceLink object
- Is turned into Acts measurement by calibrator (context aware)

$$\vec{r} = (l_0, l_1, \phi, \theta, q/p, t)^T$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$
Kalman Filter EDM: Measurements ii

template <typename SL, ParID_t... P>
class Measurement;
template <typename SL>
class FittableMeasurement;

- Knows dimensions at compile-time: **fixed-size** matrix operations
- Projector is known from type
- FittableMeasurement type-erases by tagged union
- Dispatch on concrete type at runtime
std::visit([&](const auto& calib) {
    const projection_t& H = calib.projector();
    gain_matrix_t K = pred_covariance * H.transpose() * 
    (H*pred_covariance*H.transpose() + calib.covariance()).inverse();
    filt_parameters = pred.parameters() + K * calib.residual(pred);
    filt_covariance = 
    (CovMatrix_t::Identity() - K * H) * pred_covariance;
    parameters_t filt(/* ... */);
    meas_par_t res = calib.residual(filt);
    ts.parameter.chi2 = (res.transpose() * ((meas_cov_t::Identity() 
    - H * K) * calib.covariance()).inverse() * res).value();
    ts.parameter.filt = std::move(filt);
}, *ts.measurement.calibrated);
Event Data Model

- Current EDM still uses dynamic allocation in a number of places (track parameters)
- Goal: further minimize heap allocation as much as possible
- Take inspiration from ATLAS’ xAOD (column wise storage, collection based)
- Currently ongoing: explore EDM replacement for Kalman Filter
  - Investigating if that can be used for other fitters as well
  - Investigating compatibility for interfacing to other execution environments
  - Look into performance characteristics
Status and outlook
Status

- Propagation, single Kalman Filter, seed finder developed and usable
- Time propagation implemented and tested
- So far: tests mostly standalone
- Tests of ATLAS geometry (ID + first tests for calorimeter modeling and navigation)
- Demonstrated multi-threaded execution with alignment
Work in Progress

- Multi-component propagation to be used for Gaussian Sum Filter (WIP MRs: #582, #588 by J. Zhang)
- Components for vertex fitting, finding are coming together (see this and more by B. Schlag)
- R&D into machine learning for vertex finding (B. Schlag) ambiguity resolution (N. Cinko)
- Perform material mapping in ATLAS using new infrastructure in Acts
Plans

- Develop better test integration in ATLAS (i.e. same interface but Acts under the hood)
- Compare Acts KF to ATLAS KF
- Rewrite navigation to make it more adaptable to various navigation approaches
- Start work on a Combinatorial Kalman Filter implementation (some preliminary tests exists)
- Get to the point where we can demonstrate a full track reconstruction chain
References


